15L513–INTERNSHIP & INNOVATION PRACTICES

DEVELOPMENT OF RISC V ISA BASED FLIGHT CONTROLLER USING SHAKTI MICROPROCESSOR - CONTROL SYSTEM DESIGN

Shrihari G (18L251)

Project report submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: ELECTRONICS AND COMMUNICATION ENGINEERING

Of Anna University

NOVEMBER 2020

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

DEVELOPMENT OF RISC V ISA BASED FLIGHT CONTROLLER USING SHAKTI MICROPROCESSOR - CONTROL SYSTEM DESIGN

Bonafide record of work done by

Shrihari G (18L251)

Project report submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: ELECTRONICS AND COMMUNICATION ENGINEERING

Of Anna University

NOVEMBER 2020

.....	.....	.....
Mr. S. Sainath	**Mr. K. R. Radhakrishnan**	**Dr. V. Krishnaveni**
Faculty guide	Faculty guide	Head of the Department

Certified that the candidate was examined in the viva-voce examination held on .....

.....
(External Examiner)

CONTENTS

CHAPTER **Page No.**

ACKNOWLEDGEMENT

We extend our sincere thanks to **Dr. K. PRAKASAN**, Principal, PSG College of Technology, for his kind patronage.

We are indebted to **Dr. V. KRISHNAVENI**, Professor and Head, Department of Electronics and Communication Engineering, for her continued support and motivation.

We express our deepest gratitude to our project guide, **Mr. S. SAINATH**, Assistant Professor, Department of Electronics and Communication Engineering, for his constant motivation, direction and guidance throughout the course of our project work.

Our sincere thanks to our tutor and guide, **Mr. K.R. RADHAKRISHNAN,** Assistant Professor, Department of Electronics and Communication Engineering, whose valuable guidance and continuous encouragement throughout the course made it possible to complete this dissertation work successfully.

Finally, we thank all our student colleagues who have helped us in completing this project without any hassles.

ABSTRACT

Unmanned Aerial Vehicles have become increasingly popular in a plethora of disciplines ranging from cinematography, transportation, and healthcare, critical applications like defense, search and rescue etc., the security, customizability and the reliability of UAVs for such applications play a vital role. This emphasizes the adoption of an open source royalty free Instruction Set Architecture, RISC-V. This allows for maximum scalability and flexibility in modifying the instruction set according to the requirements, contrary to the proprietary instruction set by ARM. Shakti microprocessors, an open source microprocessor initiative by RISE Labs, IIT Madras solves this issue by providing an open source fully customizable processor cores to suit the needs of the developer. The project aims at designing and developing an indigenous multipurpose, fully customizable flight controller using Vajra C64 microprocessor realized in Arty 7-100T FPGA board. The first step in developing a flight controller is the control systems for the UAV. This flight controller design could be easily adopted to other RISC-V based microprocessors and the control system design is also made to be processor independent by the use of SIMULINK Aerospace Block set with Flight Code generated from the SIMULINK coder. The developed control system and the proposed fight controller would prove to be an effective solution against common security flaws in proprietary processors like backdoor exploits. The indigenously developed flight controller emphasizes the “Make in India” campaign and the “Aatmanirbar Bharat” initiative.

LIST OF FIGURES

1. INTRODUCTION

The usage of Flight Controllers plays a vital part in developing autonomous and semi-autonomous aerial vehicles, which are used in a variety of domestic, industrial and military applications. The flight controller can be used for varied purposes ranging from an indigenously developed innovative domestic prototype to a commercial product. This covers the drones used in remote areas to deliver packages, medical related transportation to inaccessible areas, military surveillance, disaster mitigation and farming.

By using RISC-V based Shakti Processors, which are developed in-house, allows for the identification of security flaws and patching them quickly. The transparency in the nature of open source cores makes it a suitable candidate for deployment into mission critical applications. Moreover, the development of a RISC-V ISA based flight controller allows the flexibility of designing a dedicated RISC V based microprocessor for the application without having to completely redo the coding for the flight controller.

The use of Shakti C- Class (Vajra C-64) can be extended to perform on board computations, autonomous navigation, Machine Learning etc., in the future after the development of the basic firmware. This solves the connectivity problems and the need of a receiving station is eliminated in the cases where the Unmanned Aerial Vehicle is required to navigate to remote locations.

2. METHODOLOGY

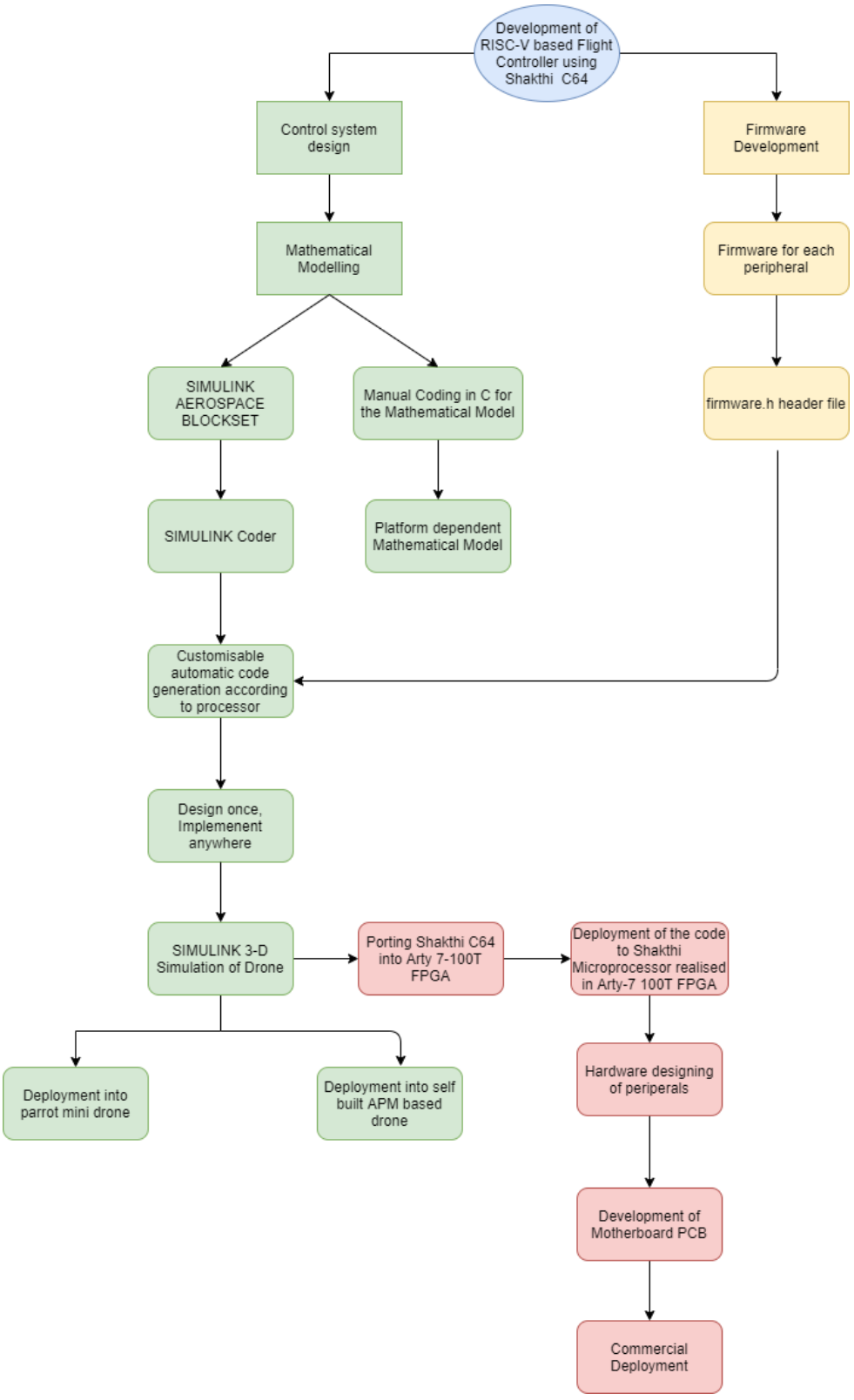


Fig 2.1: Block diagram of the development workflow [2]

The Flight Controller development takes several steps, of which Control System design is the first step. The mathematical modelling of the drone is formulated and then the motor mixing algorithm is coded as follows

$$M_{front,right} = Thrust + Yaw + Pitch + Roll$$
$$M_{front,left} = Thrust - Yaw + Pitch - Roll$$
$$M_{back,right} = Thrust - Yaw - Pitch + Roll$$
$$M_{back,left} = Thrust + Yaw - Pitch - Roll$$

If the force is applied on the center of gravity of the object it will move in pure translational motion and will not undergo any rotation. The motor thrust is designed to be atleast twice the weight of the drone to have a suitable margin of error and maneuverability. When the force is not on the Center of Gravity it undergoes rotational motion in addition to the translational motion due to a resultant torque.

