Implementation of a Trusted I/O Processor on a Nascent SoC-FPGA Based Flight Controller for Unmanned Aerial Systems

Akshatha J Kini

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Cameron D. Patterson, Chair
Paul E. Plassmann
Lynn A. Abbott

February 21, 2018
Blacksburg, Virginia

Keywords: SoC, UAV, FPGA, ZYNQ, MicroBlaze, autopilot, ArduPilot, ArduPlane, GPS, Mission Planner, Sensor, Vulnerabilities, Mailbox

Copyright 2018, Akshatha J Kini

Implementation of a Trusted I/O Processor on a Nascent SoC-FPGA Based Flight Controller for Unmanned Aerial Systems

Akshatha J Kini

(ABSTRACT)

Unmanned Aerial Systems (UAS) are aircraft without a human pilot on board. They are comprised of a ground-based autonomous or human operated control system, an unmanned aerial vehicle (UAV) and a communication, command and control (C3) link between the two systems. UAS are widely used in military warfare, wildfire mapping, aerial photography, etc primarily to collect and process large amounts of data. While they are highly efficient in data collection and processing, they are susceptible to software espionage and data manipulation. This research aims to provide a novel solution to enhance the security of the flight controller thereby contributing to a secure and robust UAS. The proposed solution begins by introducing a new technology in the domain of flight controllers and how it can be leveraged to overcome the limitations of current flight controllers.

The idea is to decouple the applications running on the flight controller from the task of data validation. The authenticity of all external data processed by the flight controller can be checked without any additional overheads on the flight controller, allowing it to focus on more important tasks. To achieve this, we introduce an adjacent controller whose sole purpose is to verify the integrity of the sensor data. The controller is designed using minimal resources from the reconfigurable logic of an FPGA. The secondary I/O processor is implemented on an incipient Zynq SoC based flight controller. The soft-core microprocessor running on the configurable logic of the FPGA serves as a first level check on the sensor data coming into the flight controller thereby forming a trusted boundary layer.

Implementation of a Trusted I/O Processor on a Nascent SoC-FPGA Based Flight Controller for Unmanned Aerial Systems

Akshatha J Kini

(GENERAL AUDIENCE ABSTRACT)

UAV is an aerial vehicle which does not carry a human operator, uses aerodynamic forces to lift the vehicle and is controlled either autonomously by an onboard computer or remotely controlled by a pilot on ground. The software application running on the onboard computer is known as flight controller. It is responsible for guidance and trajectory tracking capabilities of the aircraft.

A UAV consists of various sensors to measure parameters such as orientation, acceleration, air speed, altitude, etc. A sensor is a device that detects or measures a physical property. The flight controller continuously monitors the sensor values to guide the UAV along a specific trajectory.

Successful maneuvering of a UAV depends entirely on the data from sensors, thus making it vulnerable to sensor data attacks using fabricated physical stimuli. These kind of attacks can trigger an undesired response or mask the occurrence of actual events. In this thesis, we propose a novel approach where we perform a first-level check on the incoming sensor data using a dedicated low cost hardware designed to protect data integrity. The data is then forwarded to the flight controller for further access and processing.

Dedication

To my family

Acknowledgments

First and foremost, I would like to thank my advisor, Dr. Cameron Patterson for giving me this fantastic opportunity to work on such a challenging and exciting topic. Being a certified pilot himself, his passion and remarkable insight into the world of avionics is highly inspiring and motivational. His able guidance, consistent support and sound advice during the course of this research will remain as a solid foundation in my future professional career. As international students, we have to provide a lot of paper work during graduate school life. I'm very grateful to Dr. Patterson for always promptly obliging to my requests for any letters, recommendations and signatures. Lastly, I thank him for his tremendous patience while reviewing my thesis document. I would like to sincerely thank Dr. Lynn Abbott and Dr. Paul Plassmann for participating on my academic advisory committee.

I would like to thank the support staff of Newman Library and Graduate School, especially Matthew Grice for always being so patient with my queries and clarifying them in the best possible way. I also express my thanks to Dr. Michael Hsiao, Dr. Peter Athanas, Prof. Gino Manzo, Prof. Kenneth Schulz, Dr. Patterson (again) and Prof. Robert Martin whose classes I thoroughly enjoyed. A special mention to Devaprakash who helped me understand the various interfaces associated with a UAV. Graduate school experience at Virginia Tech was not just about long hours spent working on assignments, but also a wonderful place to make some amazing friends for life. Shobal, Shobek, Sagar, Pinto, Kumari, Trushit and Bharani have truly enriched my life in many ways and I will always cherish the times spent with them.

I'm grateful to my parents for their immense sacrifices and for giving me and my sister a happy and a carefree childhood. My world would not have been the same if not

for my younger sister, Apoorva, who has been a strong pillar of support all my life. My special thanks to my childhood friends Seema and Karthik for more than two decades of close bonding. Their friendship has been invaluable to me during every stage of my life. A special mention to my dear cousins, Ashwinanna and Anitha for coming down to visit me at Virginia Tech every year for some quality family time and giving me a well deserved break from the rigor of academics. I owe a debt of gratitude to my two mentors dear Harish Sir and Vinod Sir who introduced me to the fascinating world of FPGAs and for inculcating in me a love for digital electronics. I have to thank Ashish, my brother in law for encouraging me to apply to Virginia Tech in the first place and for being an exemplary role model as a graduate student. My sincere thanks to Suchetha, who has been my go-to person as a stress-buster these two years. I'm also extremely thankful to my parents in law for all their love and support and for being with me during every step of this incredible journey.

Last but not the least, I'm eternally grateful to my better half, Aniketh, who has been the bulwark of my life come rain or shine. I cannot thank him enough for giving wings to my aspirations, for showing me that nothing is impossible in life if you put your heart and soul into it and for always bringing out the best in me. I humbly bow to Lord Almighty without whose blessings I would not have been here.

The fulfilling academic experience, long hours spent in the Graduate Study Room, developing a kinship with fellow graduate students also burning the midnight oil, admiring the glorious sunsets across the drill field while studying, finding peace and comfort in the vicinity of duck pond, becoming good friends with American and Chinese team mates, enjoying the vagaries of nature in Blacksburg, being at the receiving end of random acts of kindness are some of my unforgettable memories of this beloved place which will stay etched in my heart forever. I shall remain a Hokie for life.

Special thanks to the team at Aerotenna for providing the OcPoC board and for their kind cooperation and support during the course of this research. This material is based

upon work supported by the Naval Air Warfare Center - Aircraft Division (NAWCAD), contract N00421-16-2-B001. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NAWCAD.

Contents

List of Figures

List of Tables

List of Abbreviations

AES	Advanced Encryption Standard
AFCS	Automatic Flight Control System
AMBA	Advanced Microcontroller Bus Architecture
APU	Application Processing Unit
ARM	Advanced RISC Machines
AXI	Advanced eXtensible Interface
BD	Block Diagram
BRAM	Block RAM
BSP	Board Support Package
CAN	Controller Area Network
CPU	Central Processing Unit
DDR	Dual Data Rate
DPRAM	Dual Port RAM
ELF	Executable and Linkable Format
FF	Flip Flop
FIFO	First In First Out

FPGA	Field Programmable Gate Array
GPIO	General Purpose Input Output
GPS	Global Positioning System
HAL	Hardware Abstraction Layer
HDL	Hardware Description Language
HMAC	Hash Message Authentication Code
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
IP	Intellectual Property
JTAG	Joint Test Action Group
LMB	Local Memory Bus
MAVLink	Macro Aerial Vehicle Link
MEMS	Micro-Electro-Mechanical Systems
MIO	Multiplexed Input Output
OcPoC	Octogonal Pilot on Chip
OS	Operating System
PID	Proportional Integral Derivative

PL	Programmable Logic
PPM	Pulse Position Modulation
PS	Processing System
PWM	Pulse Width Modulation
QSPI	Quad SPI
RAM	Random Access Memory
RC	Remote Control
RE	Reverse Engineering
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RTOS	Real Time Operating System
SCA	Side Channel Attack
SD Card	Secure Digital Card
SDK	Software Development Kit
SEM	Scanning Electron Microscopes
SEP	Secure Enclave Processor
SoC	System on Chip
SPI	Serial Peripheral Interface
TEM	Transmission Electron Microscopes

TIPR	Trustworthy Isolation of Privilege and Resources
UART	Universal Asynchronous Receiver Transmitter

Chapter 1

Introduction

As innovation and technology continues to soar beyond unchartered territories, this era has spurred mankind to envision problems that we don't even know exist yet. Nuclear power, personal computer, aeroplane, rocketry, to name a few, are some of the top innovations of the 20th century. One of the major offshoots of technological innovations in the last few decades is the emergence of UAV. It has influenced our society in awe inspiring ways and its power continues to grow ever so rapidly.

The demonstration of a remotely controlled boat by Nikola Tesla in 1898 led to the birth of remotely controlled UAV, which was invented by Archibald M. Low two decades later. UAVs have gone from being curiosities to practical autonomous systems which are as big as small airliners or the size of insects. Since the last few decades, UAVs have been used in a number of fields, from agriculture to meteorology and from research to military warfare. The list of applications for UAVs keeps growing.

UAS is an all encompassing system that encapsulates the aircraft or the UAV, the ground based controller and the communication system connecting the two. These systems, like any other computer based system, are vulnerable to different types of cyber attacks and hence some degree of cyber security is required. These attacks can be broadly categorized into 4 groups:

- Hardware attacks - The UAV components are compromised.

- Wireless attacks - Attacks are carried out through one of the wireless communication channels.

- Senor spoofing - False data is passed through the onboard sensors of the UAV autopilot.

- Software attacks - Introduction of malicious code either into the autopilot's or ground station's software.

In this research a new generation of flight controller hardware, called OcPoC is introduced to overcome the limitations of current technologies in flight controllers.

1.1 Contributions

The following contributions were made during the course of this research:

- A new generation of SoC-FPGA based flight controller is successfully implemented, realized and tested in an academic environment.

- The various challenges associated with a complex Linux based embedded system are explored. The work required a through understanding of hardware/software interfaces and a wide spectrum of protocols ranging from simple UART to MAVLink.

- OcPoC (Octogonal Pilot on Chip) was transformed from a complex SoC-FPGA board into a compact flight controller ready to easily be adapted for future research and development in the domain of flight controller algorithms and UAV security.

1.2 Thesis Organization

The remainder of the thesis is organized as follows. Background information and a general introduction to the different terms and classifications in UAS are discussed in Chapter 2. Various vulnerabilities and security issues faced by current UAS technologies are outlined in Chapter 3. Chapter 4 provides an understanding of how flight controllers have evolved over time and the way they are embracing new technology to overcome the limitations of the old technology. Chapter 5 delves into the details of the SoC-FPGA based flight controller and the work involved in board setup and execution to get it up and running. Chapter 6 and 7 describe the proposed approach to integrate a secondary I/O controller with the OcPoC and the end results. Chapter 8 concludes with what was achieved and provides future directions for this research.

Chapter 2

Background

UAS are increasingly being used for civil applications such as remote sensing, aerial photography, pipeline monitoring etc. In military applications, UAS are useful for solid intelligence gathering. These systems are based on cutting edge developments in aerospace technologies. The vast potential for research and development make UAS a highly interesting topic of study. This chapter gives an overview of the architecture of UAS followed by its operation. This is followed by a brief description of the communication and control protocols used in UAS and the types of UAVs used.

2.1 Architecture of a UAS

A UAS is comprised of the following elements, as illustrated in Figure 2.1.

1. UAV aircraft : Pilot-less vehicle which houses the onboard processor, sensors, actuators and other physical components which together make up a UAV.

2. Autopilot : The device used to guide an aircraft without direct assistance from the pilot. It consists of an onboard computer responsible for processing sensor data and the implementation of the flight controller application.

3. Magnetometer : Used for measuring direction by sensing magnetic north.

4. GPS : Used to determine the geographic location.

5. Airspeed/Altimeter : Used to measure air speed and altitude.

6. Power System : Responsible for providing power to the entire UAV.

7. Inertial Measurement Unit (IMU) : Used to measure the linear and angular motion of the UAV.

8. Actuator/Servo: Receives commands from autopilot and moves the control surfaces.

9. Ground Station : A software application running on a ground-based computer used for configuring and communicating with a UAV via wireless telemetry.

10. Telemetry modules : They are made up of a pair of radio devices that transmit and receive data between the ground station and the UAV. One of them is located onboard the UAV and the other one is plugged to the ground station. The radio devices have to be "paired" together in order to communicate.

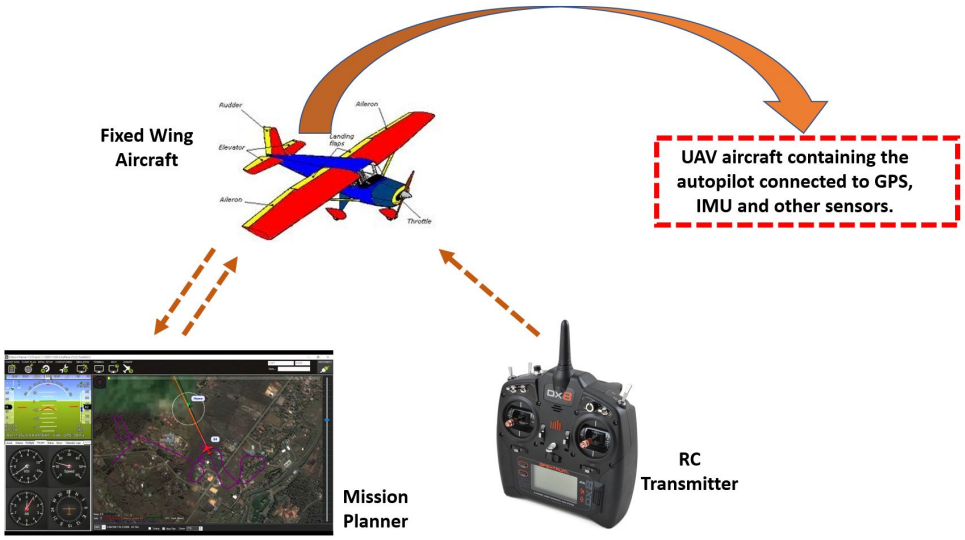


Figure 2.1: UAS Block Diagram

2.2 Operation of a UAS

The onboard processor, housed inside the aircraft, is the nerve center of a UAV. Its basic function is to provide stabilization and control to all actuators. A single sensor is not sufficient to control a UAV and hence several sensors are used. By combining the measurements from all the sensors and using complex mathematical algorithms such as Kalman filtering, Dijkstra's algorithm for Path Finding, the flight controller software application called an autopilot is responsible to keep the aircraft stable.

The flight controller reads in the sensor data with respect to the UAV's current position and orientation and outputs control signals to the actuators. The control signals depend on the way the UAV is being controlled, either through human operator on ground or using the Ground Station software. The flight controller may be programmed to fly according to a predetermined waypoint using the Ground Station. The different types of flight modes are:

- Stabilize – the UAV will fly straight and level, unless commanded otherwise by the RC transmitter
- Loiter – the UAV will circle around a particular waypoint
- Auto – the UAV will fly a series of waypoints
- Manual – the UAV will pass through all RC commands as they are

2.3 Ground Control Station (GCS)

The GCS, depicted in Figure 2.2 is a software application running on a ground-based computer or a laptop used to communicate with the UAV via wireless telemetry. It serves

as a "virtual cockpit" by displaying the UAV's position and performance and can also be used to control a UAV during a real time flight operation by uploading mission commands and setting parameters. GCS is often also used to monitor the live video streams from a UAV's cameras. There are different GCS applications available such as Mission Planner, APM Planner 2, MAVProxy, Tower (DroidPlanner 3), AndroPilot, MAVPilot, iDroneCtrl and QGroundControl. **Mission Planner** is the GCS software used in this project.

Figure 2.2: Mission Planner application

2.4 Micro Aerial Vehicle Link (MAVLink)

MAVLink is a communication protocol mainly used by GCS to talk to the UAV. It is a lightweight, header only packet protocol that allows up to 256 UAVs to communicate on the same frequency band [14]. MAVLink messages can be of two types: information requests and mission commands. Information requests are used by GCS to retrieve firmware data such as vehicle type, initial GPS readings etc, after powering on the board. Mission commands are used during the flight to control the trajectory of the UAV. MAVLink messages are sent

as a stream of bytes encoded by Mission Planner to the autopilot board either via wireless telemetry or via USB. Both interfaces are mutually exclusive. MAVLink message structure is shown in Table 2.1.

Table 2.1: Structure of the MAVLink message

message length = 17 (6 bytes header + 9 bytes payload + 2 bytes checksum)
6 bytes header:
0. message header, always 0xFE
1. message length (9)
2. sequence number – rolls around from 255 to 0 (0x4e, previous was 0x4d)
3. System ID - what system is sending this message (1)
4. Component ID- what component of the system is sending the message (1)
5. Message ID (e.g. 0 = heartbeat)
Variable Sized Payload (specified in octet 1, range 0..255)
Checksum: For error detection

2.5 Principle of Operation of a Servo

Servo motors are used to regulate the control surfaces of a UAV. The stability of the aircraft depends, among other factors, on the appropriate commands sent to the servo actuators. A servo motor works on the principle of Pulse Width Modulation (PWM). Its angle of rotation is controlled by the duration of an applied pulse to its control pin. A servo motor consists of a DC motor which is controlled by a variable resistor (potentiometer) and a gear system. The servo motor can be moved to a desired angular position by sending the PWM signal on the control wire. The servo reacts to the PWM signal by changing its speed or direction. A pulse of width varying from 1 to 2 milliseconds in a repeated time frame is sent to the servo at a rate of around 50 times per second. The width of the pulse determines the angular position. For example, as depicted in Figure 2.3, a pulse of 1 millisecond moves the servo towards 0°, while a 2 milliseconds wide pulse would take it to 180°. The pulse

width for intermediate angular positions can be interpolated accordingly. Thus a pulse of width 1.5 milliseconds will shift the servo to 90°.

A sequence of such pulses (50 in one second) is required to be passed to the servo to sustain a particular angular position. When the servo receives a pulse, it can retain the corresponding angular position for the next 20 milliseconds. So one pulse in every 20 millisecond time frame must be sent to the servo.

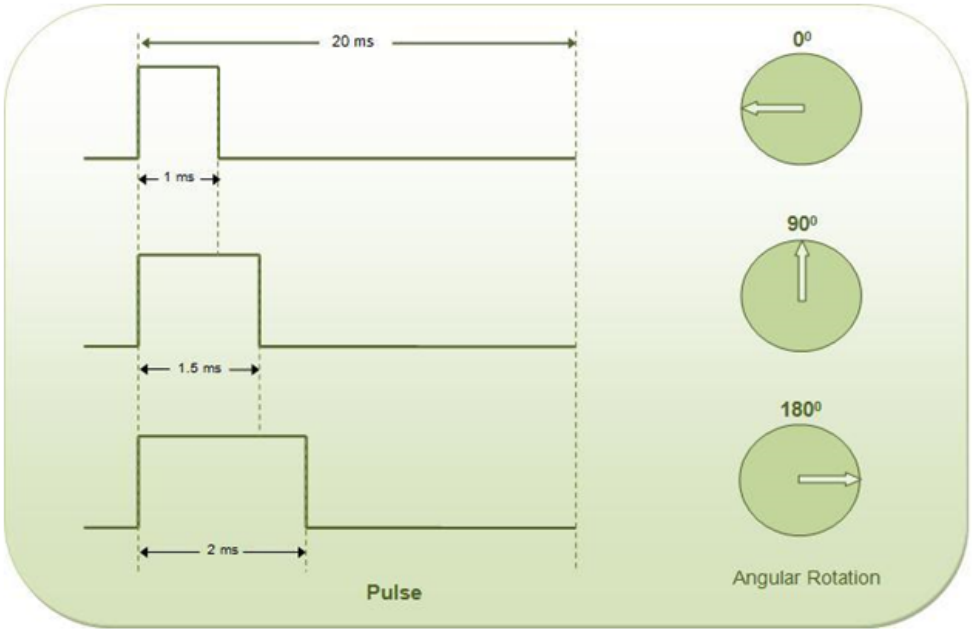


Figure 2.3: PWM operation

2.6 Pulse Position Modulation (PPM)

PPM multiplexes several PWM signals and transmits them on a single wire. Multiplexing helps in reducing the number of wires for RC receiver input. PWM is employed by servos whereas RC transmitters and receivers use PPM sending and receiving control inputs.

2.7 Types of UAV aircraft

UAVs are generally classified into two main categories, namely the fixed wing and rotary wing UAVs.

2.7.1 Fixed wing UAVs

Fixed wing UAVs, as shown in Figure 2.4, consist of a rigid wing that has a predetermined airfoil which enables flying by generating lift caused by the UAV's forward airspeed. A propeller powered by an electric motor generates the thrust needed to gather airspeed. The UAV control system comprises of ailerons, a throttle, an elevator and a rudder which are the hinged sections of the flying surface. They allow the UAV to freely rotate around three perpendicular axes and intersect at the UAV's center of gravity. The throttle controls the amount of air flowing into the engine, elevator controls pitch (lateral axis), ailerons control roll (longitudinal axis) and the rudder controls yaw (vertical axis).

Figure 2.4: Fixed wing UAV

Advantages of a fixed wing UAV are:

- Simpler structure vis-a-vis other UAV aircraft models.

- Ensures more efficient aerodynamics that enable longer flight durations at higher speeds.
- Can carry greater payloads for longer distances which helps to carry bigger and better sensors.

A major disadvantage of a fixed wing aircraft is the need for a runway for takeoff and landing.

2.7.2 Rotary wing UAVs

Rotary wing UAVs, as shown in Figure 2.5, consist of multiple rotor blades that revolve around a fixed mast, known as a rotor. Rotary wing UAVs come in different configurations which vary in the number of rotors. For example: 1 rotor (helicopter), 3 rotors (tricopter), 4 rotors (quadcopter), 6 rotor (hexacopter), 8 rotors (octocopter), etc. The rotating action of the rotor blades produce the required airflow over their airfoil to generate lift. They are controlled using the variation in thrust and torque from the rotors. For example, to move downwards, the rear rotors of a quadcopter produce more thrust than the rotors in the front, this enables the rear of the quadcopter to raise higher than the front thus producing a nose down attitude.

The biggest advantage of rotary UAVs is the ability for takeoff and land vertically. This allows the user to operate with in a smaller vicinity with no substantial landing/take off area required. On the flip side, rotary wing aircraft involve greater mechanical and electronic complexity which translates generally to more complicated maintenance and repair processes.

Figure 2.5: Rotary wing UAV

Chapter 3

Vulnerabilities of UAS

UAVs are used in a wide range of missions such as border surveillance and reconnaissance. They are expected to be reliable and autonomous. The amount and kind of information that UAVs process make them an extremely interesting target for espionage, manipulation and cyber attacks.

A good understanding of a system's vulnerabilities gives an insight into how well it is protected. A system vulnerability is defined as a particular characteristic of a system that increases the probability of malfunction due to specific incidents. The specific incident can be an external attack or an internal software triggered event. Depending on the degree of severity of the malfunction, the outcome can vary anywhere between a complete loss of control of a UAV to a minor error in calculation of a sensor value. Some of the different threats encountered by a UAS are:

1. Reverse Engineering
2. Side Channel Attacks
3. GPS Spoofing
4. Sensor Spoofing

Reverse Engineering, GPS spoofing and sensor spoofing are known as active attacks since they involve making changes to the components whereas side channel attacks are passive form

of attacks which only monitor the components. Each of the above threats are described in the upcoming sections.

3.1 Reverse Engineering (RE)

RE refers to the process of information retrieval from proprietary binary programs or hardware chips in order to understand their composition and inner workings. RE is often associated with illegitimate actions such as Intellectual Property (IP) infringement [28], weakening of security functions, or disclosure of necessary information for injecting hardware trojans [28].

In the context of UAVs, unauthorized direct access to UAV's components enable hardware reverse engineering. An entire UAV can be torn down to examine the structure and see how it is manufactured or to identify internal components and boards. To further probe the hardware circuits, advanced imaging laboratories can help in exposing the internal gates and interconnections of the chips using optical Scanning Electron Microscopes (SEM) or Transmission Electron Microscopes (TEM) [26].

Apart from reverse engineering hardware components, wireless protocols used between a UAV and a ground system can be decrypted using automatic protocol reverse engineering tools [23]. Using such tools, data transmitted through MAVLink protocol can be deciphered [17]. A sophisticated hacker who is able to reverse engineer the communication protocol between UAV and GCS, can control navigation, override all commands from the real operator or even crash it to the ground.

3.2 Side Channel Attacks (SCAs)

Electronic circuits are inherently leaky. They produce emissions that make it possible for an attacker to deduce how the circuit works and what data it is processing. Since these emissions do not play an active role in the operation of the circuit itself, they are simply a byproduct of a working device and the use of these radiations to hack into the systems is known as SCAs.

SCAs take advantage of the changes in processing behavior that take place at different times during algorithm execution. Attackers use a range of side channel properties, such as heat generated, power consumed, or execution time. For embedded systems where the attacker has access to the hardware, heat and power represent the most important sources of leaks, although timing-based attacks are likely to increase on multitasking and multiprocessor systems where the interactions between existing applications can be studied to track behavior[24]. An SCA set up is shown in Figure 3.1. The power consumption of the board is captured using a current probe and is displayed on an oscilloscope.

Since SCA is a passive attack, the attack itself cannot be detected. It can only be mitigated by avoiding the generation of side channel leakage. Countermeasures involve algorithm-level techniques such as randomization where the order of operations on the data is constantly changing. Architecture-level countermeasures are independent of algorithm and involve specialized hardware [24]. They are characterized by a large overhead in area, power and performance.

3.3 GPS Spoofing

Military GPS signals are encrypted to prevent unauthorized use whereas civil GPS signals are freely accessible to all and therefore not secure [30]. GPS Spoofing is an active

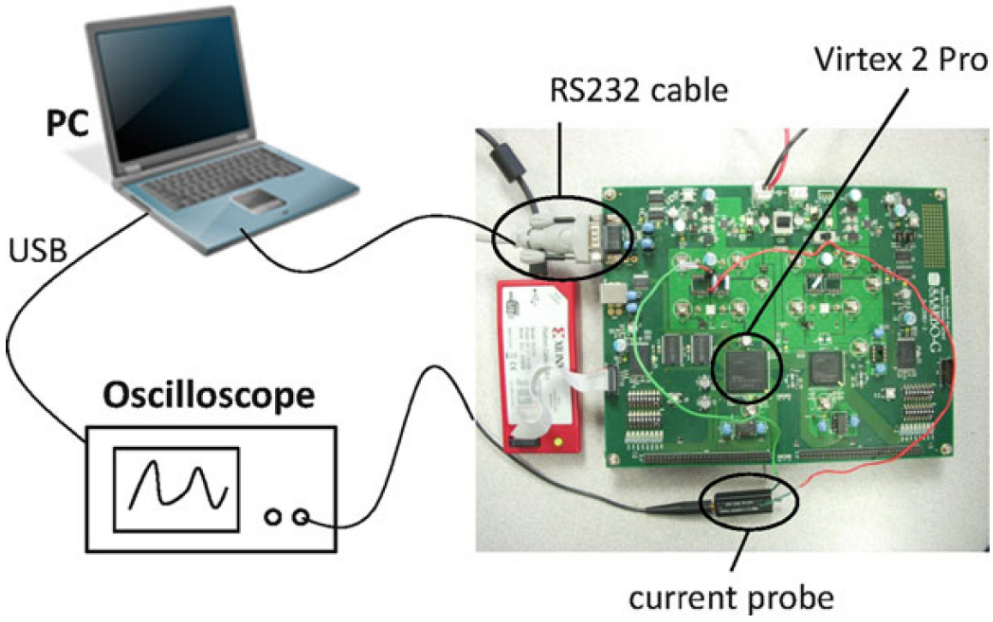


Figure 3.1: Side channel analysis set up [24]

attack in which transmitters masquerade as GPS satellites and broadcast incorrect GPS signals such that the receiver calculates erroneous position and time estimates. A research team at University of Texas, Austin successfully demonstrated the capture and control of a UAV via GPS spoofing[18].

Blocking and jamming are two other ways to attack a GPS receiver [29]. Blocking prevents the satellite signal from reaching the antenna of the GPS receiver. Jamming intentionally introduces interference to overload the receiver circuitry and thereby disable it.

To eliminate the threat of GPS spoofing, the following measures can be used: (1) hardware techniques, which require multiple atomic clocks [9], (2) signal processing techniques, which develop correction algorithms from raw GPS data [32], (3) mixed hardware and signal processing techniques, which detect signal anomalies such as wrong time, suspiciously low noise, artificial spacing of signals, etc [9].

3.4 Sensor Spoofing

Sensors are classified according to the type of reference they use. For example, a GPS sensor uses an external reference (i.e. a GPS satellite) where as an Inertial Navigation System (INS) sensor uses internal references such as acceleration or angular rate [16]. The potential risks of using a sensor depend on the characteristics of the sensor, the importance of the property being measured/observed and the checks in place for detecting spoofed or false sensor values.

An example of a sensor spoofing attack is described next. A UAV has multiple sensors such as gyroscopes, accelerometers and barometers. A gyroscope is a device which measures angular velocity. Micro-Electro-Mechanical Systems (MEMS) gyroscopes are preferred in UAVs since they are light weight and inexpensive. Resonating frequency of a MEMS gyroscope is higher than the audible frequency range. However it has been found that some gyroscopes do resonate at audible as well as ultrasonic frequencies. A commercially available audio amplifier was used to attack a target drone which could not ascend or recover its control due to the gyroscope resonating at unintended frequencies under the acoustic attack [25].

Chapter 4

Evolution of Flight Controller

An autopilot is a device for steering a vehicle such as an aircraft or a ship. In the context of UAVs, an autopilot is a highly reliable avionics system for advanced control of a UAV. ArduPilot is a class of open source autopilots commonly used as a flight controller in UAVs. Pixhawk is a microcontroller-based board running the ArduPilot application. To overcome the limitations of Pixhawk, a new generation of flight controller called OcPoC is introduced. The following sections describe how flight controllers have evolved from the concept of autopilot to the design of OcPoC.

4.1 Autopilot

An autopilot is an Automatic Flight Control System (AFCS) used to control a UAV [12]. An autopilot mainly consists of a processor which continuously collects data from various sensors such as GPS and accelerometers, performs complex calculations and compares them to a set of predetermined states. A control mode is a setting made from the GCS where mission parameters are configured, or from a human operator controlling the UAV remotely. Depending on the difference in measurement between sensor inputs and the desired states, the autopilot is able to determine if the UAV is flying as required.

Proportional Integral Derivative (PID) is the most common type of control algorithm used in autopilots to control speed, altitude and orientation. The core algorithm is

implemented as a closed control loop which involves a PID controller reading the sensor data and calculating the rate at which the servos should rotate. The algorithm essentially tries to correct the "error" or the difference between the measured variable (i.e. the sensor data) and a desired set point (i.e. a control mode value) by adjusting the speed of the servos for every loop iteration [7].

Implementing a good autopilot is a challenging task. Autopilots usually run on resource constrained embedded hardware due to size, cost and energy limitations. Hence low level languages are preferred to build autopilots which help in aggressively optimizing implementations. ArduPilot, Aeroquad, OpenPilot, Navio, Paparazzi UAV are some of the open source autopilots available [8].

4.2 ArduPilot

ArduPilot is a widely employed, open source autopilot implementation which exemplifies state of the art progress in autopilot algorithm development, platforms and testing. It is capable of controlling different types of vehicles ranging from multicopters, traditional helicopters, multirotors, fixed wing aircrafts and rovers [4].

Figure 4.1 shows the execution of ArduPilot's control loop. Setup is the initial phase where all the variables are initialized. The fast loop executes the PID control algorithm. All the tasks in this group are called with highest priority and have guaranteed execution in each cycle. The scheduler contains the list of noncritical tasks and operates in a best-effort manner by distributing the remaining time left after the execution of fast loop among the noncritical tasks. Key features of ArduPilot are enumerated in Table 4.1.

ArduPilot runs on various kinds of embedded hardware platforms typically con-

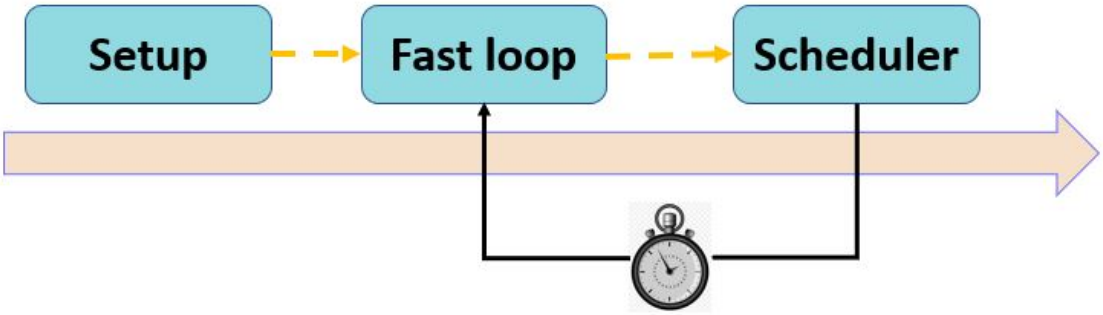


Figure 4.1: ArduPilot's Loop

Table 4.1: Key features of ArduPilot

Sensor communication via SPI, I^2C, CAN Bus, Serial communication, SMBus.
Fully autonomous, semi-autonomous and fully manual flight modes, programmable missions with 3D waypoints, optional geofencing.
Support for vertical take off and landing (VTOL) options.
Real time telemetry and control between ground station device and flight controller hardware.
Rich documentation through ArduPilot wiki.
ArduPilot devices can be contolled either using an RC controller or from a GCS software running on a laptop, tablet or a smartphone.

sisting of a microcontroller connected to peripheral sensors used for navigation. Currently ArduPilot supports the following autopilot boards: Pixhawk, PX4 FMU, VRBrain, Fly-Maple, BeagleBone Black, APM1 and APM2. The main flight code for ArduPilot is written in C++. Support tools, such as APIs or Ground Station software, are written in a variety of languages, most commonly in Python. ArduPilot is released as free software under the GNU General Public License version 3 (or later) and is supported by the company 3DR.

A primary example of ArduPilot is the Pixhawk board, which features a 168 MHz Cortex-M4 MCU and a full sensor array for navigation. On Pixhawk boards, Ardupilot's control loop is statically configured to run at 400 Hz. It also includes a 32-bit STM32F103 failsafe co-processor. Pixhawk runs on the NuttX real time operating system. Some of the limitations of Pixhawk are described in the next section.

4.3 Limitations of Pixhawk

The core of Pixhawk is a microcontroller and so its performance is limited by a microcontroller's capabilities. Real time interrupts demand the CPU's immediate attention to read the I/O and peripherals before the data is lost. Handling an interrupt requires potentially stalling other latency sensitive tasks, incurs context switching overhead, and introduces a wide range of challenges such as managing latency when multiple interrupts occur concurrently, all of which reduce predictability and processor responsiveness. To be able to handle the high data rates and frequencies of real-time I/O and peripherals, microcontrollers must process data more efficiently. This efficiency, however, cannot come from increased clock frequency (which increases power consumption) but through internal changes in microcontroller architectures.

Since Pixhawk is microcontroller-based, it is prone to vulnerabilities such as microprobing, software attacks, eavesdropping and fault generation [31]. Microprobing techniques allow access to the chip surface in order to observe, manipulate and interfere with the integrated circuit. Software attacks are typically carried out on communication protocols and cryptographic algorithms (An example being buffer overflow). Eavesdropping is clandestinely intercepting the analog characteristics of the power supply, interface connections and any electromagnetic radiation by the processor during normal operation. Fault injection involves skipping or replacing some instructions executed by a microcontroller.

One of the major drawbacks of Pixhawk is its saturated I/O capabilities which limit the scope for adding new sensors and applications. Another shortcoming is that Pixhawk is a single core processor and the next generation of flight controllers are embracing dual-core processors to enhance multitasking capabilities. UAV applications are becoming increasingly complex and require more and more processing and I/O interfaces. A microcontroller-based

flight controller has a limited potential due to restricted processing power and I/O extension capabilities. An SoC-FPGA-based flight controller provides an advantage by allowing integrated sensor fusion, real time AI and deep learning [6].

4.4 SoC-FPGA-based Flight Controller

Microcontrollers and FPGAs (Field Programmable Gate Arrays) are the prevalent cores of most embedded systems. Integrating the high-level management capability of processors and the efficient parallelism, extreme data processing and interface functions of an FPGA into a single device forms an even more powerful embedded computing platform. System-on-Chip - FPGA (SoC-FPGA) is a combination of FPGA and microcontrollers in one chip. This new class of programmable devices exploits the flexibility of the FPGA architecture and combines it with the management functionality of a processor. It combines an ARM-based processor system consisting of a processor, peripherals and memory interfaces with an FPGA fabric using a high speed interconnect backbone. SoC-FPGAs are therefore ideal for:

- Reducing power consumption, costs and board size by the integration of processors and digital signal processing (DSP) functions in a FPGA.

- Enhancing the system performance by high bandwidth interconnection between the processor and FPGA.

In this project, we have used OcPoC (shown in Figure 4.2), which is the first of its kind, SoC-FPGA-based open source flight controller.

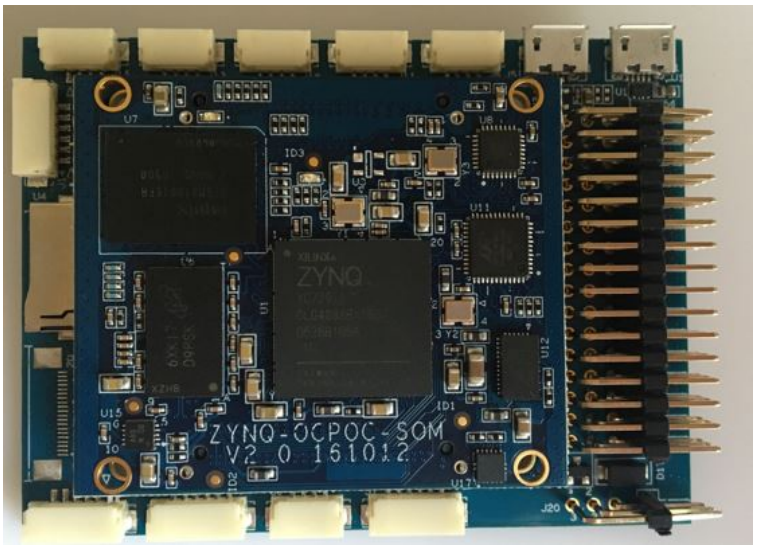


Figure 4.2: OcPoC Board

4.4.1 OcPoC

OcPoC is a flight controller hardware capable of running ArduPilot applications. Apart from the typical sensor options for common peripherals, OcPoC is equipped with efficient input and output capabilities to include fully programmable PWM, PPM and General Purpose Input Output (GPIO) pins to communicate with a wide variety of complex sensors. It has a standard SD card connector and is compact, powerful and fully programmable.

OcPoC has a Xilinx Zynq Z-7010 FPGA-SoC chip at its core. This chip incorporates a 667MHz dual-core ARM Cortex-A9 processor along with Artix-7 FPGA fabric consisting of 28k logic cells and also has a 512Mb DDR3 RAM.

4.4.2 CPU Performance Comparison between Pixhawk and OcPoC

Table 4.2 shows a performance comparison between the CPUs used in Pixhawk and OcPoC respectively. The performance metrics indicate that OcPoC is a superior choice over Pixhawk in terms of CPU capabilities.