The motors in the opposite extremes are designed to spin in the same direction. This way in the under actuated system we can Yaw without affecting the thrust. The Roll, Pitch and Yaw can be independently controlled by varying the speeds of the motors. To achieve translational motion, two rotational motion components are applied at the same time. i.e., to move forward the drone is made to pitch and thrust is increased. This is the fundamental principle governing the motor mixing algorithm

This is then implemented in SIMULINK Aerospace Block set is verified with the inbuilt simulator. The trajectory is planned and the trajectory is tested to evaluate to verify the behavior of the system to utilize the different maneuvering capabilities. The flight controller code is generated for three platforms

1. Parrot Mini Drone
2. APM based Flight Controller
3. Shakti C64 Vajra Microprocessor based Flight Controller[3]

The generated code is then dumped and tested in Parrot Mini Drone and APM based Flight controller. The workflow of adopting automatic code generation allows for deployment in any processor of choice, contrary to the Embedded C code which is specific for the processor.

In order to dump the code in Vajra C64 processor the mgs file and bit stream from the GitLab Repository[3] of RISE LAB is cloned and uploaded into the Arty 7-100T FPGA. The PlatformIO IDE detects the board and the generated C code with suitable modifications according to the PINMUX mapping of Vajra C64 is dumped[12-17]

The firmware for the peripherals are written. At first I2C is written and the codes are used in PlatformIO IDE[4] to debug and evaluate performance using spiking. The design is verified within the IDE.

3. Control System

Control System design is the first step in the development of a Flight controller. A mathematical model was developed for a Quad-rotor Drone at first and modelled using SIMULINK Aerospace Block set. The control systems also simulates the external environment and receives feedback via the sensors forming a closed loop and controls the actuators to maintain the stability of the system at test. The development of control systems can be done in two ways.

a. Developing and modelling the system natively in Embedded C
b. Developing the model in Simulink Aerospace Block set and generating the flight code with SIMULINK Coder

The later approach is found to faster and simpler. By designing the control systems using SIMULINK it can easily be scaled, modified and improved. This allows the flight control code to be processor independent facilitating the maximum portability of the code. The objective of adopting the SIMULINK based workflow emphasizes on “Design Once, Deploy anywhere”. The flight controller is designed with up to 3 Layers of abstraction. Upon diving deeper into the third layer predefined SIMULINK Blocks available in Aerospace Block Set are used for the mathematical model.

The Control system design takes place in three layers

3.1 Quadcopter model
3.2 Flight Control System
3.3 State Estimator Design
3.4 Thrust Control System

3.1 Quadcopter model

This the top most layer of the design. The flight control system which is the main part of the control system is considered as a sub system and the quadcopter is modelled. The RF Signal to the drone is modelled via the signal editor. The four sensors as in Fig 2.2, are interfaced with the Flight Control System. The Airframe is considered to be non - linear and is modelled via the Airframe Block. The Flight Control System Block which models the drone interacts with the environment which is the Non Linear Airframe. The sensors receive the input from the non - linear airframe. The environment variables interact with the sensors and the sensors measure the current state of the system and provide a feedback to the flight Control System providing for maximum stability and agility.

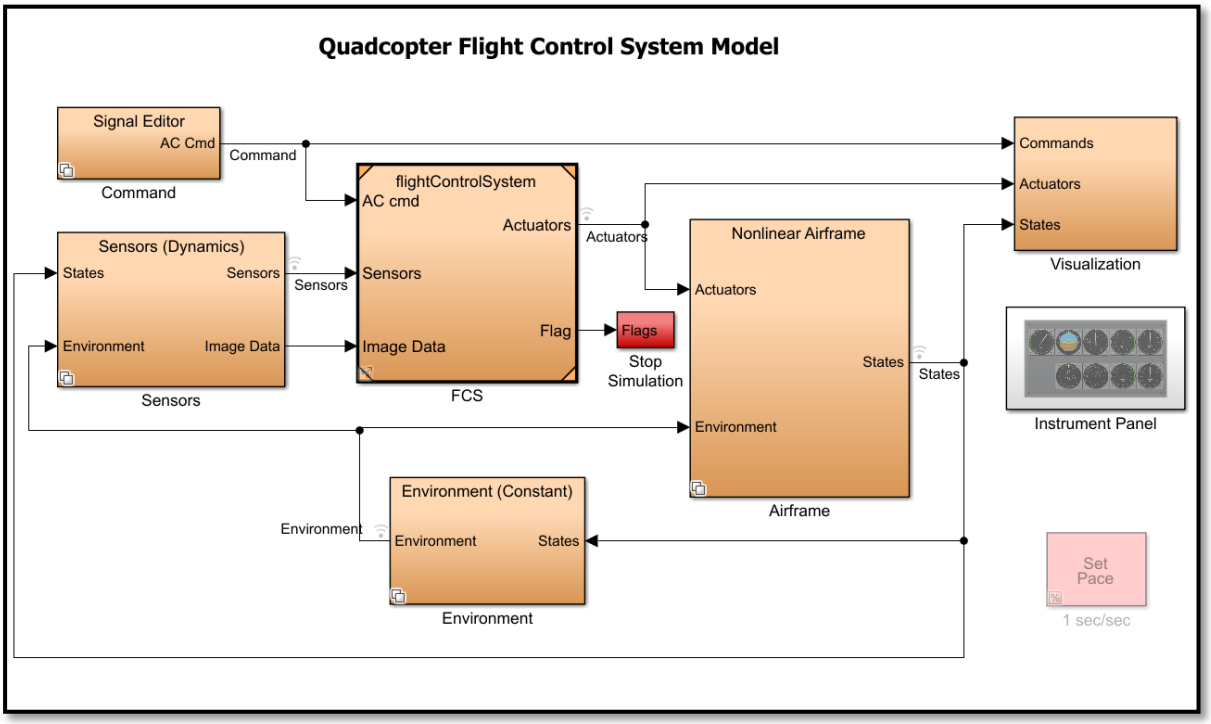


Fig 3.1: Quadcopter Flight Control System Model

The following table illustrates the different sensors used in the flight control systems and their functioning

S.NO	Sensor	Uses
1.	Ultrasonic	It is used to measure the distance from the immediate obstacle by sending an ultrasonic pulse and measuring the time taken by the pulse to return back
2.	Pressure	It is used to measure the altitude indirectly using the inverse relation between the pressure and altitude as the ultrasonic waves cannot travel larger distances
3.	Inertial Measurement Unit	This consists of a gyroscope and an accelerometer. The three axis gyroscope measures the orientation of the drone and the accelerometer measures the acceleration.
4.	Image (Camera)	The onboard camera is used to perform image processing and enable autonomous navigational capabilities. Here the camera uses a technique called as optical flow to estimate the horizontal velocity of the drone by comparing two images taken at an specified intervals

Fig 3.2: Sensors used in the Flight Control System model

3.2 Flight Control System

The Implemented Flight system consists of the data from the sensors, camera and the RF Signal from the remote controller as the input and sends the actuating signals to the motors and an emergency flag as outputs. The Flight Control System also includes an Image Processing System Block and this implements the optical flow algorithm to estimate the instantaneous velocity of the drone. This image processing block can be also scaled up to perform image processing algorithms via Convolutional Neural Networks in the future enhancements. The flight control system consists of the state estimator.