Table 4.2: CPU Performance Comparison

Parameter	**Pixhawk(Cortex-M4 CPU)**	**OcPoC (Cortex-A9 CPU)**
Multi Core Processing	No	Yes
Instructions per Second	1.25 DMIPS/MHz per CPU	2.5 DMIPS/MHz
Maximum Clock Frequency	180 MHz	800MHz
Computational Units	Single precision floating point unit	Single and double precision vector floating point unit
Single Instruction Multiple Data (SIMD) feature	Not available	Advanced SIMD extension known as NEON present
Applications	Designed specifically to target microcontrollers	Designed for devices using Linux or Android

Chapter 5

Preliminary Work Done on OcPoC

OcPoC is a custom made SoC-FPGA-based board developed for running ArduPilot applications. This is a pilot project and everything has been developed from scratch. A prototype of the proposed idea was developed on a board provided by a start up company called Aerotenna [2]. A brief overview of the project workflow is shown in Figure 5.1 followed by a more detailed explanation. Conceptualize phase involved envisioning the idea of Trusted Isolation of Privilege and Resources (TIPR) on a UAV platform. Identify phase involved selection of hardware and software components to demonstrate TIPR. Develop phase included the work done to bring up OcPoC to a functional level and the Validate phase involved conducting a successful flight test.

5.1 Conceptualize

In today's era, rapid proliferation of technological growth has given rise to complex embedded systems. Most embedded systems including avionics do not have an internal malware safeguard such as firewalls or an end user to detect abnormalities. The goal is to showcase TIPR on a UAV platform, as represented in Figure 5.2. As explained in earlier chapters, communication hijacking and sensor spoofing are some common forms of attacks in avionics. An SoC-FPGA is used to address these concerns in a novel way. The ability to dynamically alter the system hardware to meet the security and trust needs through

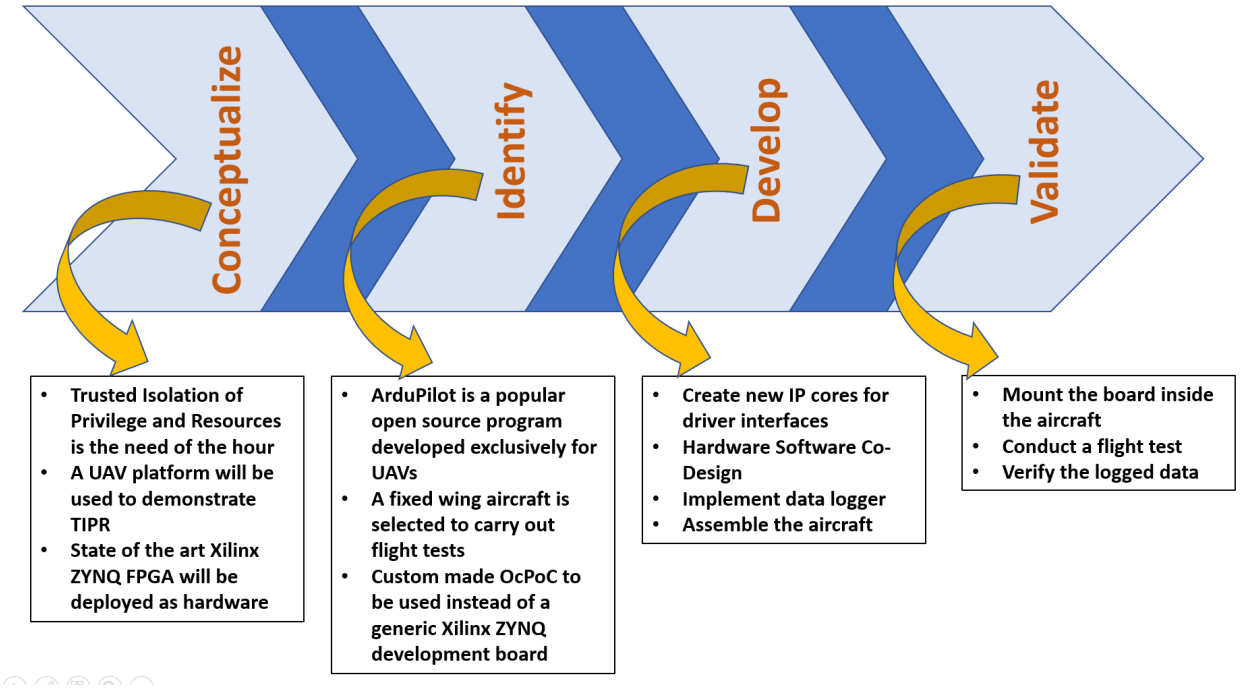


Figure 5.1: Project Workflow

reconfiguration of the FPGA coupled with a powerful processor on the same chip to run an embedded software application makes an SoC-FPGA a candidate to implement a robust and secure flight controller.

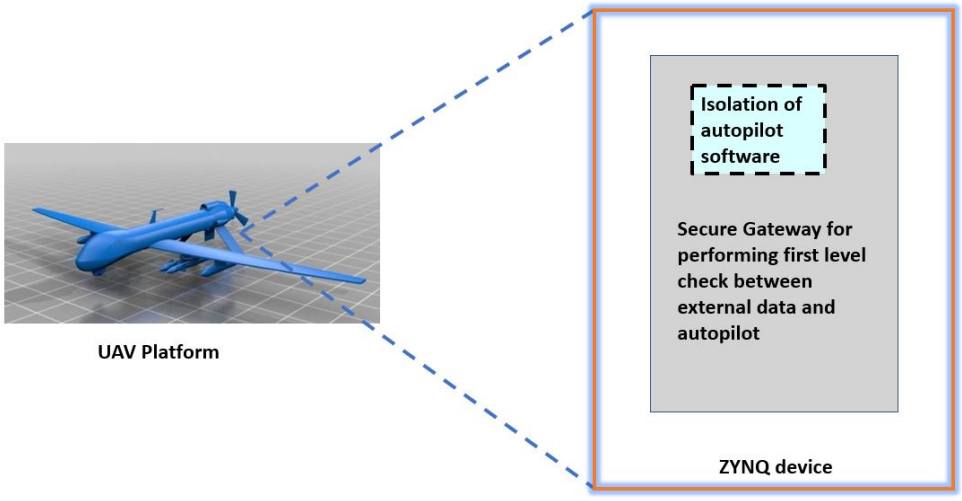


Figure 5.2: TIPR on a UAV platform using a Zynq device

The integrated design flow and the readily available suite of hardware and software development tools for Zynq make it an ideal choice for an SoC-FPGA based platform to develop a flight controller. The Zynq SoC combines a dual-core ARM Cortex-A9 processor along with Xilinx 7 series FPGA fabric. Additionally, multiprocessor capabilities and high I/O access speeds can be leveraged to break many barriers at the application level.

5.2 Identify

ArduPilot is an open source autopilot software program developed exclusively for UAVs. Initial attempts were made to custom build the program and try to port it to a readily available Zynq development board. It was concluded that this effort itself would require lot of time and a generic development board might not be the right choice of hardware. With the introduction of a custom Zynq board called OcPoC, by Aerotenna, a new hardware platform was available, on which the ArduPilot application, whose overview is shown in Figure 5.3, could be integrated seamlessly.

This research is being carried out in collaboration with Virginia Tech's AOE Department. The goal is to eventually implement their control algorithms on top of the ArduPilot software stack. The focus of their algorithms is to improve the disturbance attenuation capability of a UAV in the presence of external noise in the form of unpredictable environmental conditions. A fixed wing aircraft is preferred over a rotary wing aircraft to test their algorithms since testing can be done in the presence of atmospheric turbulence, sensor noise and wind speed. A medium sized aircraft was chosen to carry out the flight tests. Figure 5.4 shows an overview of the identified UAV platform and the chosen hardware and software components.

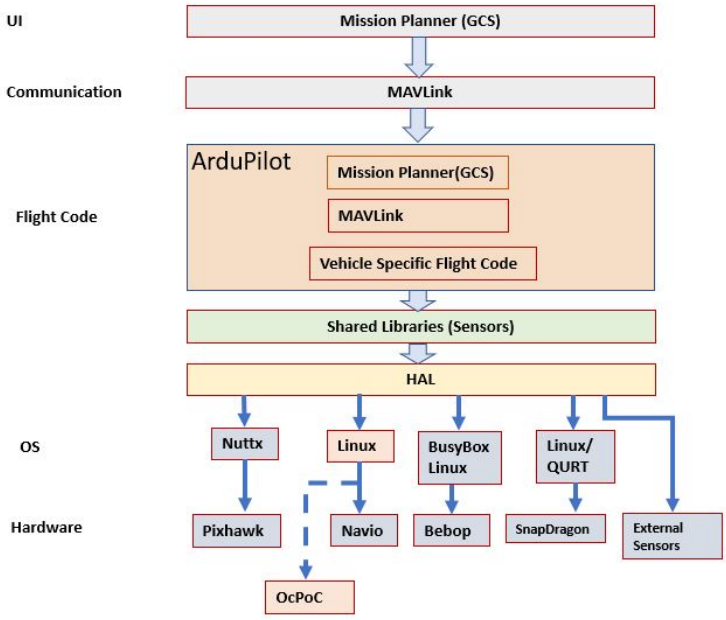


Figure 5.3: ArduPilot software stack

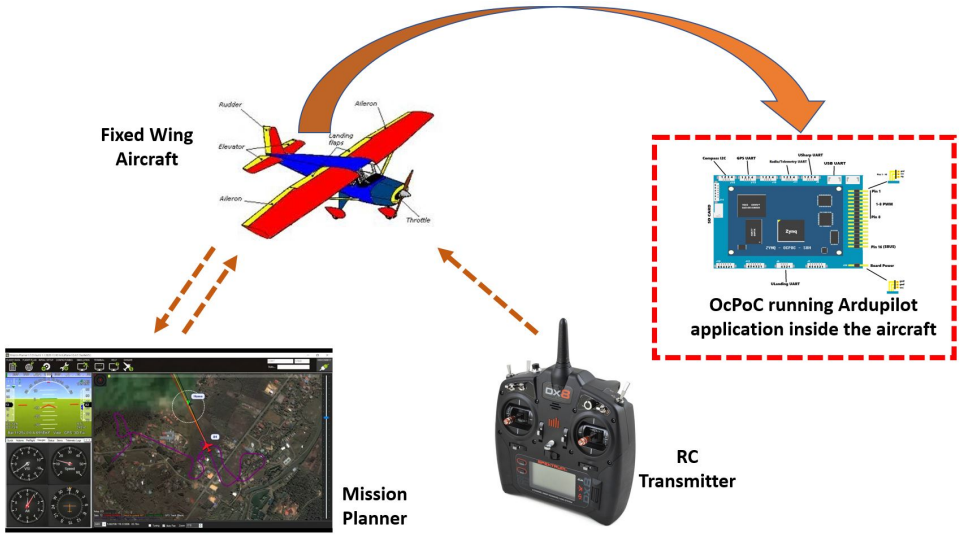


Figure 5.4: UAV platform: A fixed wing aircraft, Autopilot software: ArduPilot, Hardware Platform: OcPoC

5.3 Develop

Every aspect of the design was revisited during the development phase of getting the board ready to fly an aircraft. The skeletal framework was provided by Aerotenna and the basic requirement of the project was to have ArduPilot software application run on the processor with drivers for incoming sensor data and actuators implemented on the FPGA fabric. The ideal choice for such a requirement is a hardware software co-design approach where the overall system is partitioned into hardware and software components, as depicted in Figure 5.5.

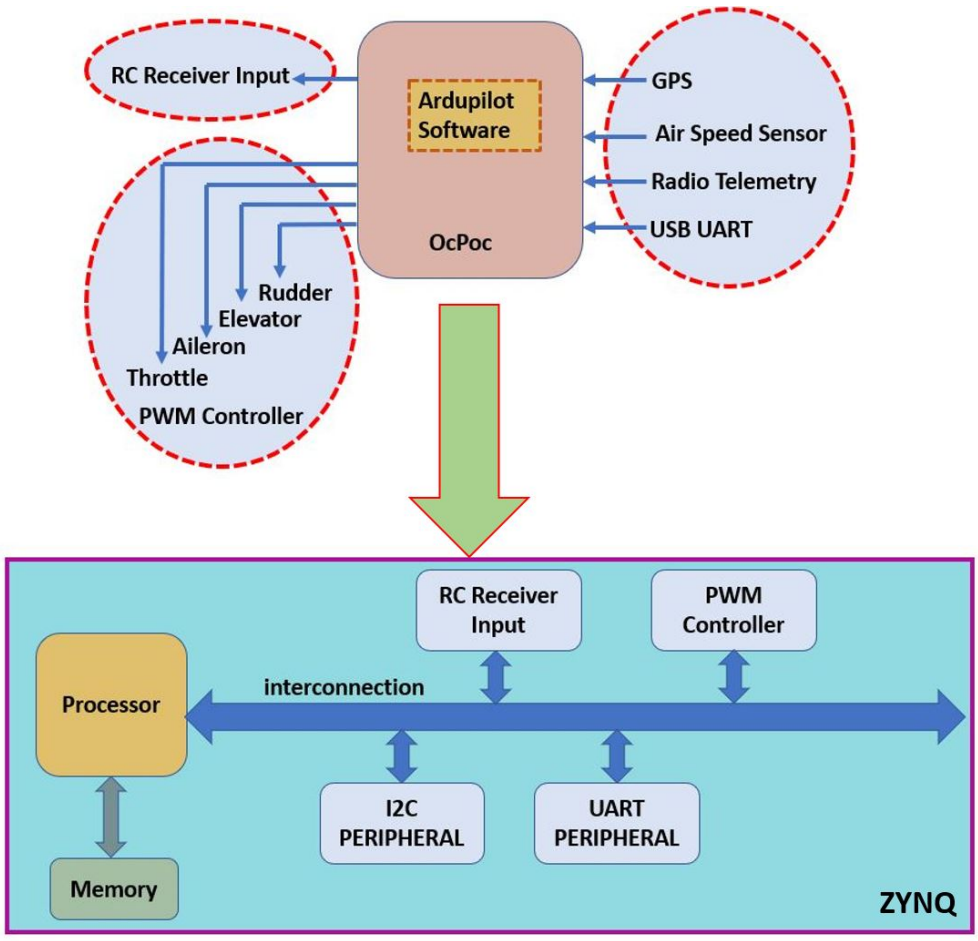


Figure 5.5: Hardware-Software Co-design Approach

The ArduPilot software application runs on embedded Linux on the Zynq's ARM processor. The peripherals are implemented as functional blocks using the reconfigurable FPGA logic resources and are known as soft IP blocks. "Soft-core" refers to the fact that the IP blocks are implemented on a re-programmable device, whereas a "hard-core" processor, like the ARM processor on Zynq is physically implemented in silicon. The dual-core ARM Cortex-A9 processor in Zynq is also addressed as Processing System (PS) and the FPGA fabric is known as Programmable Logic (PL). The interconnects between the ARM processor and the IP cores are implemented via the AXI protocol interface. AXI is the 3rd generation of AMBA interface developed by ARM exclusively for high performance and high clock frequency system design. It is commonly used for management of a large number of controllers and peripherals in SoCs. Figure 5.6 shows which components will be implemented in the PS and PL respectively.

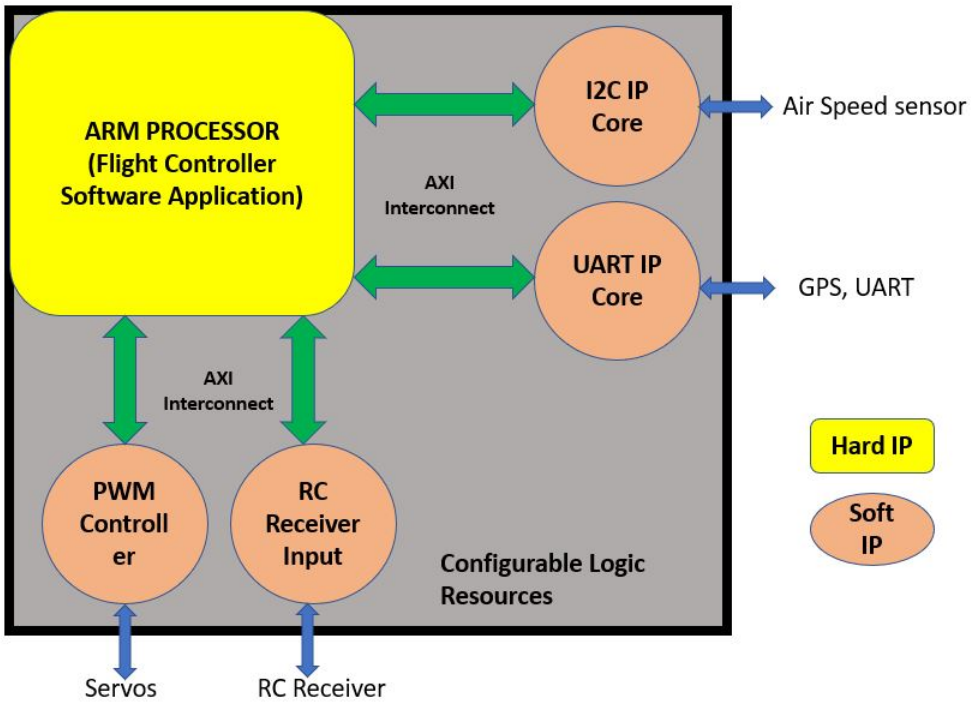


Figure 5.6: Hardware and Software Components inside Zynq

5.3.1 Hardware Development

The hardware system was developed using the Xilinx Vivado IDE suite. The IP Integrator component of Vivado provides a rich set of pre-verified IP blocks in its library. These IP blocks allow for custom configuration [34]. Legacy or custom HDL code can also be converted into soft IP blocks for design reuse and ease of integration. Once the Vivado block diagram is configured, it is followed by the regular FPGA design flow which consists of logic synthesis, pin assignments, place and route, timing analysis and bitstream generation. The following sections describe the specifications of the Zynq device used in OcPoC, the concept of memory mapping followed by an overview of the IP cores used in the OcPoC hardware configuration.

5.3.1.1 Target Zynq SoC Device

The Zynq device used in OcPoC is **xc7z010clg400-1** and its specifications are given in Table 5.1.

The hardware design for OcPoC consists of a mix of Vivado IP blocks and custom soft IP blocks. Standard Vivado IPs for I2C and UART drivers are used, where as PWM controller and RC receiver input are custom made IP blocks.

The top level Block Diagram (BD) for OcPoC is shown in Figure 5.7

5.3.1.2 Memory Mapping

In an ARM based system, IPs are accessed by the processor through memory mapped registers. Each IP has a unique identifier known as its base address which can be accessed by the processor to write to or read from. The processor treats each IP block as an individual

Table 5.1: Specifications for Z-7010

PL:	
FPGA Technology	Artix-7
No. of Logic cells	28K
No. of 6-inputLUTs	17600
No. of Flipflops	35200
No. of 36Kb Block RAMs	60
No. of DSP48 slices	80
PS:	
CPU Architecture	Dual-core ARM Cortex-A9
L1 Cache	32Kb Instruction and Data per processor
L2 Cache	512Kb
SDRAM	DDR3
Peripherals	UART, CAN, I2C, SPI, GPIO

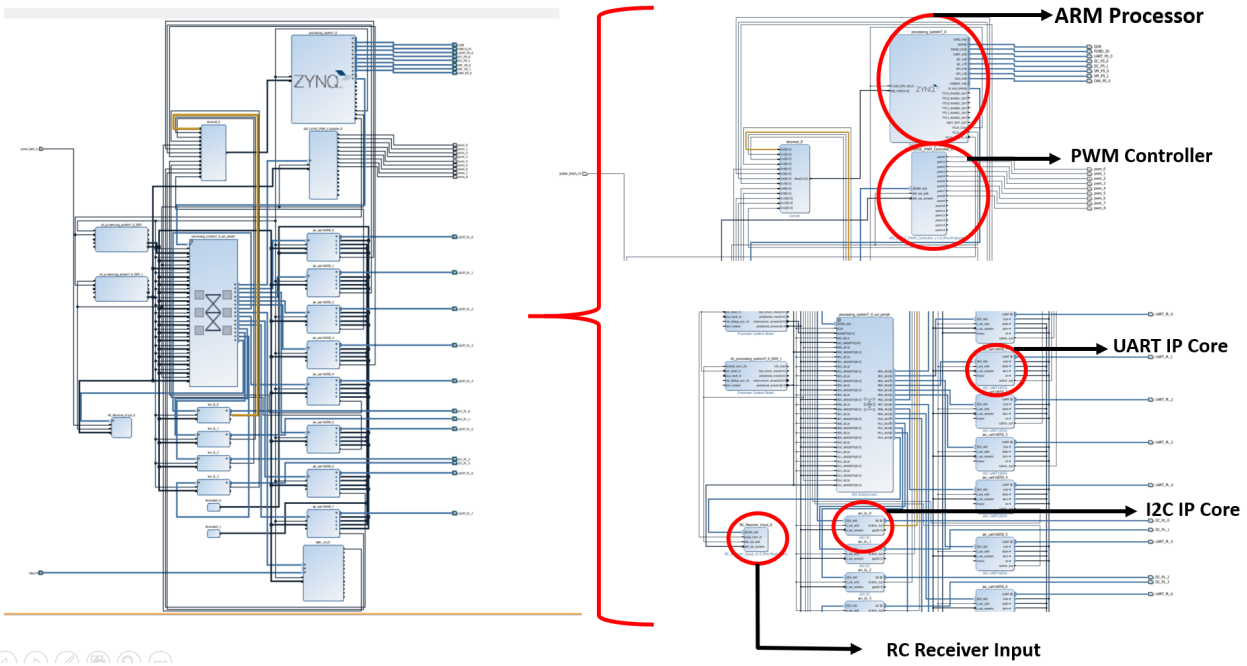


Figure 5.7: Zoomed view of Vivado BD for OcPoC

hardware device using normal memory access instructions. These unique addresses are also used to build the device tree during the software development stage.