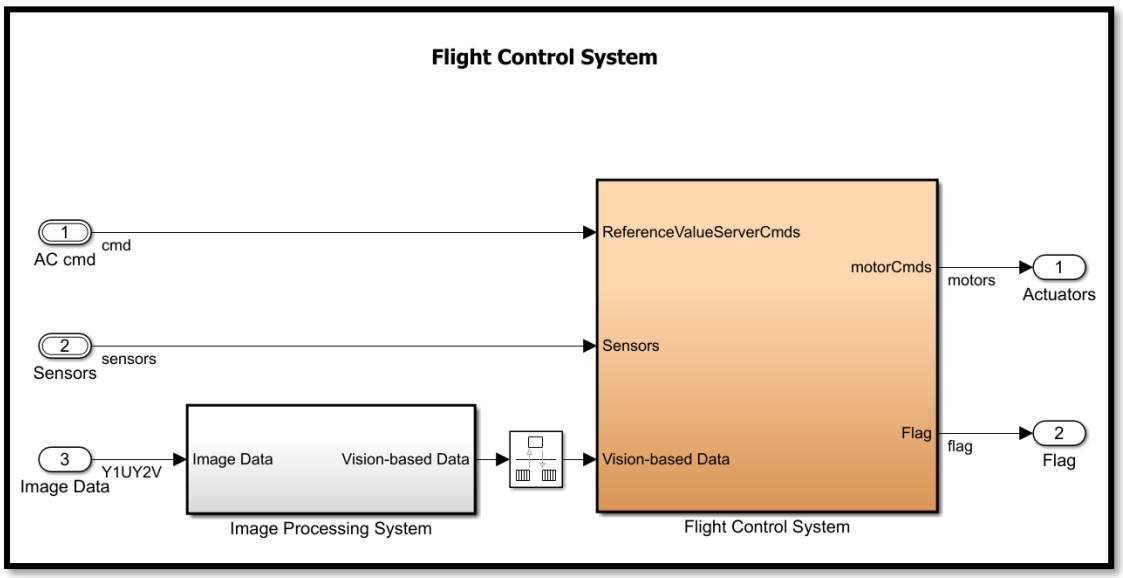


Fig 3.3: Flight Control System SIMULINK block

3.3 State Estimator

State estimator performs the core mathematical modelling using the various data obtained from the sensors. The State Estimator block is the used to interpret the angular orientation of the system from the Inertial Measurement Unit, acceleration from the accelerometer, battery level from the battery level indicator, altitude from the ultrasonic and the pressure sensors. This performs the computations of the feedback path data and applies the necessary corrections in the system thereby counter acting the external disturbances by an algorithmic use of motors. The state estimator is the vital part in the mathematical processing of the acquired information

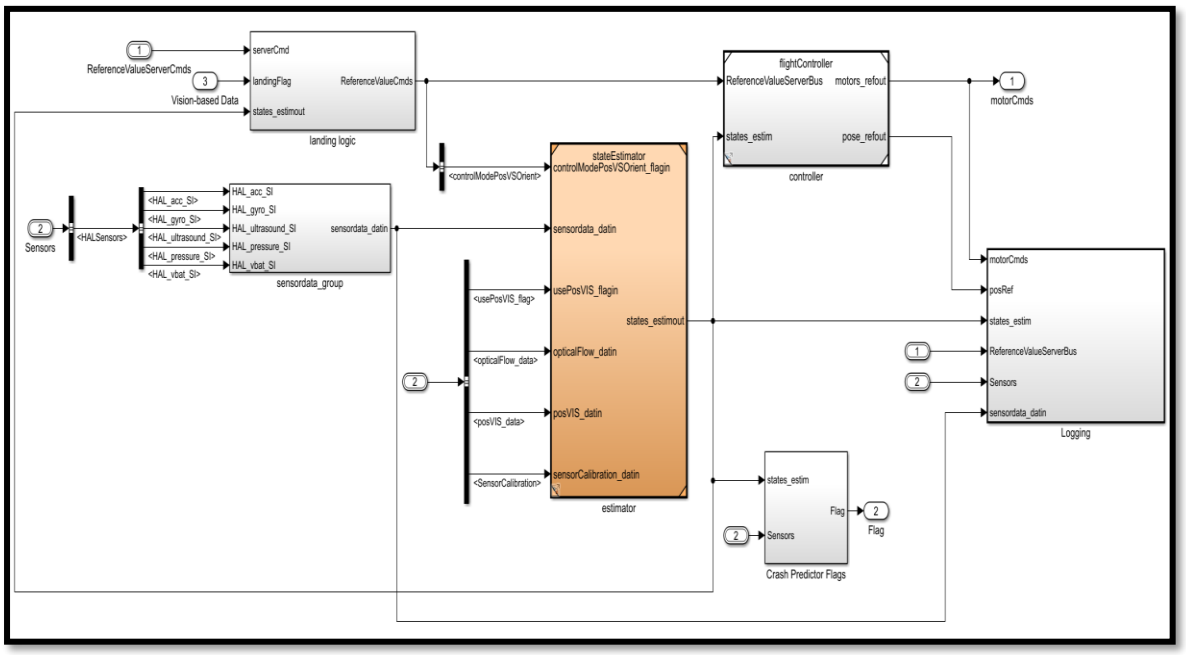


Fig 3.4: State Estimator Design in SIMULINK

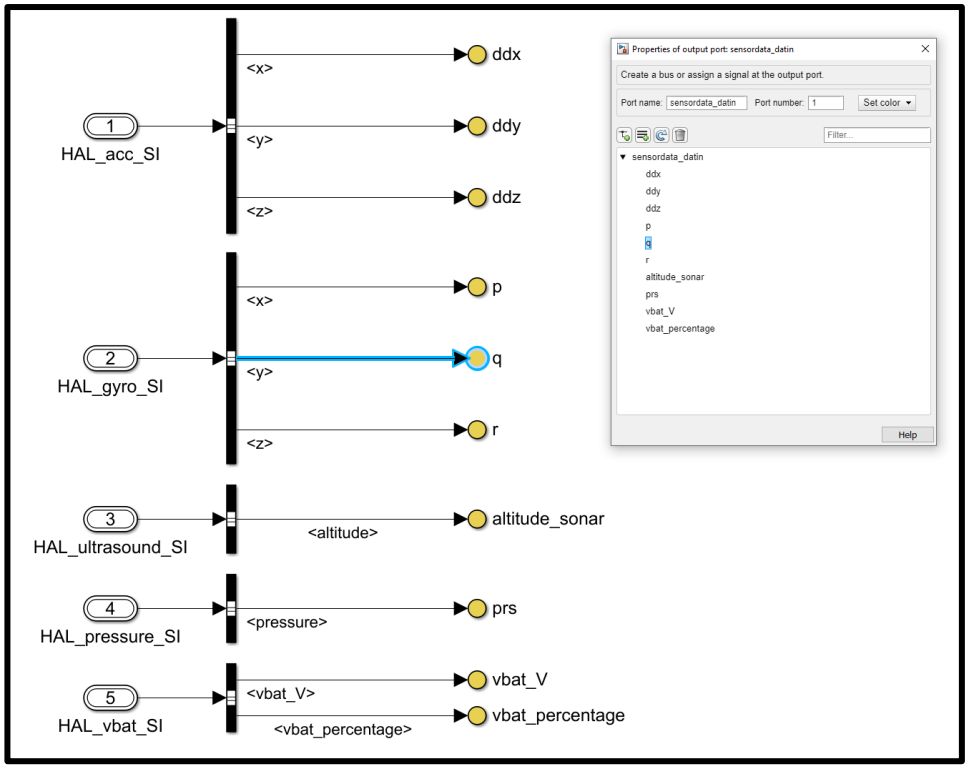


Fig 3.5: Sensor data block in SIMULINK

3.4 Thrust Control System

The thrust control system interprets the position of the drone in the drone reference frame mapping from the real world reference frame. The Position Controller also employs PID Controller to control the roll and pitch. There are six degrees of freedom composed of three translational and three rotational degrees of freedom. These six degrees of freedom are controlled with only six actuators making it an under actuated system. This requires the use of a Motor Mixing Algorithm to control only four actuators and yet achieve the desired six degrees of freedom.