The Address Editor feature of Vivado automatically assigns base addresses to each of the IPs. They can be assigned manually in the Address Editor. IP Integrator allows for a validation check for critical faults in the BD or address map. The address map for OcPoC is shown in Figure 5.8

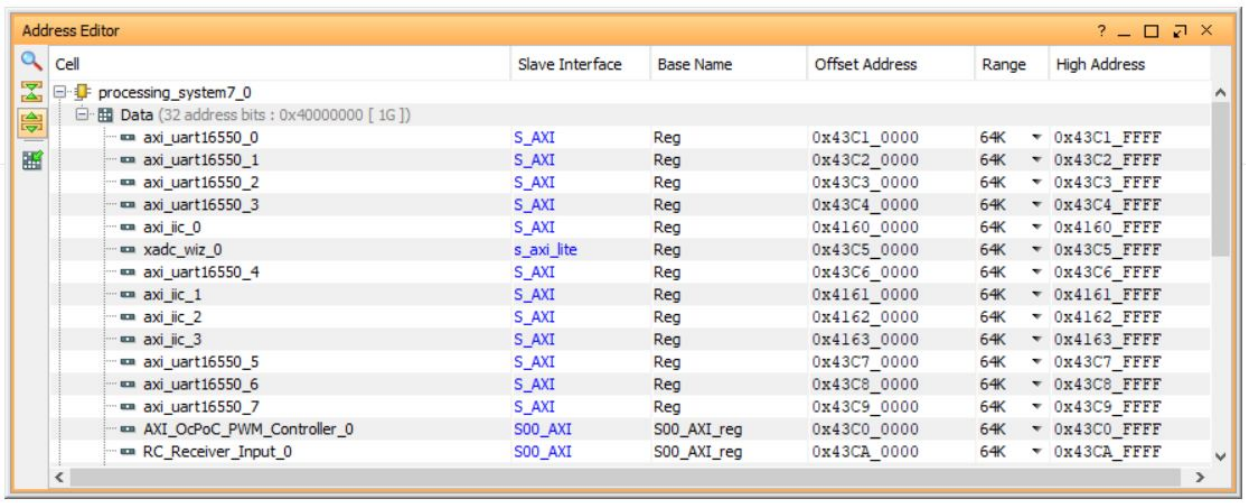


Figure 5.8: Address map for OcPoC

5.3.1.3 A brief description of the IP cores in OcPoC

1. **Processing System:** PS IP, as shown in Figure 5.9, is the most important IP as it is used to configure the ARM processor. Configuration includes defining the available Multiplexed Input Output (MIO) interfaces such as UART, setting up different clock frequencies needed for the system, and the configuration of the AXI ports for communication with other soft IP cores. This IP hosts Linux applications such as ArduPilot.

2. **Peripheral IPs such as UART and I2C:** Shown in Figure 5.10, these IPs provide

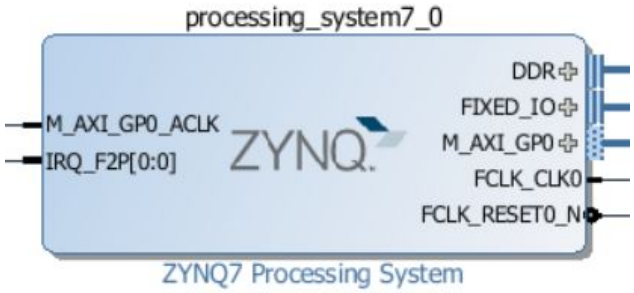


Figure 5.9: Zynq PS hard IP

low speed serial interfaces to any external peripherals and generate interrupts when required. In OcPoC they are used to set up data transfer between the PS and sensor or GPS modules.

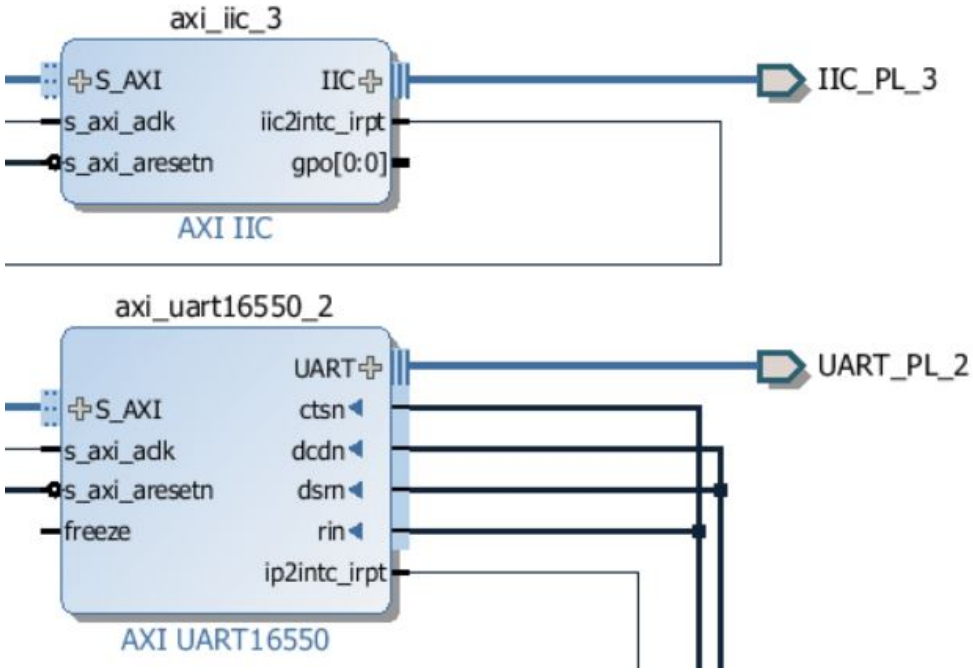


Figure 5.10: Peripheral IPs in OcPoC

3. **Processor System Reset:** Shown in Figure 5.11, this IP is automatically generated by the IP Integrator and is used to synchronously reset all the IP cores in the PL. The reset can be configured to be either active low or active high.

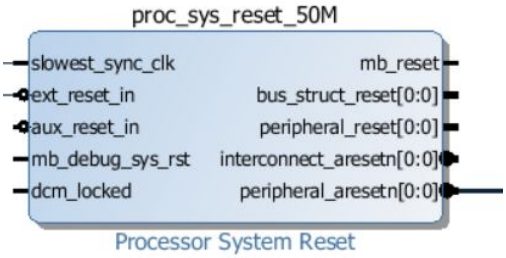


Figure 5.11: System reset IP

4. **AXI Interconnect IP:** Shown in Figure 5.12, this IP is made up of a cluster of smaller IP cores intended to bridge together one or more master AXI interfaces with one or more AXI slave interfaces. The clock and reset for all IP cores are also routed through this IP. AXI Interconnect greatly simplifies the task of interfacing all IP cores with the PS by automatically generating the connections.

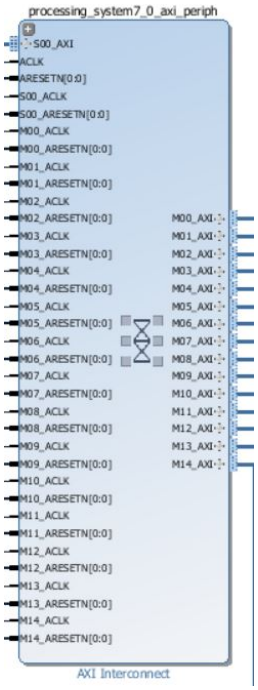


Figure 5.12: AXI Interconnect IP

5. **Concat:** Shown in Figure 5.13, this IP is used to concatenate bus signals of varying

width. In OcPoC, it is used to concatenate all interrupts and route them to the PS interrupt controller.

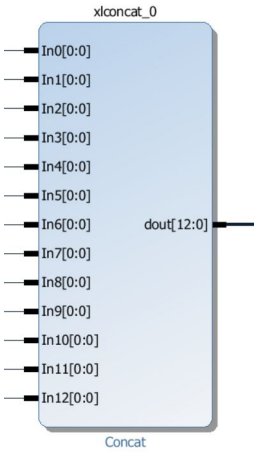


Figure 5.13: AXI Concat IP

6. **AXI OcPoC PWM Controller:** Shown in Figure 5.14, this custom made IP is used to send PWM signals to each of the servos that make up the aileron, throttle, rudder and elevators in the fixed wing aircraft. The number of channels depends on the number of GPIO pins available on the board to support PWM signals and on the vehicle type and configuration parameters. OcPoC supports upto 12 PWM signals out of which eight are configured to support eight servo channels.

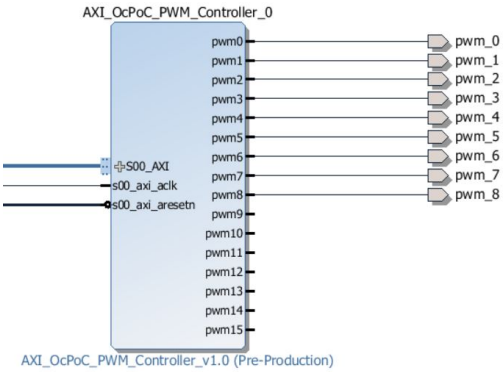


Figure 5.14: PWM Controller IP

7. **RC Receiver input:** Shown in Figure 5.15, this IP receives the RC commands from the RC transmitter. RC input is one of the most important components of an autopilot and is used to control the air frame, change modes and configure the auxiliary channels. RC input is received via SBUS which is a 100 kilobaud inverted UART serial communication protocol developed by Futaba.

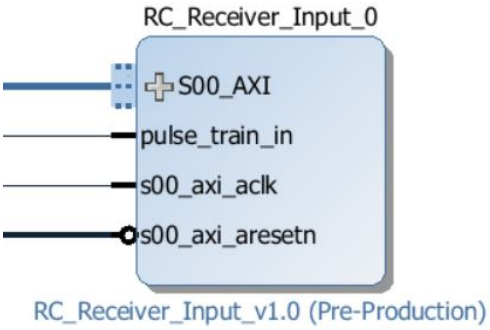


Figure 5.15: RC Receiver Input IP

5.3.2 Software Development

The flexibility of the Zynq platform allows for the hardware configuration on which the software operates to vary. The project exported from Vivado represents the tailor made hardware upon which the software is developed. The software setup can be considered to be a stack of layers, as illustrated in Figure 5.16, built upon the underlying hardware[13].

Hardware Base System contains information about the processor type, cache configuration, memory size and type, common peripherals and interrupt sources in a format understandable by the OS. Board Support Package (BSP) is a collection of libraries and drivers upon which simple software applications can be developed even without a supporting OS.

There are 2 ways in which software development can progress:

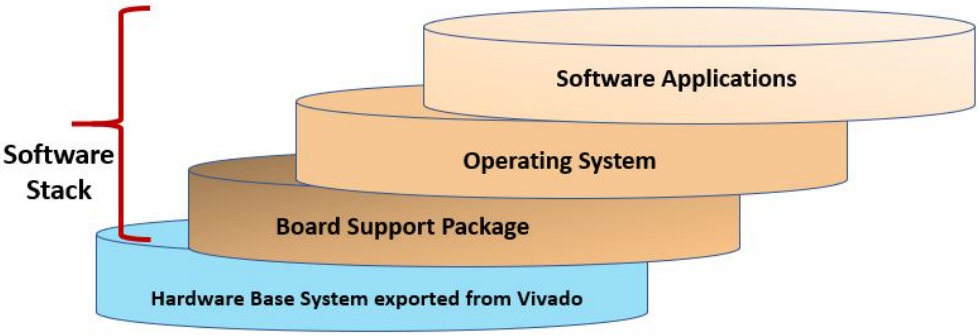


Figure 5.16: Hardware and software layers in a Zynq SoC

- **Baremetal applications:** The BSP allows the application to directly interact with the hardware through the drivers provided by BSP.
- **OS applications:** These applications run on an embedded OS such as Linux or an RTOS. More complex applications which involve scheduling can be run. The kernel which is at the intermediary level arbitrates the interactions between the software application above and the hardware below.

5.3.3 Running Linux on ARM

Linux offers the flexibility of a multipurpose computer providing great support for multitasking, file systems, networking etc. Linux has a vast ecosystem of open source tools and languages making it a developer friendly embedded OS option. Moreover, porting of ArduPilot application on Linux is easier as compared to porting it to an RTOS.

Linux can be made to boot either from a QSPI flash or SD card. In OcPoC, we boot Linux from an SD card. There are different aspects involved to successfully boot Linux on Zynq and run a software application. PS configuration data and custom hardware information must be clustered into a single entity called the BOOT.bin file. Device tree is a data structure describing the physical devices in the system. As shown in Figure 5.17, there

is a one-to-one match to the memory mapped addresses configured earlier in Vivado. Device tree should be converted into a .dtb file. BOOTbin, devicetree.dtb along with Linux kernel image and the root file system provided by Aerotenna on their github page, must be loaded on the SD card. Figure 5.18 illustrates the files needed to successfully boot Linux on a Zynq device.

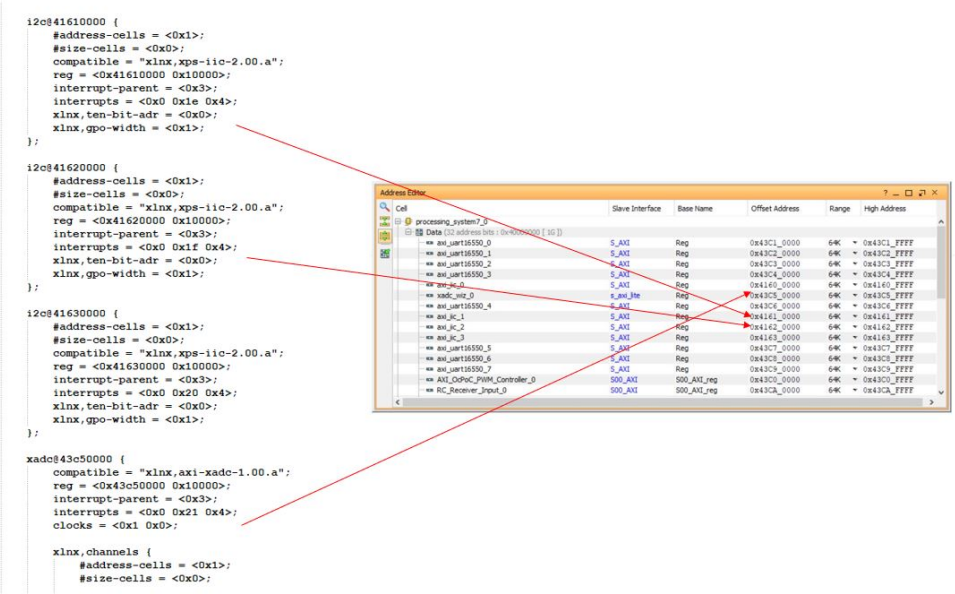


Figure 5.17: Device tree compiled using addresses assigned in Vivado

5.3.4 ArduPlane Application

The basic structure of ArduPilot code consists of 5 parts [3]:

- **vehicle code** : We use the Plane vehicle code i.e. ArduPlane application since our vehicle is a fixed wing aircraft. The ArduPlane code overview is shown in Figure 5.19.

- **shared libraries** : This code base is common for different vehicle types i.e. Copter,

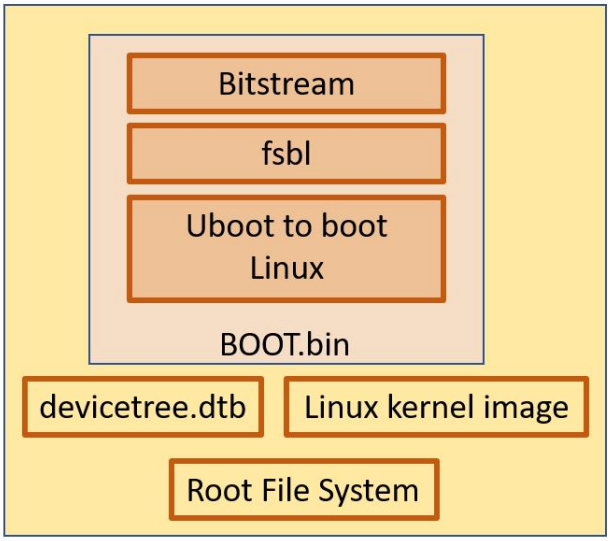


Figure 5.18: Files required for booting Linux

Rover, Plane and Antenna Tracker. These libraries include sensor drivers, attitude and position estimators, PID controllers etc.

- **Hardware Abstraction Layer (HAL)** : This section of the code makes it possible to port ArduPilot application across different hardware platforms. To customize an ArduPilot application for Zynq, a new section is added, as illustrated in Figure 5.20.

- **tools directories** : These form miscellaneous support directories.

- **external support code** : Used to support additional features such as access to MAVLink protocol.

After making the necessary changes to the ArduPlane code to make it compatible with Zynq, we need to compile the code using Xilinx-Linux cross compiler tools and generate an executable which will also be placed in the SD card along with other files.