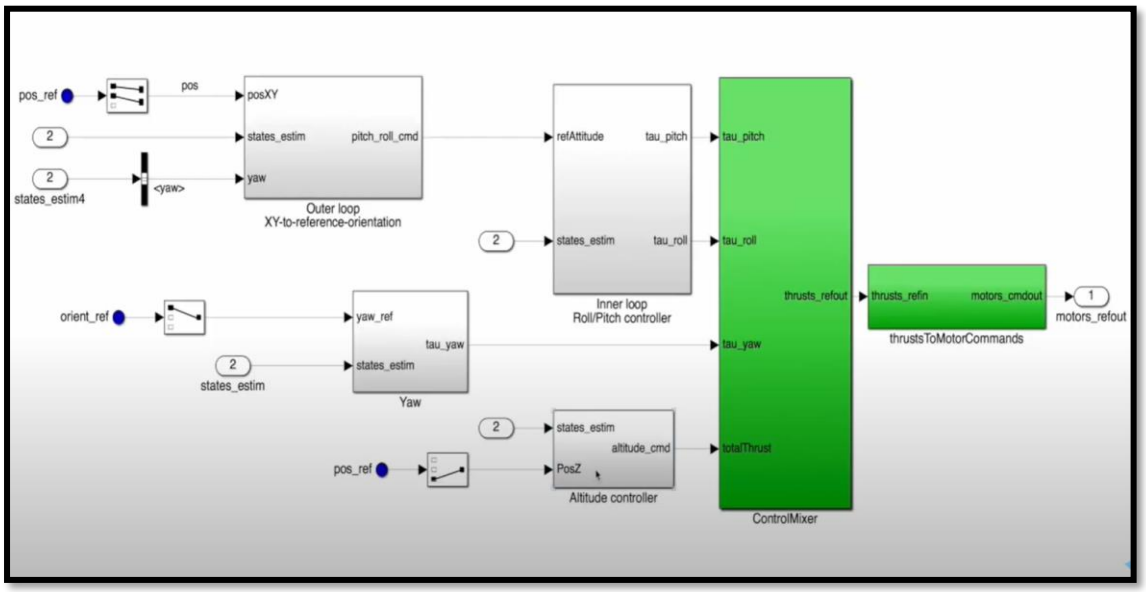


Fig 3.6: Thrust Control System Block in SIMULINK

4. Deployment of control system

4.1 Generation of Flight Code

The flight controller code is generated for three platforms

1. Parrot Mini Drone
2. APM based Flight Controller
3. Shakti C64 Vajra Microprocessor based Flight Controller

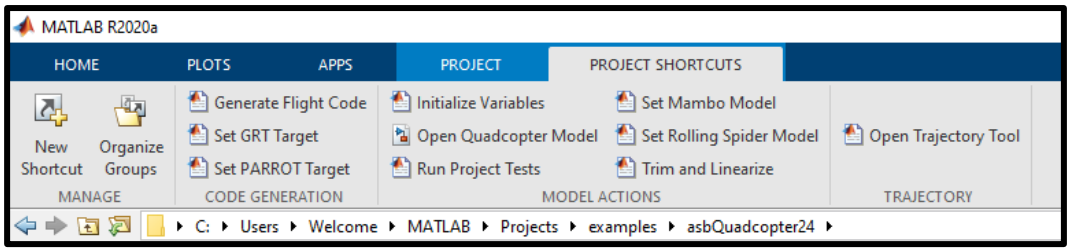


Fig. 4.1: Deployment toolbar in Quadcopter Project

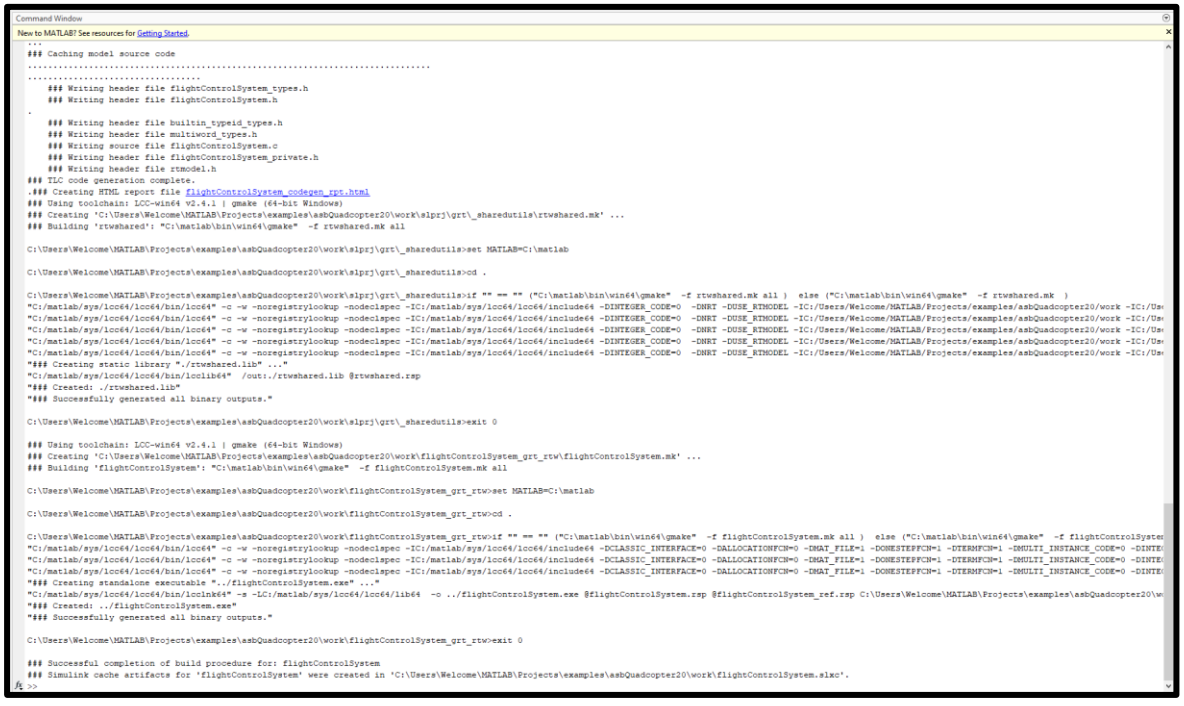


Fig 4.2: Generation of C Code from SIMULINK Coder

The generated code is then dumped and tested in Parrot Mini Drone and APM

based Flight controller. The workflow of adopting automatic code generation allows for deployment in any processor of choice, contrary to the Embedded C code which is specific for the processor

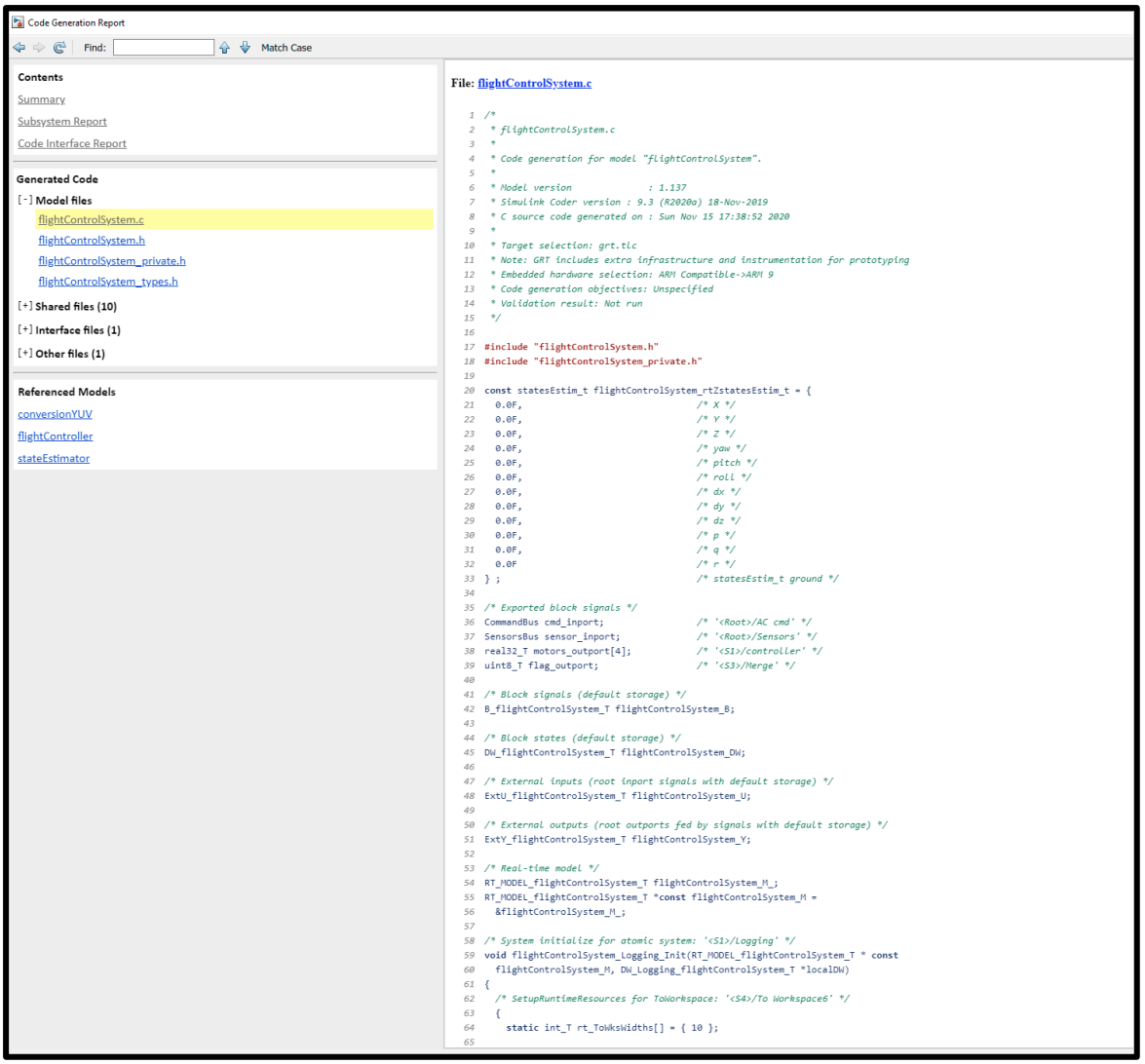


Fig 4.3: Snippet of ”flightcontroller.c” C code generated by SIMULINK Coder

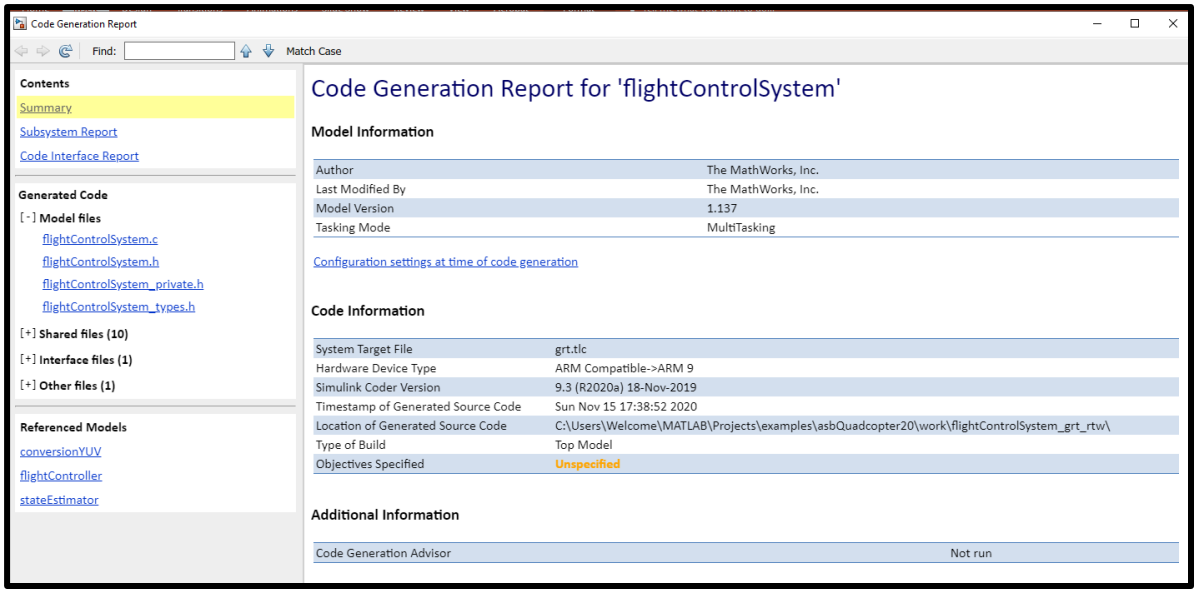


Fig 4.4: Code Generation Report

4.2 Deployment on custom processor

The developed control system can be deployed on any processor making the workflow a design once and deploy everywhere approach. This approach reduces the development time improves scalability and flexibility in adopting the control system for any processor in the future. This will also support RISC-V ISA variants as the data type size can be fixed.

The Control system though developed for RISC-V based Shakti C64 Vajra Microprocessor, the control system has been implemented on APM based drone as well as the commercially available Parrot Mini Drone

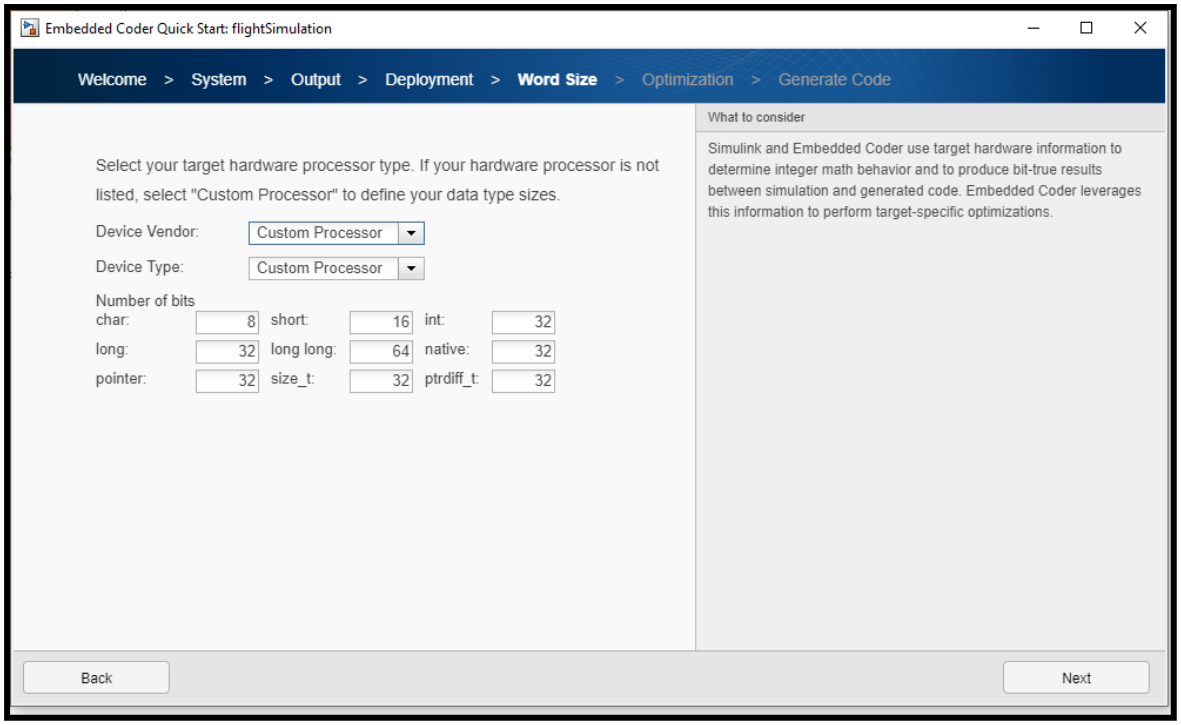


Fig 4.5: Embedded Coder - custom processor (Vajra C64) configuration

The control system code can be optimized for the performance by a suitable tradeoff between the RAM allocation and the time for instruction execution. This makes the code less RAM intensive as the Shakti Vajra C64 ported in Arty 7-100T comes with only 256 MB of RAM. This also proves that a clock frequency of more than just 50 MHz is enough to suffice the application without much troubles