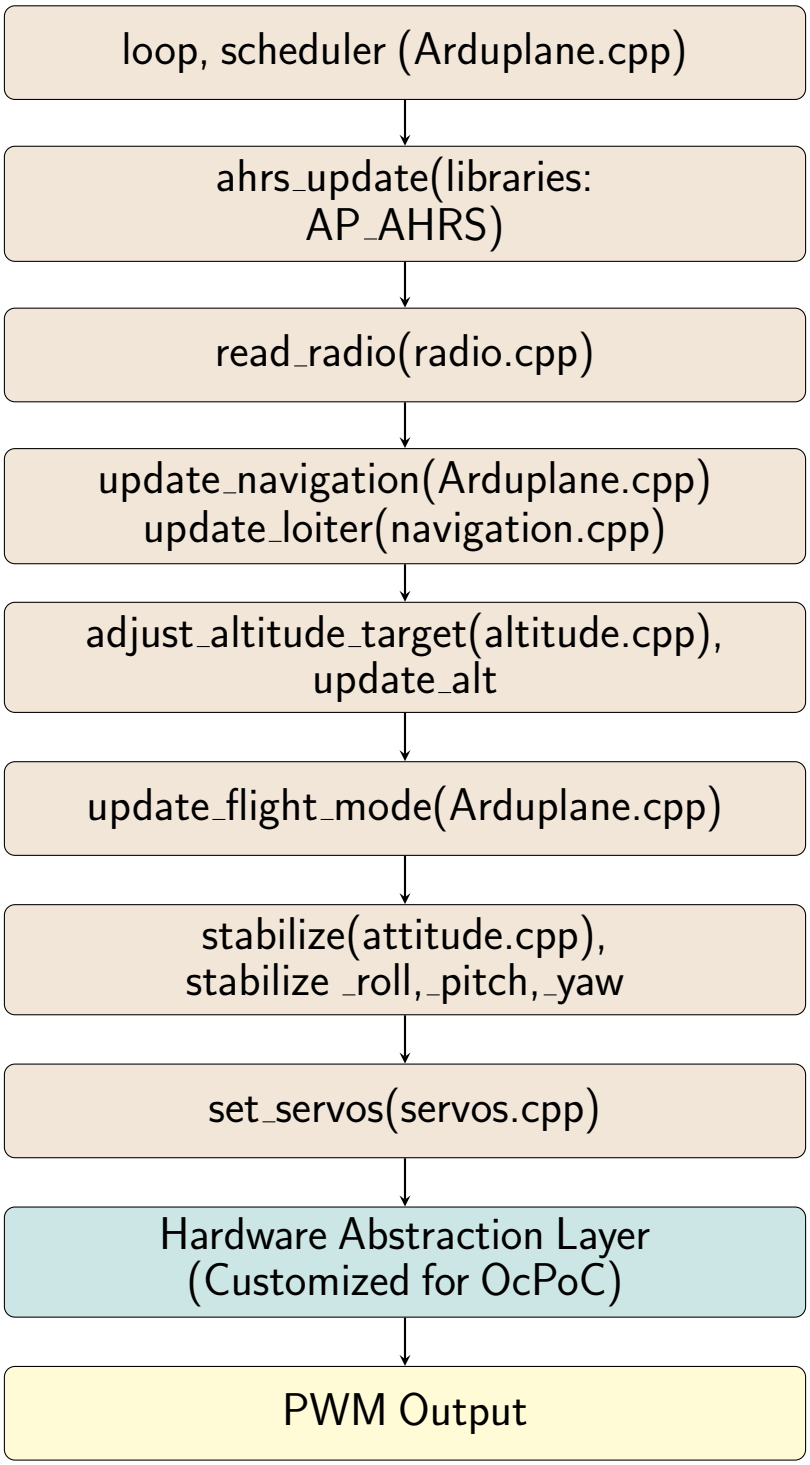


Figure 5.19: ArduPlane software architecture overview [3]

```
#ifndef __AP_HAL_BOARDS_H__
#define __AP_HAL_BOARDS_H__
/**
 * C preprocesor enumeration of the boards supported by the AP_HAL.
 * This list exists so HAL_BOARD == HAL_BOARD_xxx preprocessor blocks
 * can be used to exclude HAL boards from the build when appropriate.
 * Its not an elegant solution but we cant get too fancy if we want to
 * work with the Arduino mk and IDE builds without too much
    modification.
 */

#define HAL_BOARD_APM1     1
#define HAL_BOARD_APM2     2
#define HAL_BOARD_AVR_SITL 3
#define HAL_BOARD_SMACCM   4 // unused
#define HAL_BOARD_PX4      5
#define HAL_BOARD_FLYMAPLE 6
#define HAL_BOARD_LINUX    7
#define HAL_BOARD_VRBRAIN  8
#define HAL_BOARD_EMPTY    99

// default board subtype is -1
#define HAL_BOARD_SUBTYPE_NONE -1

/**
   HAL Linux sub-types, starting at 1000
 */
#define HAL_BOARD_SUBTYPE_LINUX_NONE     1000
#define HAL_BOARD_SUBTYPE_LINUX_ERLE     1001
#define HAL_BOARD_SUBTYPE_LINUX_PXF      1002
#define HAL_BOARD_SUBTYPE_LINUX_NAVIO    1003
#define HAL_BOARD_SUBTYPE_LINUX_ZYNQ     1004 // Zynq platform defined
    for OcPoC
#define HAL_BOARD_SUBTYPE_LINUX_BBBMINI  1005

/**
   HAL PX4 sub-types, starting at 2000
 */
#define HAL_BOARD_SUBTYPE_PX4_V1         2000
#define HAL_BOARD_SUBTYPE_PX4_V2         2001
```

Figure 5.20: **AP_HAL_Boards.h** modified to accommodate a Zynq board (OcPoC)

5.3.5 Feature Addition - Auxiliary Channel Controlled Data Logger

Requirement: Capture the real time sensor data calculated by the ArduPlane application while flying.

Challenge: Retain the integrity of the logged data on the SD card after powering off the board.

Approach: The scheduler algorithm in an ArduPlane application schedules a specific set of tasks to run in the allotted time depending on their priority and remaining time left before one complete loop concludes. Higher the task is in the list, higher is the execution priority.

First, a function called `write_to_file`, was created to capture sensor data during every iteration of the loop and inserted into the scheduler's list of tasks as shown in in Figure 5.24. This was followed by checking if the scheduler had enough time to capture all the required parameters and write them to a file during every iteration (shown in Figure 5.25). The execution of this task was verified by running the ArduPlane application on the OcPoC board. A snapshot of the log file is shown in Figure 5.21. Table 5.2 enumerates the sensor parameters which are logged.

Table 5.2: Logged data parameters

GPS Latitude, Longitude, Altitude
Airspeed
Attitude and Heading Reference System (AHRS) Roll, Pitch, Yaw
RC channel throttle, roll, pitch, rudder
IMU Roll, Pitch, Yaw
Body-Axis Roll rate, pitch rate, yaw rate

The ArduPlane application typically starts running once the board is powered on,

```
Run 4
Retrieved checksum: 987123
Retrieved GPS_lat: 0
Retrieved GPS_lon: 0
Retrieved GPS_alt: 0
Gps_status: 0
retrieved pitot_airspeed: 0.000000
Retrieved_ap_mode_r: 2
retrieved pxk_phi_r: -3.137104
retrieved pxk_theta_r: -0.043993
retrieved pxk_psi: 0.434999
retrieved pxk_p: 0.000447
retrieved pxk_q: 0.000361
retrieved pxk_r: -0.000542

Run 5
Retrieved checksum: 987123
Retrieved GPS_lat: 0
Retrieved GPS_lon: 0
Retrieved GPS_alt: 0
Gps_status: 0
retrieved pitot_airspeed: 0.000000
Retrieved_ap_mode_r: 2
retrieved pxk_phi_r: -3.137092
retrieved pxk_theta_r: -0.043994
retrieved pxk_psi: 0.434969
retrieved pxk_p: 0.000553
retrieved pxk_q: 0.000094
retrieved pxk_r: 0.000974

Run 6
Retrieved checksum: 987123
Retrieved GPS_lat: 0
Retrieved GPS_lon: 0
```

Figure 5.21: Snapshot of the logged data inside the lab environment

and it continues to run until the board is powered off. Initially the data was logged continuously for the duration ArduPlane was running. Powering off of the board was treated as an abruptly terminated process resulting in logged data being corrupted or erased. Typical Linux kernel commands such as SIGINT or SIGKILL invoked to kill a running application were not effective in saving all the data and therefore required a different solution.

On further analysis, we were able to infer that data logging needs to be performed only when the aircraft is in the air and not during take off or landing. A novel way of controlling the start and stop of data logging through an auxiliary channel on the RC transmitter was proposed and implemented. `write_to_file` function is called only if there is an active "ON" signal from the auxiliary channel of the RC transmitter. The human operator controlling the RC transmitter can decide when to start and stop data logging. The logs also included timestamps to verify the log data depending on the flight envelope and the number

of runs.

5.3.6 Assembling the Aircraft

Virginia Tech's AOE Department has developed expertise in building fixed wing vehicles and quadcopters for their various research projects. The task of assembling the aircraft, as shown in Figure 5.22 and mounting the enclosure containing the OcPoC board was undertaken by Devaprakash Muniraj, a PhD student in the AOE Department, who also worked on control algorithms eventually to be implemented on OcPoC.

Figure 5.22: Fixed wing aircraft assembled for running flight tests with OcPoC as the flight controller hardware

5.4 Validate

Flight tests were conducted with the newly implemented data logging feature. The data captured in the log file verified that the basic ArduPlane application ran as expected

and the sensor data was captured correctly. Verifying the correct operation of all interfaces and conducting the flight tests successfully paved the way for further development on the OcPoC.

Figure 5.23 summarizes the tasks carried out on OcPoC board to get it up and running. Each task was an incremental addition to build a successful SoC-FPGA based flight controller.

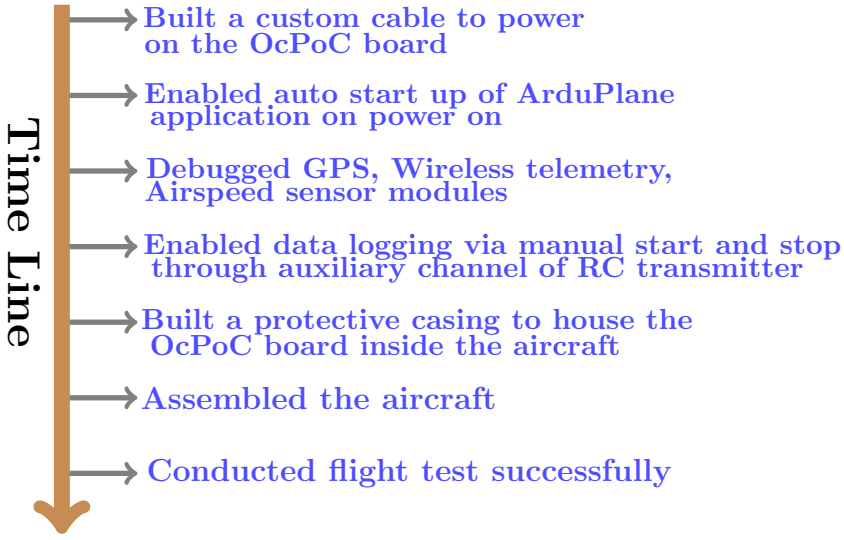


Figure 5.23: Milestones achieved during different stages

```
#include "Plane.h"
#include <stdio.h>
#include <string.h>

#define SCHED_TASK(func, rate_hz, max_time_micros) SCHED_TASK_CLASS(
    Plane, &plane, func, rate_hz, max_time_micros)

/*
  scheduler table - all regular tasks are listed here, along with how
  often they should be called (in Hz) and the maximum time
  they are expected to take (in microseconds)
 */
const AP_Scheduler::Task Plane::scheduler_tasks[] = {
                            // Units:   Hz      us
    SCHED_TASK(ahrs_update,           400,    400),
    SCHED_TASK(read_radio,             50,    100),
    SCHED_TASK(check_short_failsafe,   50,    100),
    SCHED_TASK(update_speed_height,    50,    200),
    SCHED_TASK(update_flight_mode,    400,    100),
    SCHED_TASK(compass_save,          0.1,    200),
    SCHED_TASK(Log_Write_Fast,         25,    300),
    SCHED_TASK(write_to_file,          50,    500), // write_to_file
        function inserted in scheduler tasks
    SCHED_TASK(update_logging1,        10,    300),
    SCHED_TASK(update_logging2,        10,    300),
    SCHED_TASK(parachute_check,        10,    200),
    SCHED_TASK(terrain_update,         10,    200),
    SCHED_TASK(update_is_flying_5Hz,    5,    100),
    SCHED_TASK(dataflash_periodic,     50,    400),
    SCHED_TASK(avoidance_adsb_update,  10,    100),
    SCHED_TASK(button_update,           5,    100),
    SCHED_TASK(stats_update,            1,    100),
};

/*
  update AP_Stats
 */
void Plane::stats_update(void)
{
    g2.stats.update();
}

void Plane::setup()
{
    cliSerial = hal.console;

    // load the default values of variables listed in var_info[]
    AP_Param::setup_sketch_defaults();
```

Figure 5.24: Inserting the task in the scheduler list

```
MS4525: no sensor found
MPU: temp reset 5664 0
Init
Run: Time available:20000
Task:write_to_file, dt:1, Tick Cnt:1, last run:0, interval_ticks:1,
    _loop_rate_hz50, Time allowed:100
Task:write_to_file took:150 microsec, Time Available:19559
Run: Time available:20000
Task:write_to_file, dt:1, Tick Cnt:2, last run:1, interval_ticks:1,
    _loop_rate_hz50, Time allowed:300
Task:write_to_file took:106 microsec, Time Available:19841
Run: Time available:20000
Task:write_to_file, dt:1, Tick Cnt:3, last run:2, interval_ticks:1,
    _loop_rate_hz50, Time allowed:100
Task:write_to_file took:107 microsec, Time Available:19907
Run: Time available:20000
Task:write_to_file, dt:1, Tick Cnt:4, last run:3, interval_ticks:1,
    _loop_rate_hz50, Time allowed:300
Task:write_to_file took:107 microsec, Time Available:19913
Run: Time available:20000
Task:write_to_file, dt:1, Tick Cnt:5, last run:4, interval_ticks:1,
    _loop_rate_hz50, Time allowed:100
Task:write_to_file took:107 microsec, Time Available:19878
Run: Time available:20000
.
.
.
.
Task:write_to_file took:107 microsec, Time Available:19891
Run: Time available:20000
Task:write_to_file, dt:1, Tick Cnt:455, last run:454, interval_ticks:1,
     _loop_rate_hz50, Time allowed:100
Task:write_to_file took:107 microsec, Time Available:19861
Run: Time available:20000
Task:write_to_file, dt:1, Tick Cnt:456, last run:455, interval_ticks:1,
     _loop_rate_hz50, Time allowed:300
Task:write_to_file took:221 microsec, Time Available:19888
Run: Time available:20000
Task:write_to_file, dt:1, Tick Cnt:457, last run:456, interval_ticks:1,
     _loop_rate_hz50, Time allowed:100
Task:write_to_file took:108 microsec, Time Available:19852
Run: Time available:20000
Task:write_to_file, dt:1, Tick Cnt:458, last run:457, interval_ticks:1,
     _loop_rate_hz50, Time allowed:300
Task:write_to_file took:150 microsec, Time Available:19861
```

Figure 5.25: Verification of write_to_file being executed for every run

Chapter 6

Implementation of I/O Processor

6.1 Introduction

SoC-FPGAs today consist of millions of logic gates, megabytes of memory and high speed transceivers coupled with powerful processors. For data and mission sensitive applications running on UAS, improving the security is a continuous goal. As the complexity and scope of applications running on FPGAs has increased, the need to protect the data and applications has gained paramount importance. The security objectives revolve around the following four parameters [19]:

- Availability : Ensures that a system's services are available whenever expected, in spite of the presence of attacks.
- Confidentiality : The secrecy of data transmitted between two communicating parties is maintained without a third party being able to eavesdrop.
- Authentication : Verifying the true source of data i.e. prevent a malicious party from masquerading as someone else.
- Data Integrity : Protection of data against unauthorized changes including intentional alteration or accidental change.

The above parameters are applied to the OcPoC environment to assess its characteristics (shown in Figure 6.1).

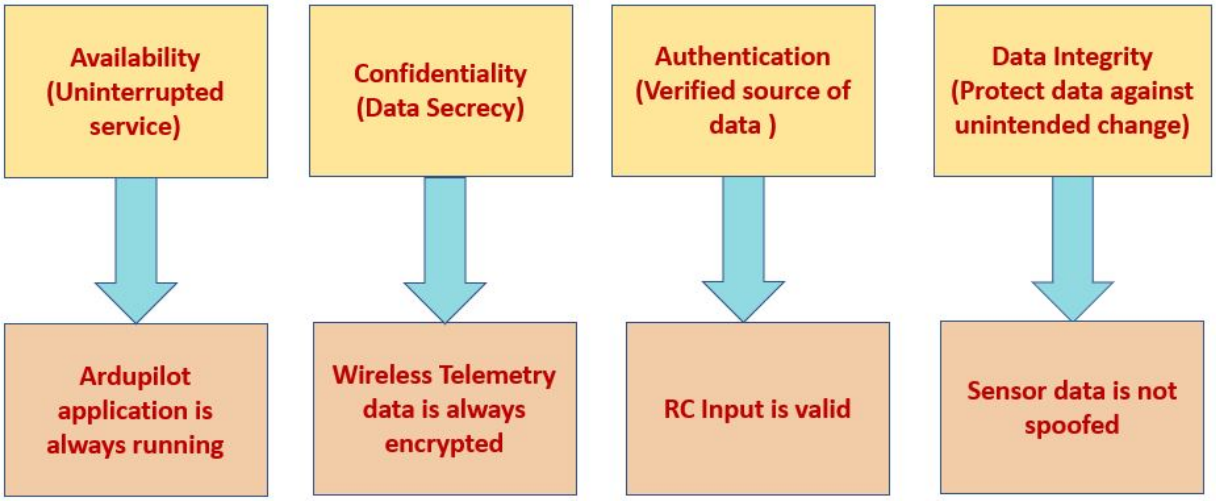


Figure 6.1: Security parameters as applied to OcPoC

A layered security approach, as shown in Figure 6.2, helps ensure that a robust and safe system design is implemented. Our focus was to enhance data security by leveraging the FPGA fabric's isolation from software.

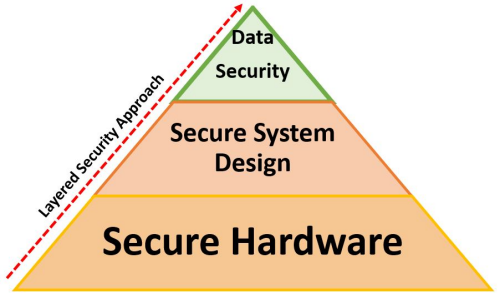


Figure 6.2: Layered Security in SoCs

6.2 Considerations before Implementation

6.2.1 FPGA Bitstream Protection via Encryption

The FPGA fabric is programmed using a bitstream. This binary data defines how internal resources are configured to perform logic operations. Reverse engineering an FPGA bitstream is tedious and requires tremendous effort. Assuming this is somehow done, reverse engineering might reveal the netlist of the application, but transforming a multimillion gate netlist into an understandable design that can be modified is a huge problem and may be infeasible [27]. Zynq devices provide a high degree of design security through 256-bit Advanced Encryption Standard (AES) bitstream encryption and on-chip bitstream decryption with dedicated memory for storing the encryption key [1]. AES decryption logic is not available to the user and is used to decrypt only the configuration bitstream. In addition to this, Zynq devices also employ Hash Message Authentication Code (HMAC) algorithm to authenticate the bitstream [5], thereby ensuring that FPGA design cannot be copied or reverse engineered for use on unintended FPGAs.

6.2.2 Judicious Use of OcPoC Resources

The OcPoC is essentially an embedded system. Any security-related feature must meet computational power and memory constraints. While the ArduPilot application consumes considerable processor resources, a good portion of precious FPGA resources is unused. This can be used to provide enhanced security. The novelty here is that instead of using FPGA logic for conventional hardware acceleration to improve system performance, we are using hardware mostly to isolate trusted functions from malware.

6.2.3 Choice of FPGA over ARM for Added Security

The FPGA fabric allows custom modules to be implemented with private hardware resources. Unlike software processes, FPGA tasks running in parallel do not compete for the same processing resources. The reconfigurability feature of an FPGA offers almost limitless flexibility to configure a soft processor (which is not the case with hard processors such as the ARM). A soft processor running on FPGA fabric can perform a plethora of tasks while consuming limited FPGA resources, not necessarily requiring an FPGA. It is important to note that using an ARM processor for simple tasks can be an overkill.

As UAVs continue to rapidly advance, the dual-core processor in Zynq may be expanded to perform asymmetric multiprocessing. The cores might be needed to carry out more computationally exhaustive tasks leaving no time or resources for adjunct features such as security checks. Also the dual-cores share a common DDR memory, L2 cache and several other resources. Since there is no intertwining of common resources between PS and PL, there is better isolation of privileged tasks and regular application related processing.