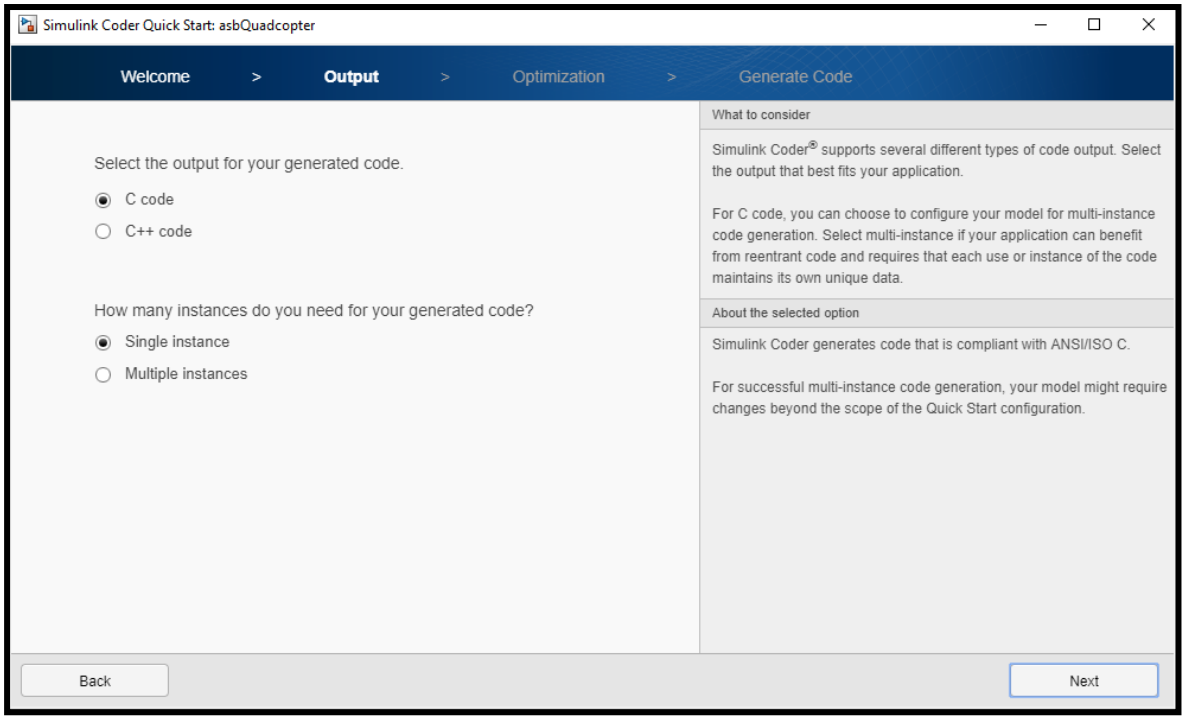


Fig 4.6: Simulink Coder for Quadcopter model

The workflow of deploying the drone with control system developed as SIMULINK blocks also allows for the possibility of a tight integration with ASIC level design via automatically generated optimized HDL Code from the MATlab toolset.

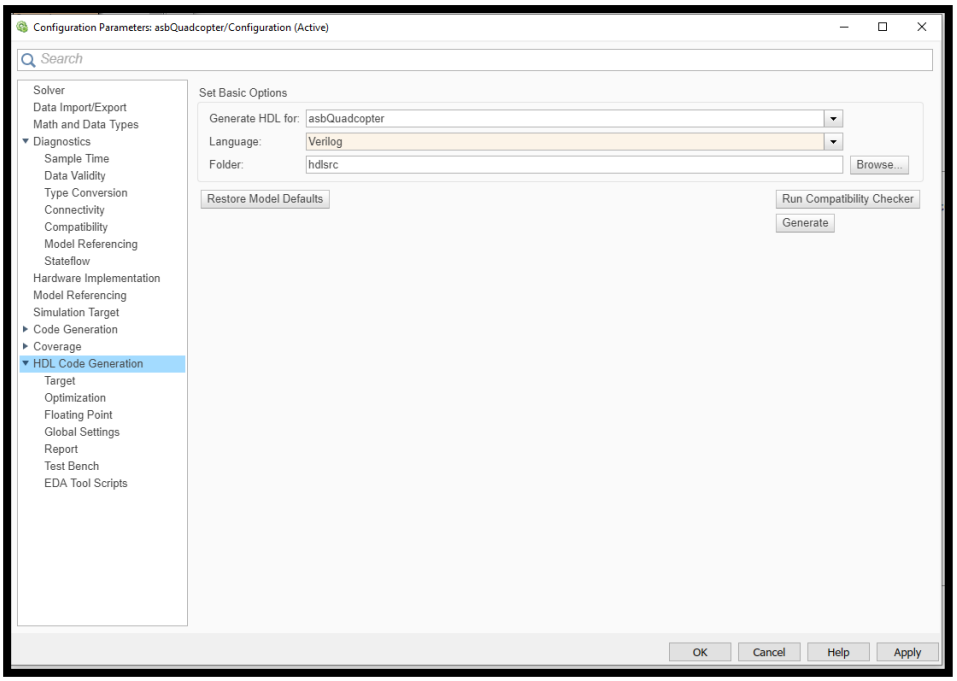


Fig 4.7: HDL coder for the SIMULINK Quadcopter model

5. Simulation and flight plan

The deployed control system is simulated using the Aerospace Block set simulator and Parrot mini Add-On in MATlab. The drone was able to hover at the start of the simulation and was able to maneuver the planned trajectory. The trajectory test succeeded as in *Fig. 5.5*

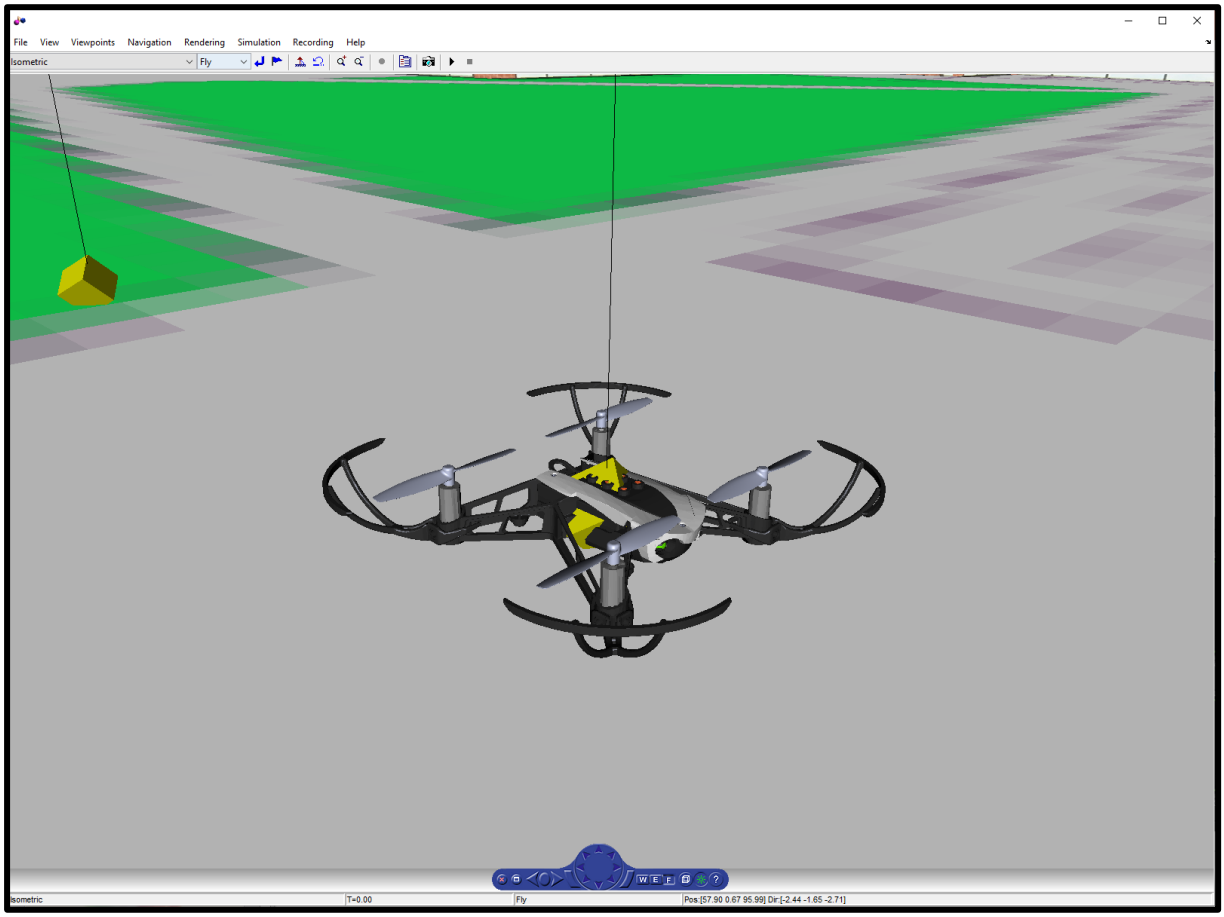


Fig 5.1: Quadcopter SIMULATION using custom developed Control System

The instrument cluster is also designed to output the different flight parameters and telemetry data namely