6.3 Applications Involving Use of Co-processor

A co-processor is used to augment the role of a primary processor (CPU). Some of the operations performed by the co-processor include floating point calculations, graphics, signal processing, encryption or I/O interfacing with peripheral devices. Secondary co-processors are being increasingly used for addressing a wide variety of computational demands such as image compression [33], encryption/decryption [10], object tracking [20], networking [21], audio video processing [22], floating point operations etc.

One interesting example is Apple's Secure Enclave Processor (SEP). Apple uses the

A7 processor to store finger print data in an encrypted form. The decryption is performed using a key available in the Secure Enclave [15]. The rest of the iOS does not have access to this data. Access to the SEP (which has its own memory) is strictly controlled by the hardware.

6.4 Design Approach

The primary goal was to design a soft-core processor whose main function was to interact with I/O peripherals and communicate with ARM running the ArduPilot software application. The soft-core processor in Zynq devices, configured using FPGA resources as shown in Figure 6.3, is called MicroBlaze.

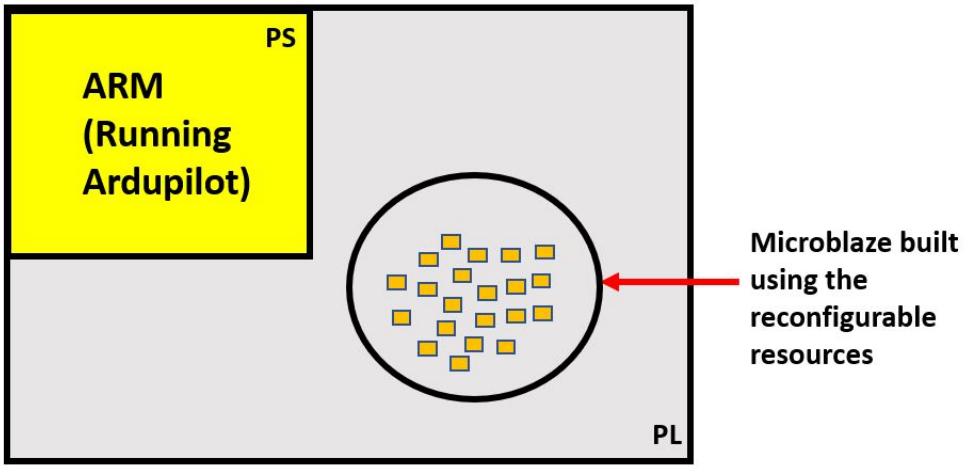


Figure 6.3: MicroBlaze in Zynq

6.4.1 MicroBlaze Overview

The MicroBlaze is a soft IP core used to implement a microprocessor entirely within the Xilinx FPGA's general purpose memory and logic fabric. The MicroBlaze comes with a

RISC instruction set [35] and uses a Local Memory Bus (LMB) to efficiently interact with Block RAM (BRAM) memory instantiated alongside the processing core within the PL. The size of BRAM dedicated to the processor memory is configurable. MicroBlaze also has support for instruction and data caches similar to the Application Processing Unit (APU) within the PS. Key features of MicroBlaze are enumerated in Table 6.1.

Table 6.1: Key features of MicroBlaze

3 or 5 stage pipeline support
Native AXI-4 support
AXI Coherency Extension (ACE) support
Cache line word length: 4, 8 or 16
Area or speed optimized configuration option
Intrusive profiling
Support for memory management unit
Low latency interrupt mode support
Fault tolerance, including Error Correction Codes (ECC) and lockstep support
MPU mode for region protection for secure RTOS applications
Instruction and data caches
Cache size configurable: 2kB - 64kB (Block RAM based)
LMB Instruction and data side interface
Hardware barrel shifter
Hardware multiplier and divider
Upto 16 AXI stream interfaces
Floating Point Unit (single precision, IEEE 754 compatible)
Processor version register
Relocatable base vectors
Support for sleep mode and sleep instruction

6.4.2 Establishing a Communication Link Between MicroBlaze and ARM

In a Zynq device, ARM constitutes the PS side and MicroBlaze belongs to the PL side. The PS is generally used to control the operations performed by the PL. Before delving into communication specifically between MicroBlaze and ARM, a thorough understanding

of different types of interfaces between PS and PL helps to guide the design. The Zynq PS and PL are interconnected via the interfaces mentioned in Table 6.2 and shown in Figure 6.4.

Table 6.2: PS-PL interfaces in Zynq

Two 32-bit master AXI ports (PS master)
Two 32-bit slave AXI ports (PL Master)
Four 32/64-bit slave high performance ports (PL Master)
One 64-bit slave Accelerator Coherency Port (ACP) (PL Master)
Four clocks from the PS to the PL
PS to PL interrupts
PL to PS interrupts
DMA peripheral request interfaces

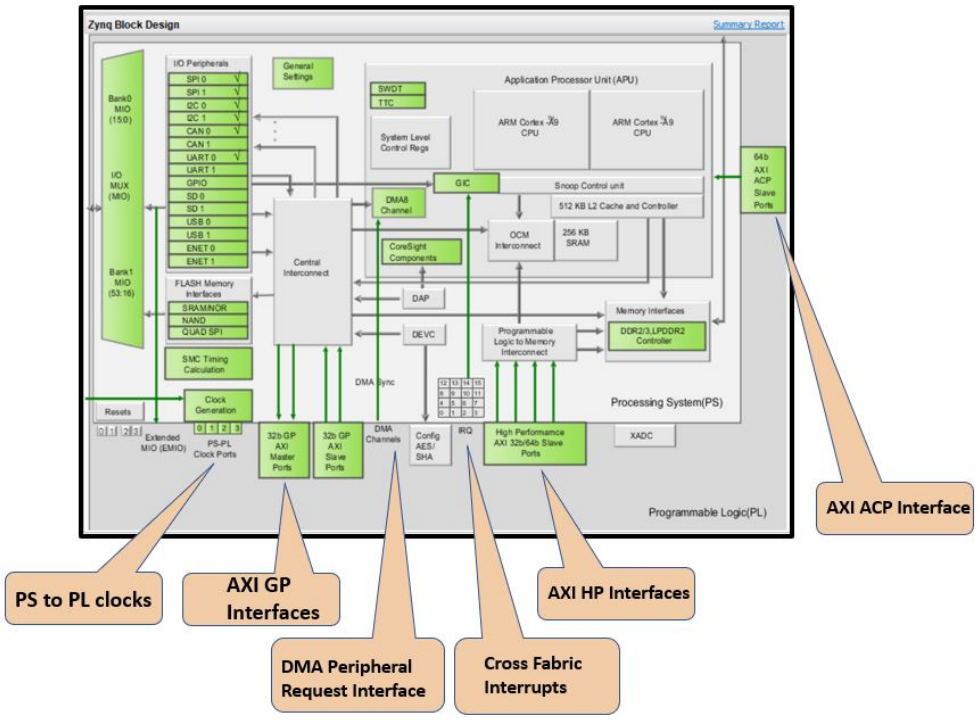


Figure 6.4: PS-PL Interface

Three different methods of communication between MicroBlaze and Zynq were explored. Each implementation is described in the following sections.

6.4.3 Zynq and MicroBlaze Sharing Data via Shared Memory

As shown in Figure 6.5, in this implementation, BRAM acts as the shared memory and is used as a Dual Port RAM (DPRAM), where each port is separately controlled by ARM and MicroBlaze. They access the respective DPRAM ports via Master AXI Interfaces and a BRAM controller. Some of the BRAM features are listed in Table 6.3.

Table 6.3: BRAM features

Each unit stores up to 36 Kbits of data.
Dual port memory with separate read/write port.
Can be configured for different data widths 16Kx1, 8Kx8, 4Kx4 and so on.
Multiple blocks can be cascaded to create larger memory.
Can function as single-port memory.
Maximum data path width of the block RAM is 18 bits.
Can be implemented as Random Access Memory (RAM), Read Only Memory (ROM) or First In First Out (FIFO) buffers.

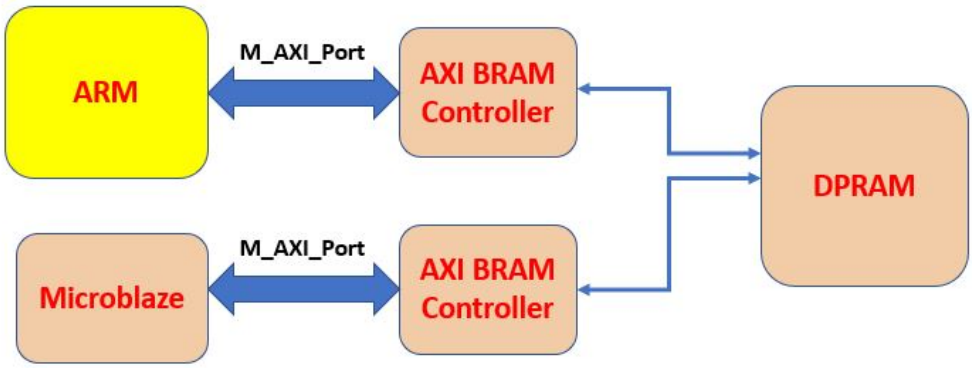


Figure 6.5: Shared memory via BRAM implementation

The block memory generator is the IP core used to generate area and performance optimized memories using embedded BRAM resources in Xilinx FPGAs. The AXI BRAM Controller is a soft Xilinx IP core that allows other Master IP cores to communicate with local BRAMs via AXI Interconnect blocks.

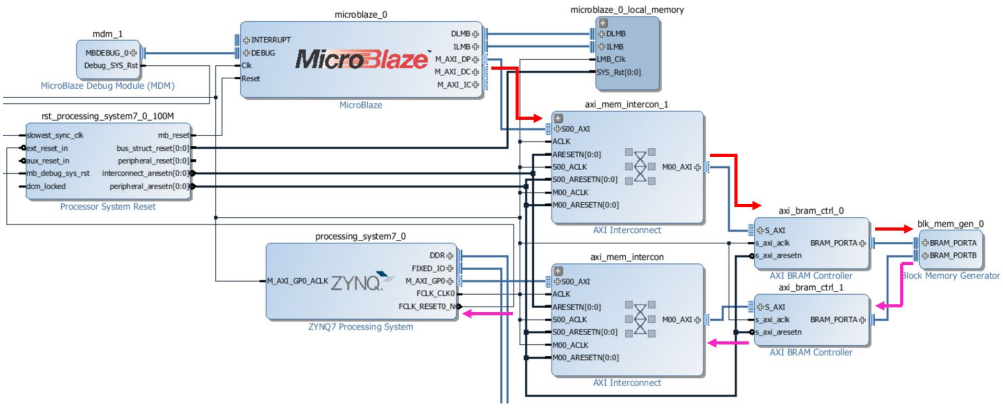


Figure 6.6: Vivado BD for Shared Memory Implementation

While this implementation (shown in Figure 6.6) works smoothly for a one-time sequential read/write operation, the overheads involved in data synchronization for continuous read and write operations make it susceptible to race condition which can lead to data corruption.

6.4.4 Zynq and MicroBlaze Connected via Master and Slave AXI Ports

As shown in Figure 6.7, in this implementation we forgo the BRAM in the FPGA fabric and both the processors communicate with each other directly via master-slave relationship. AXI Interconnect IP acts as the bridge between the two processors. The Vivado BD for this implementation is shown in Figure 6.8.

While this method allows for dataflow from MicroBlaze to ARM, it does not suit transactions where ARM initiates the data transfer. The problem with this implementation is, it is unidirectional (MicroBlaze to ARM) and requires additional AXI interfaces for sending data from the ARM to MicroBlaze. This is an inefficient implementation in terms of FPGA resource usage.

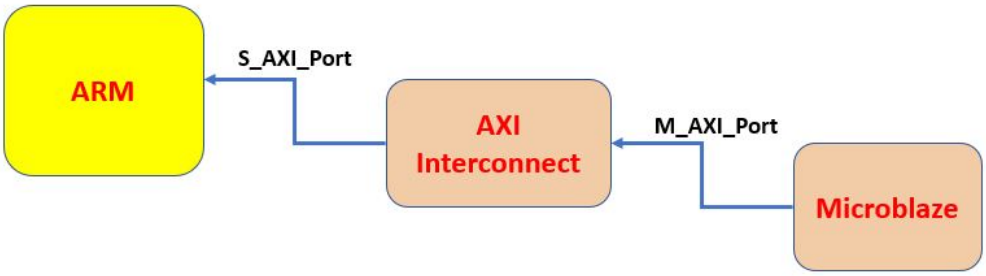


Figure 6.7: Direct master slave implementation

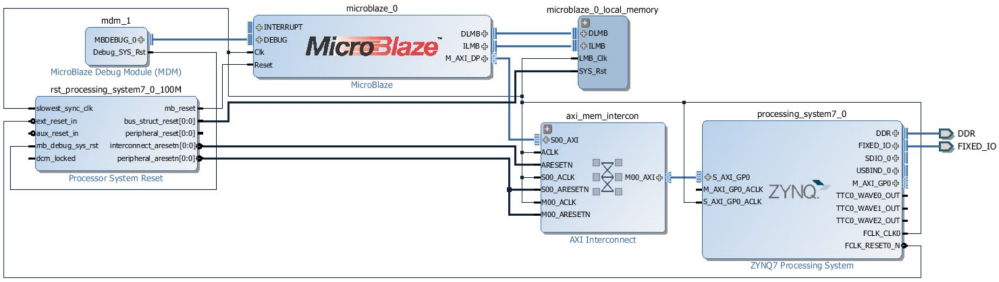


Figure 6.8: Vivado BD for the master slave implementation

6.4.5 Zynq and MicroBlaze Connected via Mailbox

The Mailbox IP, as shown in Figure 6.9, is built specifically for inter-processor communication and is implemented as a synchronous, bidirectional FIFO queue in the PL.

> "The mailbox can be considered a simplified, TCP/IP-like message channel between the processors. The reception of the message at the end of the receiver may be done in a synchronous or asynchronous fashion. In the synchronous method, the receiver actively keeps polling the mailbox for new data. In the asynchronous method, the mailbox sends an interrupt to the receiver upon the presence of data in the mailbox. [11]".

Key features of Mailbox IP are listed in Table 6.4.

As shown in Figure 6.10, in this implementation ARM and MicroBlaze interact

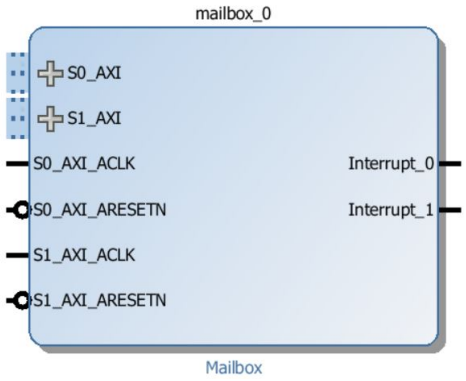


Figure 6.9: Mailbox IP block instantiation in Vivado

Table 6.4: Key features of Mailbox

Supports AXI4-Lite, AXI4-Stream, PLB v4.6 and FSL independently on each of the ports
Configurable depth of mailbox
Configurable interrupt thresholds and maskable interrupts
Configurable synchronous or asynchronous operation
Bi-directional communication
Can also be used to generate interrupts between the processors
Suited for small to medium size messages less than a few 100 Bytes

through the Mailbox via their respective AXI Interconnect IPs. This is an efficient design since data can flow simultaneously in both the directions without additional or separate AXI ports. Since the Mailbox IP has built-in FIFOs to hold the data, it eliminates the need for read/write address synchronization. The Vivado BD for this implementation is shown in 6.11. The Mailbox is an ideal solution for a bidirectional communication channel between two processors.

The Mailbox is a processor and OS independent IP core. There are two ways of communicating using a Mailbox, (1) either through blocking or (2) non-blocking read/write transactions. Blocking read/writes are not suitable since the requesting processor will stall until the requested bytes of data are received.

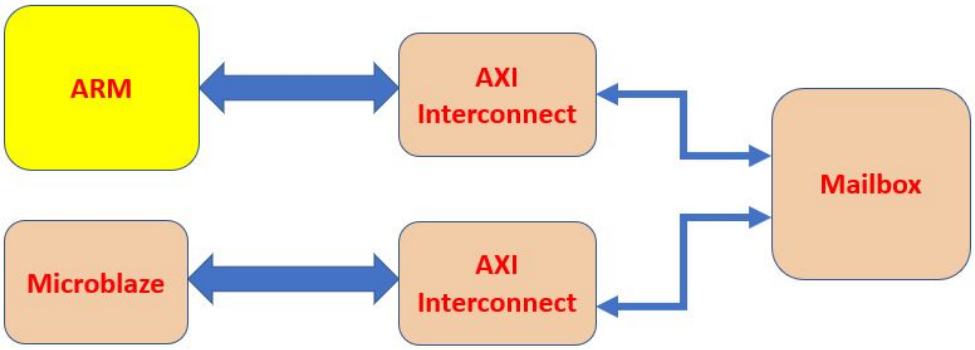


Figure 6.10: Zynq-MicroBlaze communication via Mailbox

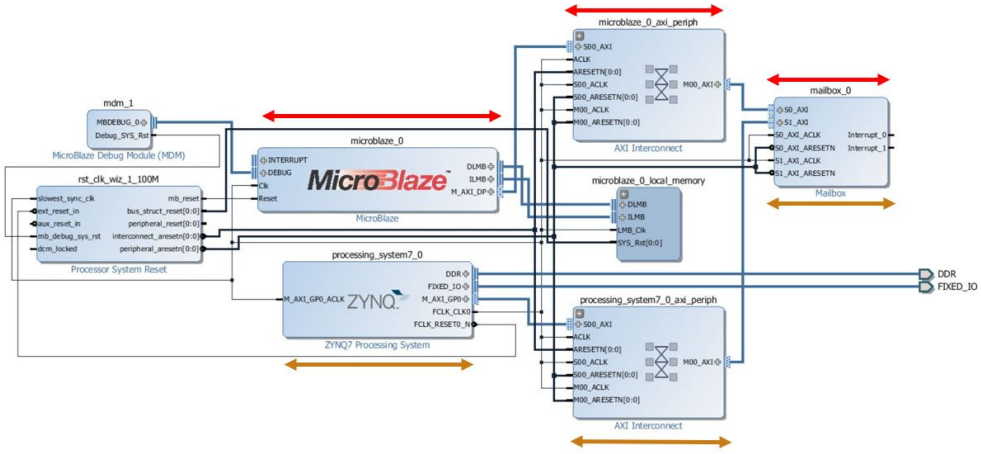


Figure 6.11: Vivado BD for Mailbox implementation

6.4.6 Interprocessor Data Communication Verification in Software

Once the hardware components were finalized, the next step in the design process was to create software components. The application specific code for both MicroBlaze and ARM will be executed on both processors simultaneously. Software development is done on Xilinx Software Development Kit (SDK) tool. The SDK application development flow is shown in Figure 6.12.

While the design flow works for application development on single processor systems, the challenge was to boot both the applications on MicroBlaze and ARM processor simulta-

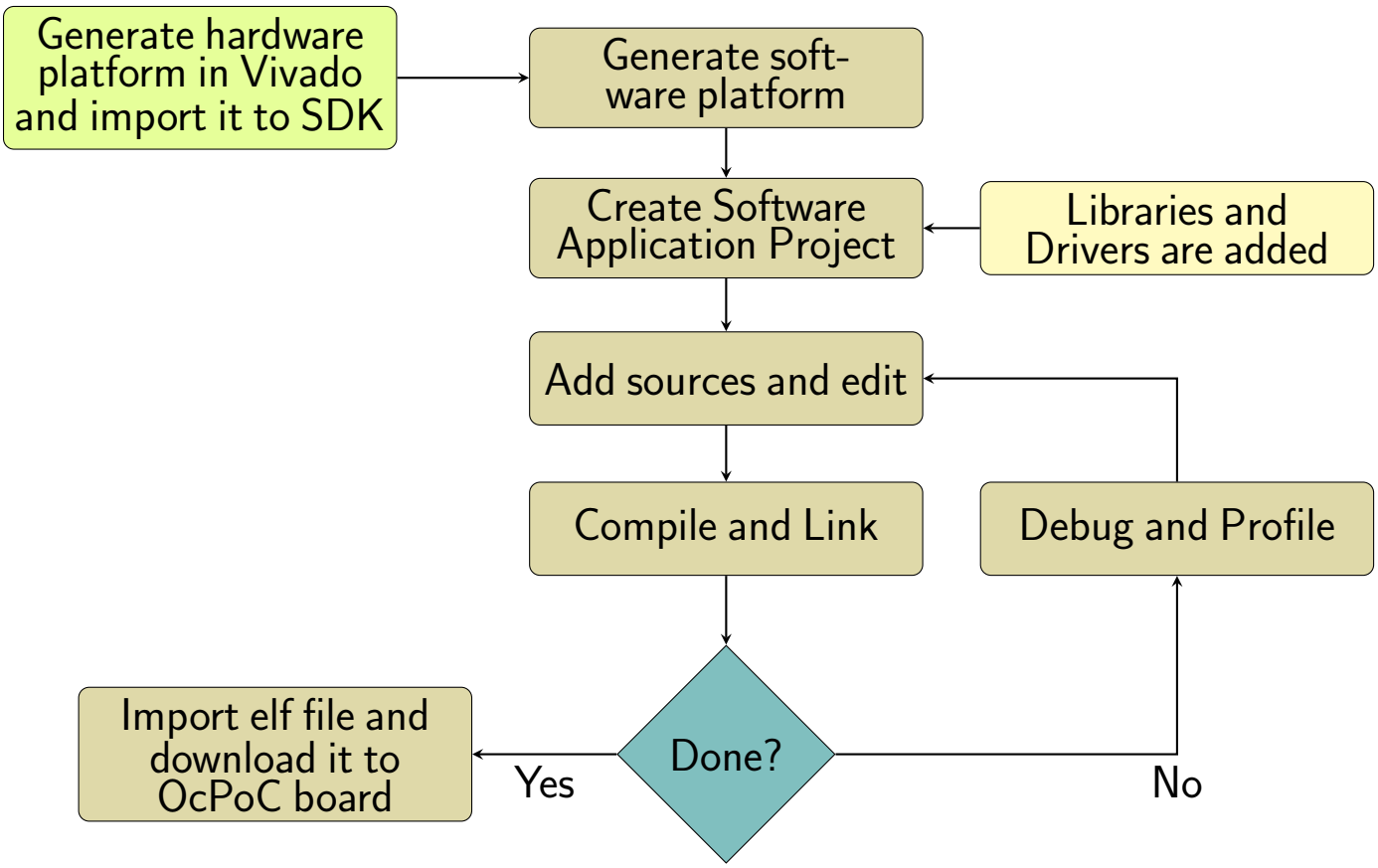


Figure 6.12: Software development flow

neously. Programming one processor resets the other processor. This happens because the FPGA erases the previous processor data every time a new .elf file is loaded. To resolve this problem, the .elf file generated for the application running on the MicroBlaze is integrated with the earlier Vivado generated bitsream, to create a new bitstream, as shown in Figure 6.13. This newly generated bitstream is used to program the PL side of Zynq followed by executing the .elf file for the application running on the ARM. The SDK development flow for applications running on both MicroBlaze and ARM is shown in Figure 6.14. The application C codes for both MicroBlaze and ARM are written in Xilinx SDK and a simplified flowchart showing software development using mailbox IP is shown in Figure 6.15.

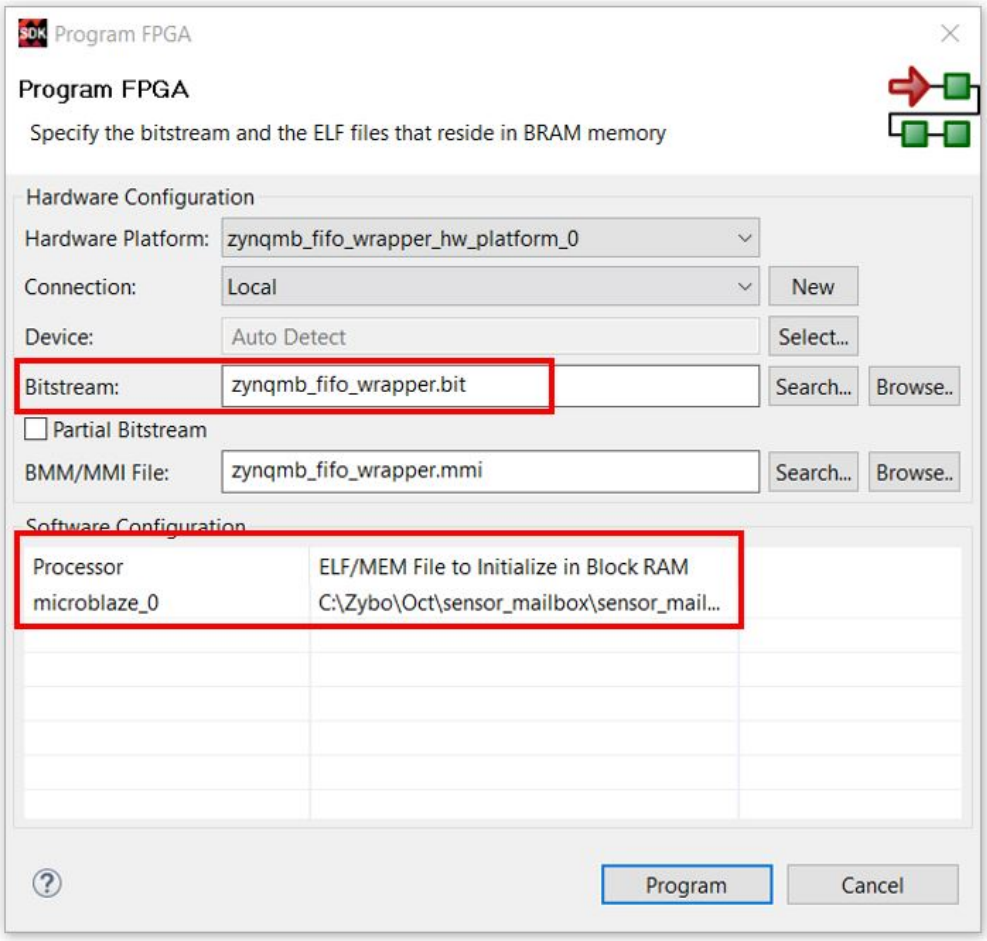


Figure 6.13: Loading dual software applications on ARM and MicroBlaze

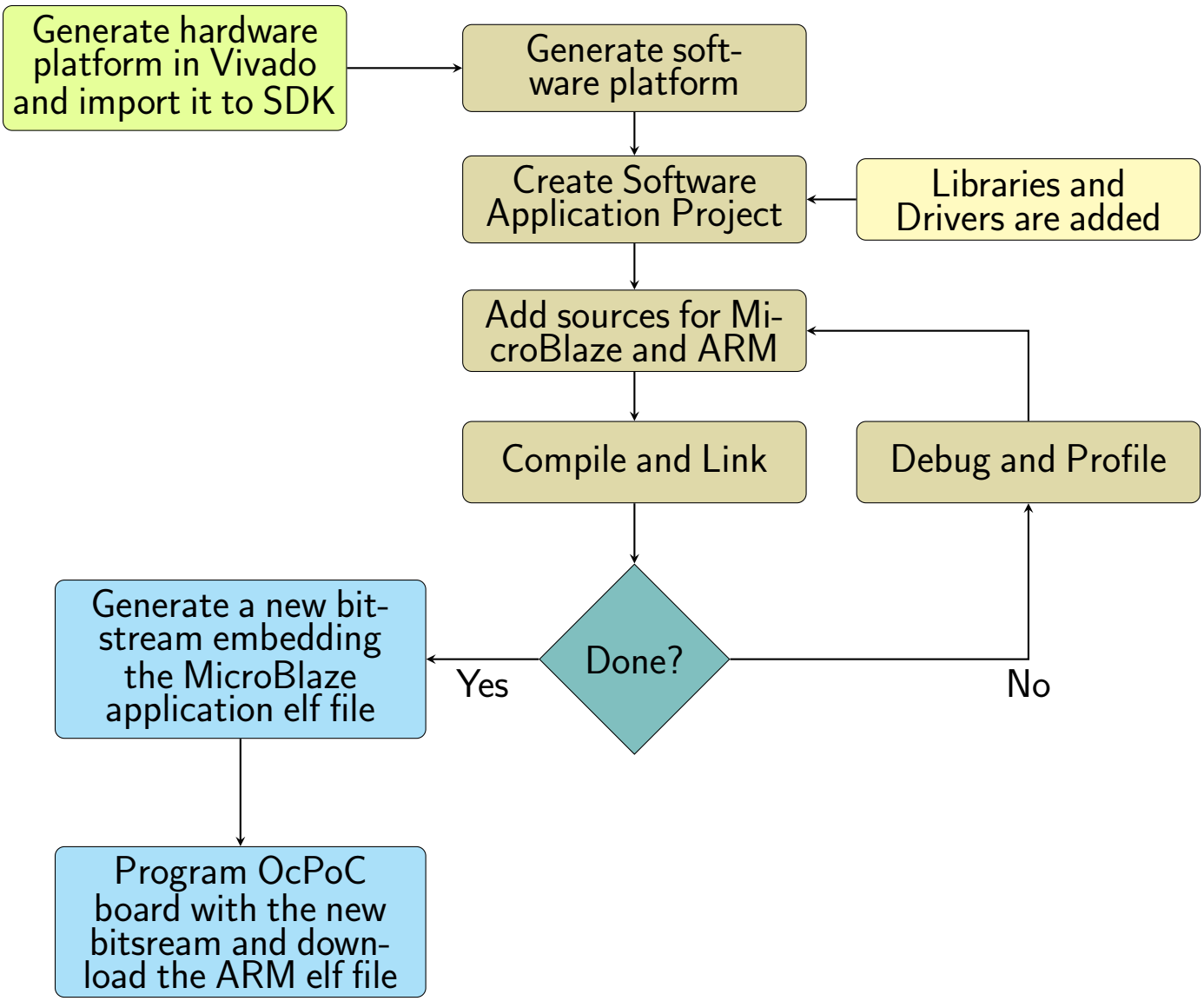


Figure 6.14: Software development flow with 2 processors

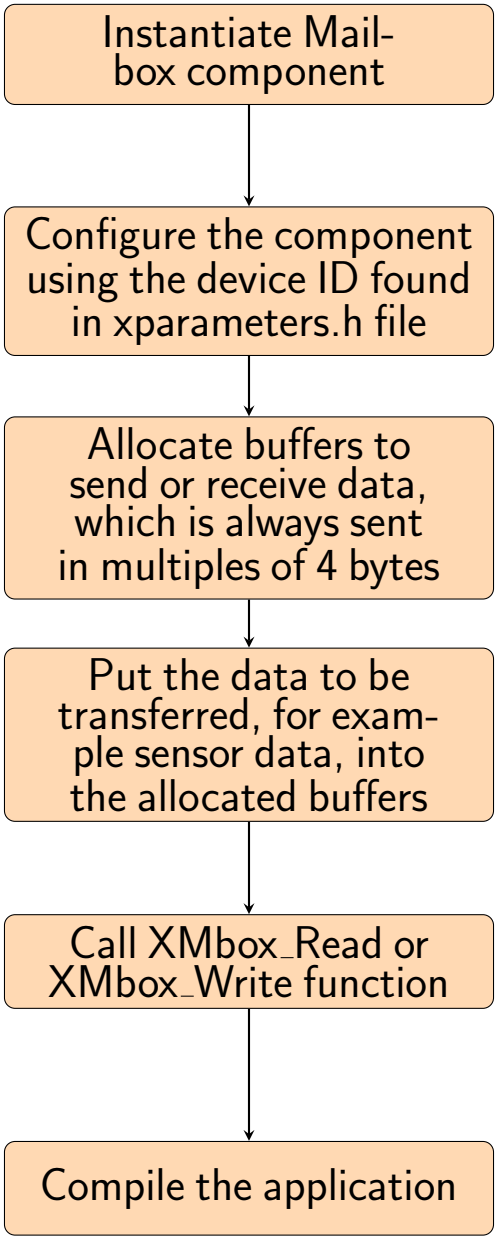


Figure 6.15: Xilinx SDK application development flow for Mailbox IP

Chapter 7

Results

The experimental setup involved interfacing OcPoC board with the following components: (i) UBlox GPS + Compass module, (ii) Servos for aileron, throttle, rudder and elevator, (iii) MS4525D0 Airspeed sensor, (iv) Mission Planner software running on laptop, (v) RC Transmitter by Futaba. (vi) RC Receiver by Futaba. (Pictures of the components are shown in Appendix A). The Vivado Suite version 2016.2 was used for hardware and software development.

7.1 Conducted Successful Flight Tests with Customized ArduPlane code on OcPoC

Two flight tests were conducted with different flight envelopes and flight durations. This verified the capability of the ArduPlane software application running on a custom made Zynq hardware board for the first time. Aerotenna had previously verified only ArduCopter, which is for a rotary wing aircraft and not the ArduPlane application. The vehicle dynamics between a fixed wing and a rotary wing aircraft vary and this experiment established that a Zynq-based flight controller could be used for fixed wing aircrafts.

The data logger software module added to the ArduPlane application was able to efficiently capture the parameters and log them to a timestamped file during each flight test.

This will be a useful feature for future flight tests.

7.2 Verified Real Time External Data Communication between MicroBlaze and ARM

Generating Vivado BD and subsequent software application development is performed in incremental iterations. An SoC-FPGA board can be programmed in three ways, (1) JTAG (2) SD card (3) QSPI flash. JTAG programming is best suited for quick physical verification of the implemented design on the board. Programming via the SD Card or QSPI flash is time consuming and results in staggered testing, hindering development. The OcPoC board requires custom JTAG cables and an adapter to perform JTAG programming. However, due to nonavailability of the cables at the time of testing and to quickly verify the proof of concept, real time interprocessor data communication was verified on ZYBO development board.

ZYBO has the same Zynq device (i.e. xc7z010clg400-1) as that found on the OcPoC board. An external sensor, a three-axis digital accelerometer, was connected to ZYBO, as shown in Figure 7.1. The incoming sensor data was read in real time via MicroBlaze and transmitted to the ARM processor. No data loss or corruption was observed. The Vivado BD for this implementation is shown in Figure 7.2.

7.3 Integration with OcPoC

Once the proof of concept was demonstrated on Zybo, the idea was extended to OcPoC. The MicroBlaze and Mailbox IPs were successfully incorporated to the existing OcPoC hardware configuration and transfer of synthetic data between MicroBlaze and ARM

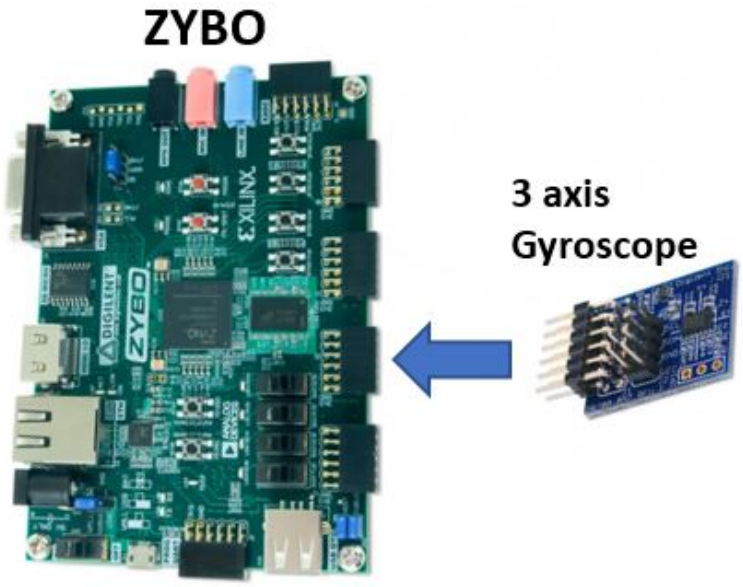


Figure 7.1: Proof of Concept verified on zybo

was verified.

7.3.1 Mailbox Configuration for OcPoC

AXI-Lite was the choice of interface between the Mailbox IP and the processors since it caters to the bandwidth needed for transferring sensor data between processors. A clock frequency of 100MHz was used to clock both the FIFOs. Mailbox FIFO depth was kept at 16 words.

7.3.2 Adding Mailbox IP to OcPoC Vivado BD

A snapshot of the OcPoC Vivado BD containing MicroBlaze is shown in Figure 7.3. It is difficult to capture the entire BD in a high resolution image due to the dense connectivity between numerous components. The memory mapped addresses of Mailbox and MicroBlaze in the address space of OcPoC is shown in Figure 7.4.

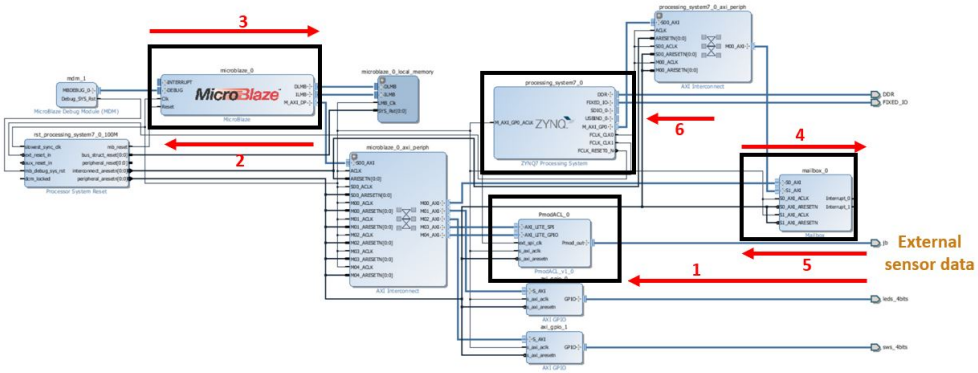


Figure 7.2: Vivado BD showing the data path of external sensor data flowing from MicroBlaze to ARM

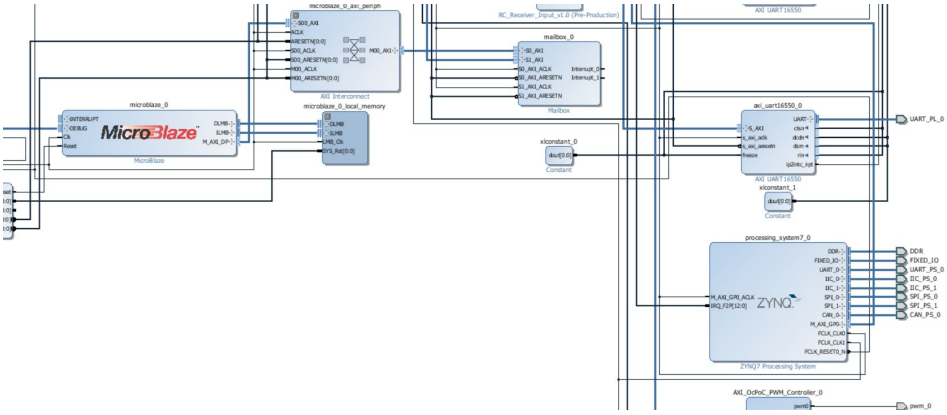


Figure 7.3: OcPoC Vivado BD containing MicroBlaze

7.3.2.1 Resource Utilization

A comparison of the FPGA resource utilization for the original OcPoC configuration and the new configuration containing the Mailbox IP and MicroBlaze is shown in Figure 7.5. The number of BRAMs used in the PL is the main addition to the existing resource usage. This is attributed to the memory needed to run the MicroBlaze application and the Mailbox IP FIFOs both of which are implemented as BRAMs. The slight increase in LUTs and Flip Flops (FFs) usage is attributed to the additional connectivity overheads to accommodate two new IP cores. The overall percentage increase in LUTs, BRAMs and FFs with the addition

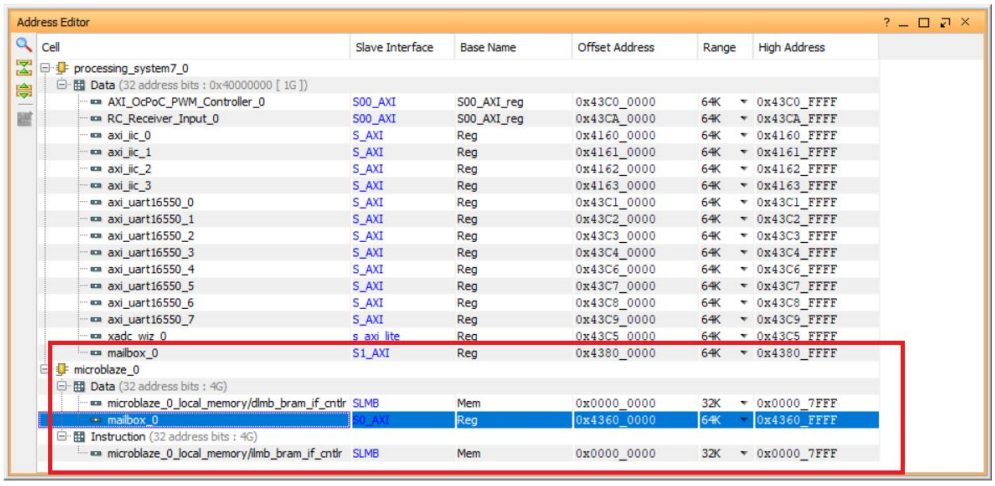


Figure 7.4: Address Editor pane with MicroBlaze and Mailbox

of these two IPs still permits the addition of other soft IP cores. The hardware resource utilization is restricted by the specific Zynq device used in OcPoC and is not a limitation of this technology since other larger Zynq devices can always be used.