1. Altitude
2. Airspeed
3. Virtual Heads On Display
4. RPM of the four rotors
5. Magnetometer
6. Vertical Speed

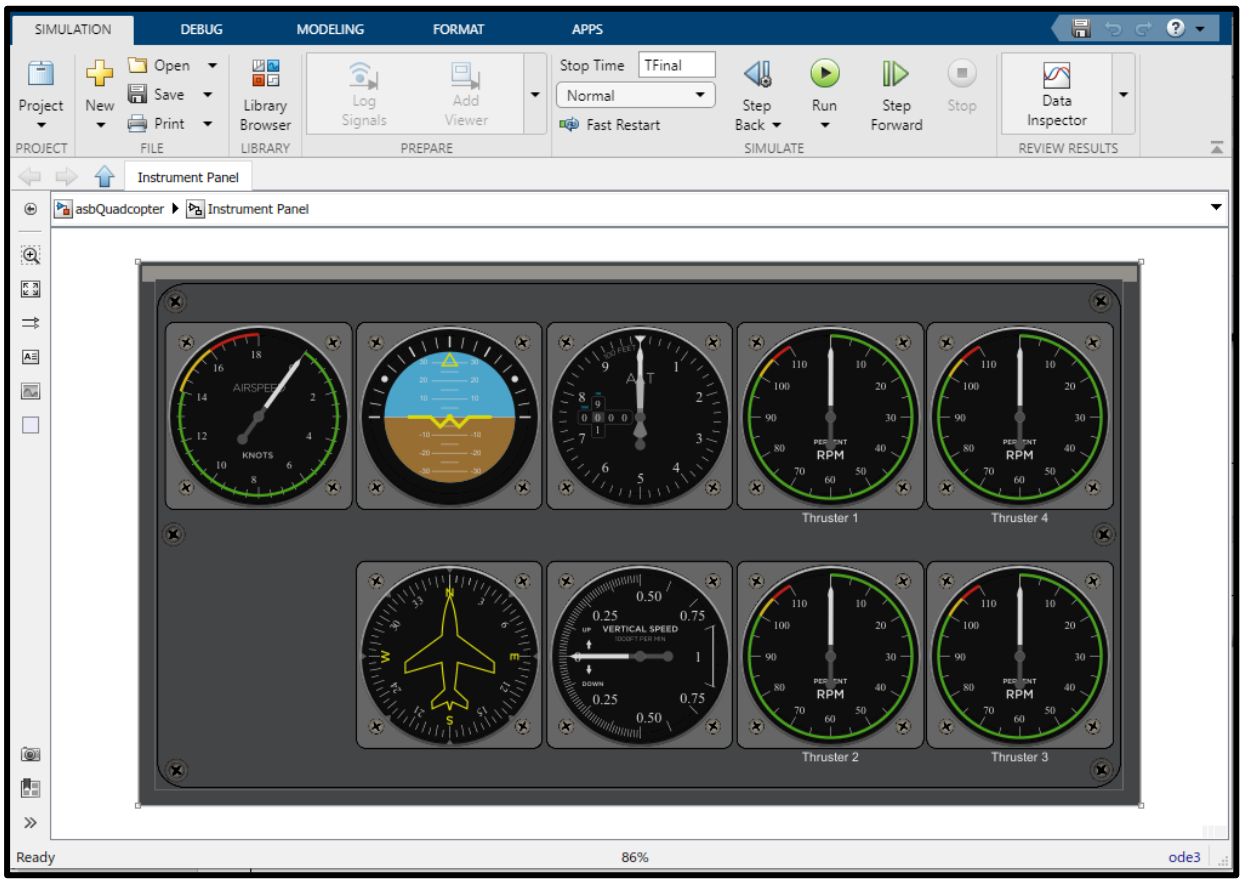


Fig 5.2: Developed Instrument Cluster for the quadcopter at the start of the simulation

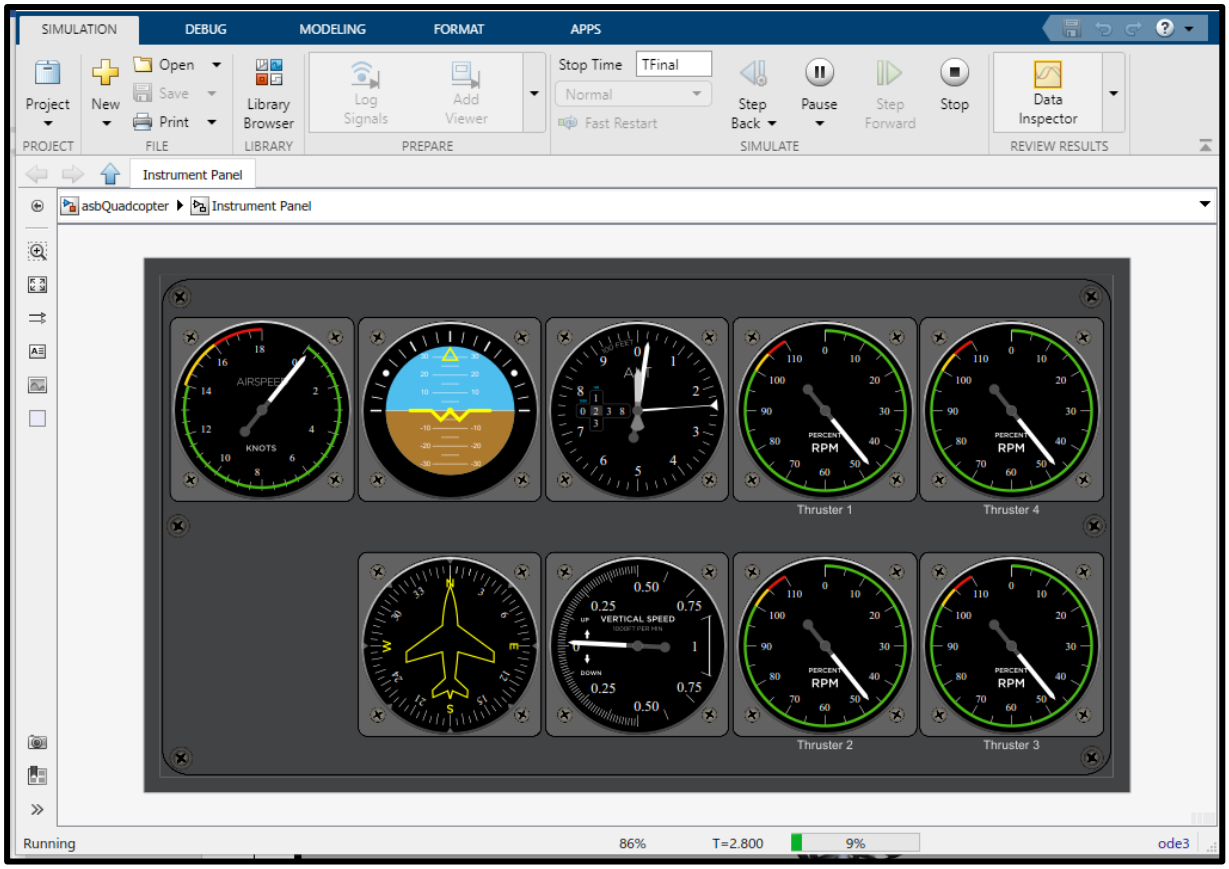


Fig 5.3: Developed Instrument Cluster for the quadcopter during the simulation

The instrument cluster before the start of the simulation and during the run time of the simulation is depicted as in the Fig.5.2 and 5.3. The simulation model was also improved using the Performance advisor option in SIMULINK