7.3.2.2 Power Consumption

The On-Chip Power panel is the total power consumed within the device. It includes device static power and user design dependent static and dynamic power. The statistics given in Table 7.1 show that the addition of a soft-core processor and a new IP core increases the power by **0.17%**. Power is a critical factor in embedded systems. Any security-related feature addition should not lead to high power dissipation. Our new design increases the power dissipation within acceptable limits.

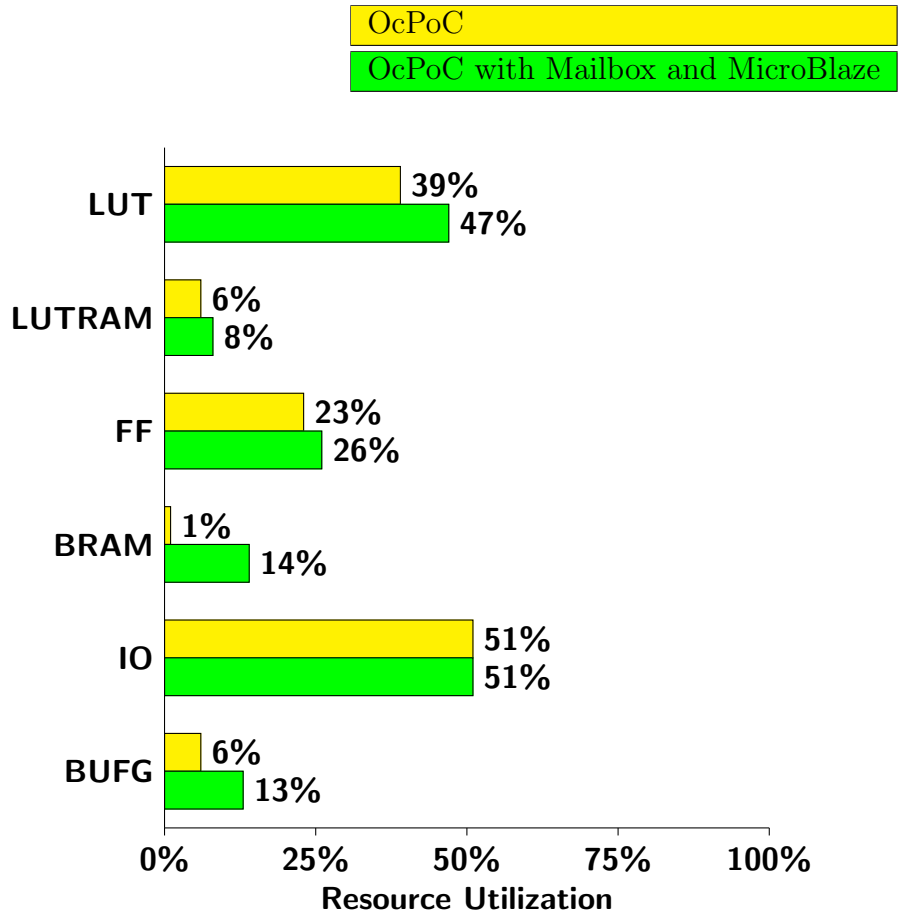


Figure 7.5: Post Implementation Zynq FPGA resource usage with and without MicroBlaze and Mailbox

Table 7.1: Estimated Power Consumption

	OcPoC	**OcPoC with Mailbox and MicroBlaze**
Dynamic Power	1.596	1.603
Static Power	0.134	0.13
Total Power	**1.73**	**1.733**

Chapter 8

Conclusions

8.1 Summary

An embedded Linux operating system combined with an SoC-FPGA device offers a versatile platform for rapid prototyping of real time systems. In this project, a relatively new, state-of-the-art technology has been chosen for a UAV platform. The ability to skillfully integrate all the avionics related to a flight controller on an SoC-FPGA board is explored and accomplished. In addition to hardware, an open source flight controller software program with a large code base was also successfully ported to this platform. The OcPoC board was validated with a successful flight test. UAS are susceptible to a variety of threats. With the end goal of enhancing security, a trusted and isolated I/O processor has also been integrated with OcPoC.

8.2 Future Work

The attempt here has been to open a new trusted gateway within the FPGA fabric to monitor sensor data. After the air speed sensor, GPS and IMUs can be channeled through the I/O processor, their data being screened for any form of sensor spoofing. The ample FPGA resources can also be utilized to develop custom IP cores for monitoring sensor data in case an entire MicroBlaze processor is not needed. The flexibility of implementing various different kinds of hardware monitors can be explored.

This form of embedded platform in the domain of UAVs is an emerging technology. An SoC-FPGA provides almost unlimited processing power with the advantage that all noncritical tasks and hardware acceleration can be pushed to the FPGA fabric. This opens the door to implementing highly efficient image and video processing algorithms on these boards. In addition, techniques such as formal verification can be applied to trusted hardware cores implemented in the FPGA fabric.

Bibliography

[1] https:https://www.xilinx.com/support/documentation/application_notes/xapp1239-fpga-bitstream-encryption.pdf.

[2] https://aerotenna.com/.

[3] http://ardupilot.org/dev/docs/learning-ardupilot-introduction.html, .

[4] https://en.wikipedia.org/wiki/ArduPilot, .

[5] https:https://www.xilinx.com/support/documentation/white_papers/wp365_Solving_Security_Concerns.pdf.

[6] https://issuu.com/xcelljournal/docs/xcell94/34.

[7] https://oscarliang.com/quadcopter-pid-explained-tuning/, October 2013.

[8] https://www.roboticstomorrow.com/article/2015/12/five-open-source-autopilot-uav-projects/7436/, December 2015.

[9] Malcolm J Airst and Senior Principal Engineer. GPS Network Timing Integrity. In *c3I Integration and Interoperability CAPSTONE program*, 2010.

[10] Todd W Arnold, C Buscaglia, F Chan, Vincenzo Condorelli, J Dayka, W Santiago-Fernandez, Nihad Hadzic, Michael D Hocker, M Jordan, TE Morris, et al. IBM 4765 Cryptographic Coprocessor. *IBM Journal of Research and Development*, 56(1.2):10–1, 2012.

[11] Vasanth Asokan. Designing Multiprocessor Systems in Platform Studio. *White Paper: Xilinx Platform Studio (XPS)*, pages 1–18, 2007.

[12] Reg Austin. *Unmanned Aircraft Systems: UAV's Design, Development and Deployment*, volume 54. John Wiley & Sons, 2011.

[13] Louise H. Crockett, Ross A. Elliot, Martin A. Enderwitz, and Robert W. Stewart. *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*. Strathclyde Academic Media, UK, 2014. ISBN 099297870X, 9780992978709.

[14] Blake Fuller, Jonathan Kok, Neil A Kelson, and Luis F Gonzalez. Hardware Design and Implementation of a MAVLink interface for an FPGA-based Autonomous UAV Flight Control System. In *Proceedings of the 2014 Australasian Conference on Robotics and Automation*, pages 1–6. Australian Robotics & Automation Association ARAA, 2014.

[15] Manu Gulati, Michael J Smith, and Shu-Yi Yu. Security Enclave Processor for a System on a Chip, September 9 2014. US Patent 8,832,465.

[16] Kim Hartmann and Christoph Steup. The vulnerability of UAVs to Cyber Attacks-An Approach to the Risk Assessment. In *Cyber Conflict (CyCon), 2013 5th International Conference on*, pages 1–23. IEEE, 2013.

[17] Ran Ji, Jian Wang, Chaojing Tang, and Ruilin Li. Automatic Reverse Engineering of Private Flight Control Protocols of UAVs. *Security and Communication Networks*, 2017, 2017.

[18] Andrew J Kerns, Daniel P Shepard, Jahshan A Bhatti, and Todd E Humphreys. Unmanned Aircraft Capture and Control via GPS Spoofing. *Journal of Field Robotics*, 31 (4):617–636, 2014.

[19] Lyes Khelladi, Yacine Challal, Abdelmadjid Bouabdallah, and Nadjib Badache. On

security issues in embedded systems: Challenges and solutions. *International Journal of Information and Computer Security*, 2(2):140–174, 2008.

[20] Seajin Kim, Byung-jin Lee, Jae-won Jeong, and Myeong-jin Lee. Multi-object Tracking Coprocessor for Multi-channel Embedded DVR Systems. *IEEE transactions on Consumer Electronics*, 58(4), 2012.

[21] Fang-Chen Kuo, Yeim-Kuan Chang, and Cheng-Chien Su. A Memory-efficient TCAM Coprocessor for IPv4/IPv6 Routing Table Update. *IEEE Transactions on Computers*, 63(9):2110–2121, 2014.

[22] Peng Li and Hua Tang. Design of a Low-power Coprocessor for Mid-size Vocabulary Speech Recognition Systems. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(5):961–970, 2011.

[23] John Narayan, Sandeep K. Shukla, and T. Charles Clancy. A Survey of Automatic Protocol Reverse Engineering Tools. *ACM Comput. Surv.*, 48:40:1–40:26, 2015.

[24] Patrick Schaumont and Zhimin Chen. Side-Channel Attacks and Countermeasures for Embedded Microcontrollers. In *Introduction to Hardware Security and Trust*, pages 263–282. Springer, 2012.

[25] Yunmok Son, Hocheol Shin, Dongkwan Kim, Young-Seok Park, Juhwan Noh, Kibum Choi, Jungwoo Choi, Yongdae Kim, et al. Rocking Drones with Intentional Sound Noise on Gyroscopic Sensors. In *USENIX Security Symposium*, pages 881–896, 2015.

[26] Randy Torrance and Dick James. The state-of-the-art in semiconductor reverse engineering. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 333–338. IEEE, 2011.

[27] Stephen M Trimberger and Jason J Moore. Fpga security: Motivations, features, and applications. *Proceedings of the IEEE*, 102(8):1248–1265, 2014.

[28] Sebastian Wallat, Marc Fyrbiak, Moritz Schlögel, and Christof Paar. A Look at the Dark Side of Hardware Reverse Engineering-A Case Study. In *Verification and Security Workshop (IVSW), 2017 IEEE 2nd International*, pages 95–100. IEEE, 2017.

[29] Jon S Warner and Roger G Johnston. A Simple Demonstration That the Global Positioning System (GPS) is Vulnerable to Spoofing. *Journal of Security Administration*, 25(2):19–27, 2002.

[30] Jon S Warner and Roger G Johnston. GPS Spoofing Countermeasures. *Homeland Security Journal*, 25(2):19–27, 2003.

[31] Fabian Weise. Reverse Engineering of Microcontrollers. *Proseminar Microcontrollers and Embedded Systems WS14*, 15.

[32] Kyle Wesson, Daniel Shepard, and Todd Humphreys. Straight Talk on Anti-Spoofing. *GPS World, volume=23, number=1, pages=32–39, year=2012.*

[33] Bing-Fei Wu and Chung-Fu Lin. An Efficient Architecture for JPEG2000 Coprocessor. *IEEE Transactions on Consumer Electronics*, 50(4):1183–1189, 2004.

[34] Xilinx. Reference Guide, ug761 (v13. 1). *URL http://www. xilinx. com/support/documentation/ip documentation/ug761 axi reference guide. pdf*, 2011.

[35] Xilinx. MicroBlaze Processor Reference Guide. October 2016.

Appendices

Appendix A

Pictures of the Components Used

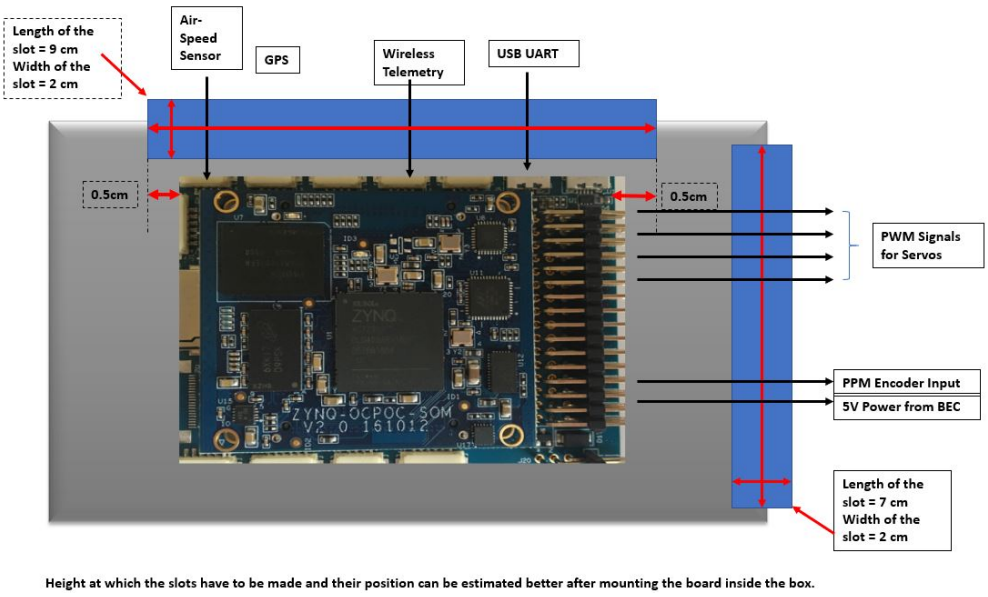


Figure A.1: Board measurements with interfaces

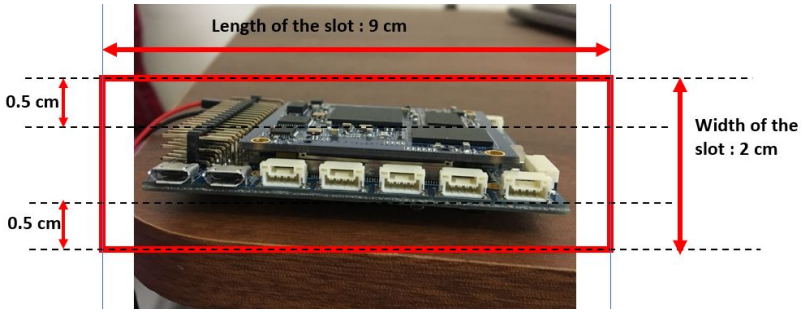
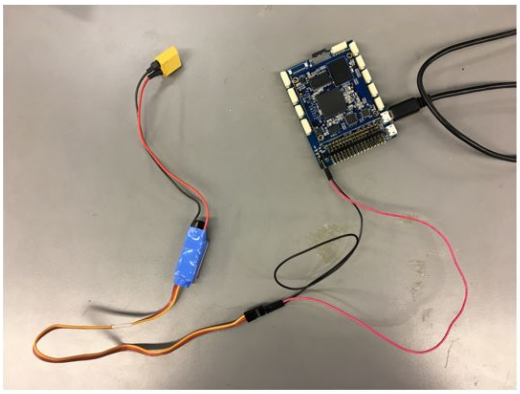


Figure A.2: Length and width of slots

Servo used during
flight tests



Battery connection for powering on OcPoC

Figure A.3: Servo used and battery connection for OcPoC

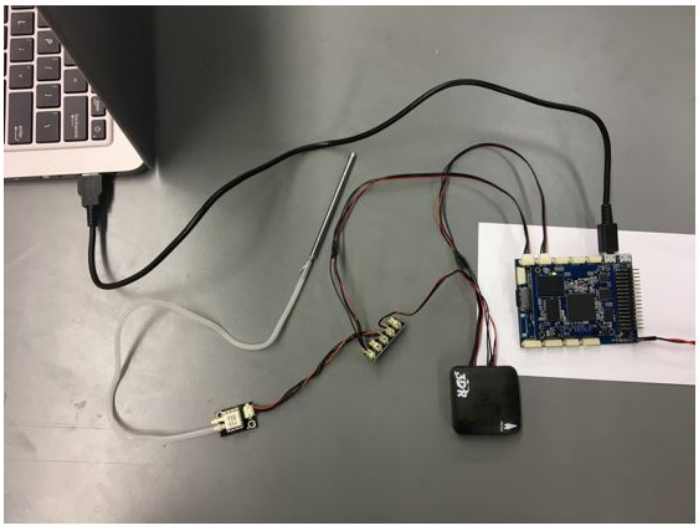


Figure A.4: GPS and airspeed sensor used during the flight tests

Figure A.5: JTAG interface for OcPoC

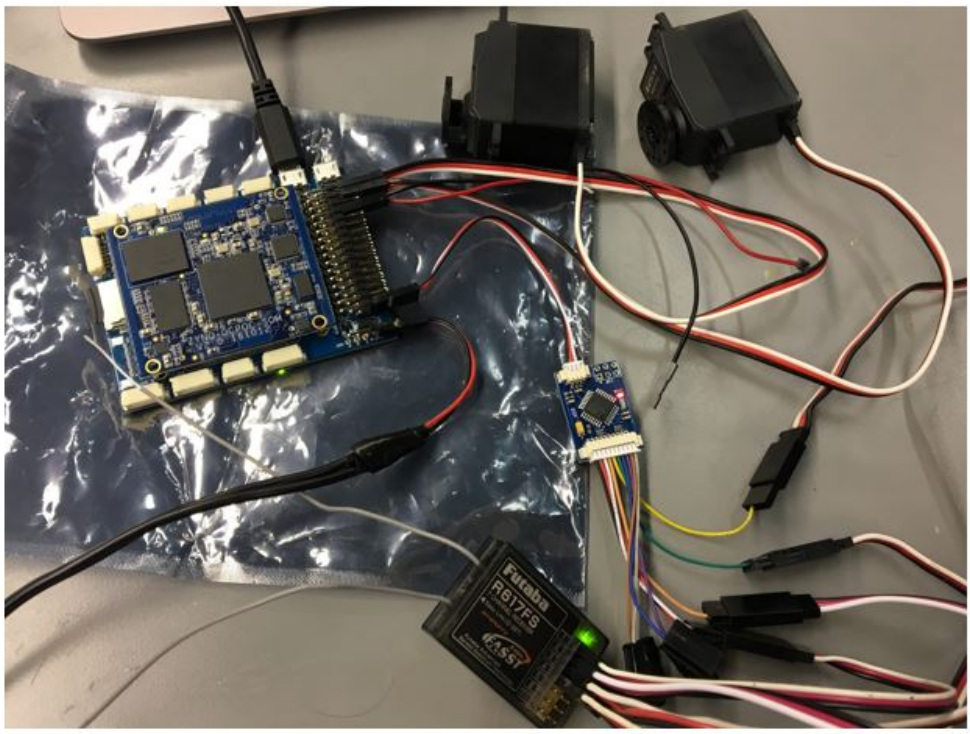


Figure A.6: Receiver, PPM encoder and servos