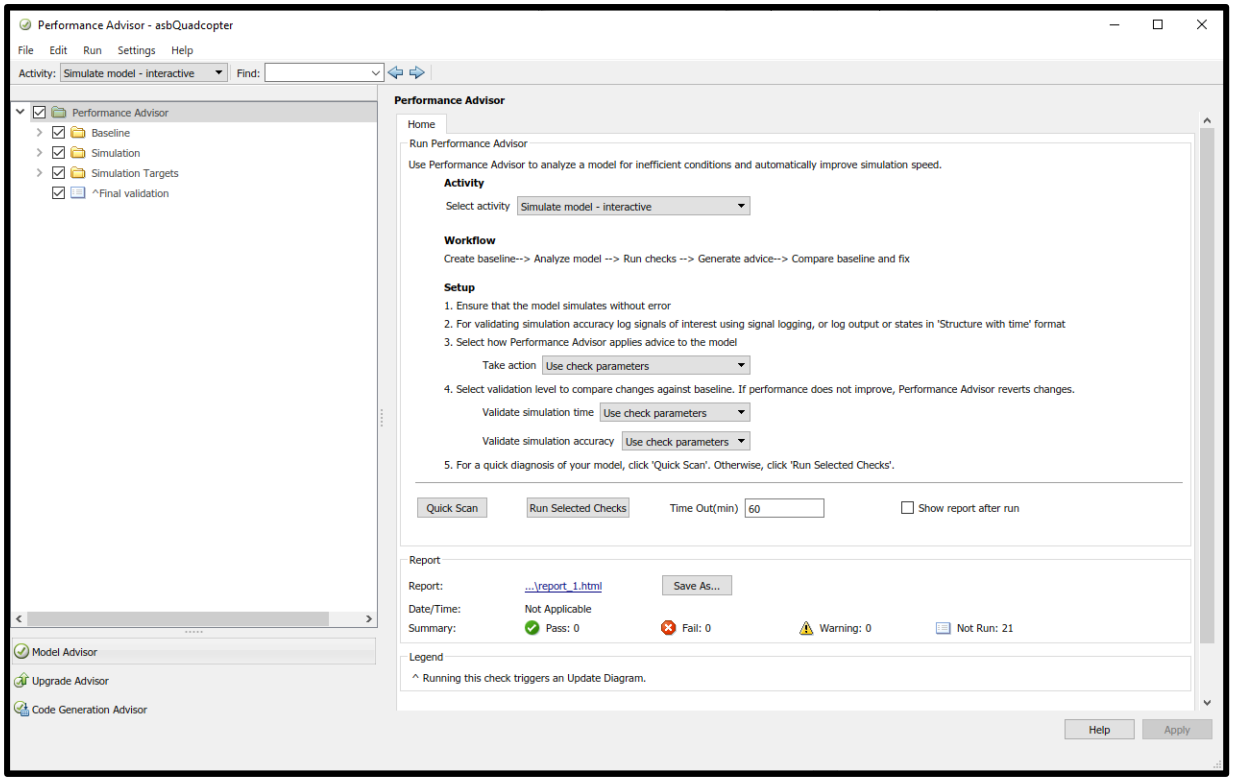


Fig 5.4: Performance Optimizer for the SIMULATION

```
### Successful completion of build procedure for: flightControlSystem
### Simulink cache artifacts for 'flightControlSystem' were created in 'C:\Users\Welcome\MATLAB\Projects\examples\asbQuadcopter20\work\flightControlSystem.slxc'.
Running trajectoryTest
.
Done trajectoryTest
__________


ans =

  TestResult with properties:

           Name: 'trajectoryTest/testTrajectoryGeneration'
         Passed: 1
         Failed: 0
     Incomplete: 0
       Duration: 0.5761
        Details: [1×1 struct]

Totals:
   1 Passed, 0 Failed, 0 Incomplete.
   0.57615 seconds testing time.
```

Fig 5.5: Trajectory test result

6. Hardware deployment

In order to dump the code in Vajra C64 processor the .mcs file and bit stream from the GitLab Repository of RISE LAB is cloned and uploaded into the Arty 7-100T FPGA. The PlatformIO IDE detects the board and the generated C code with suitable modifications according to the PINMUX mapping of Vajra C64 is dumped

The firmware for the peripherals are written. At first I2C is written and the codes are used in PlatformIO IDE to debug and evaluate performance using spiking. The design is verified within the IDE.

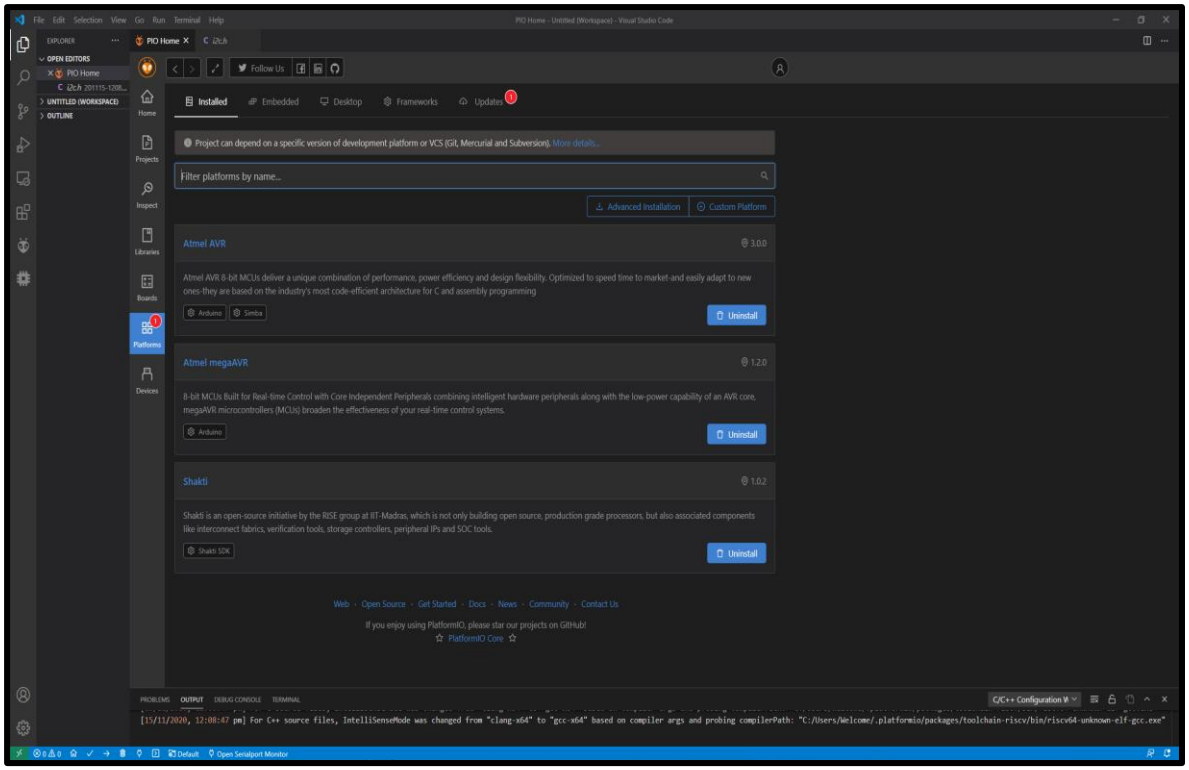


Fig 6.1: PlatformIO IDE with Shakti SDK Ported

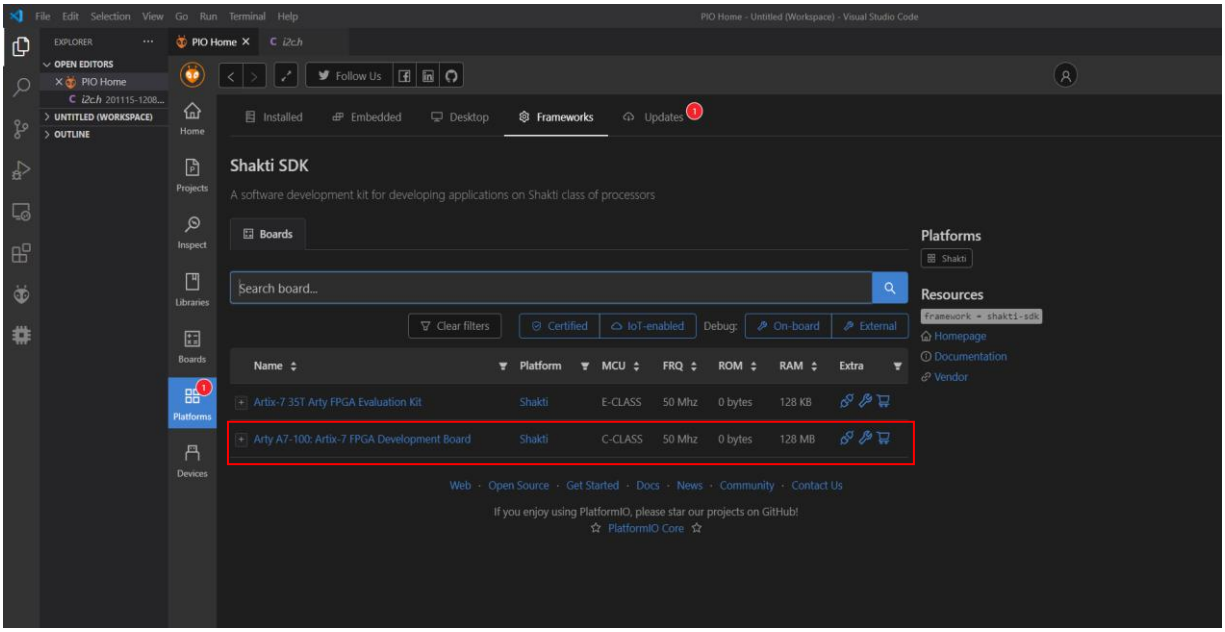


Fig 6.2: Arty 7-100T FPGA Board Module configuration

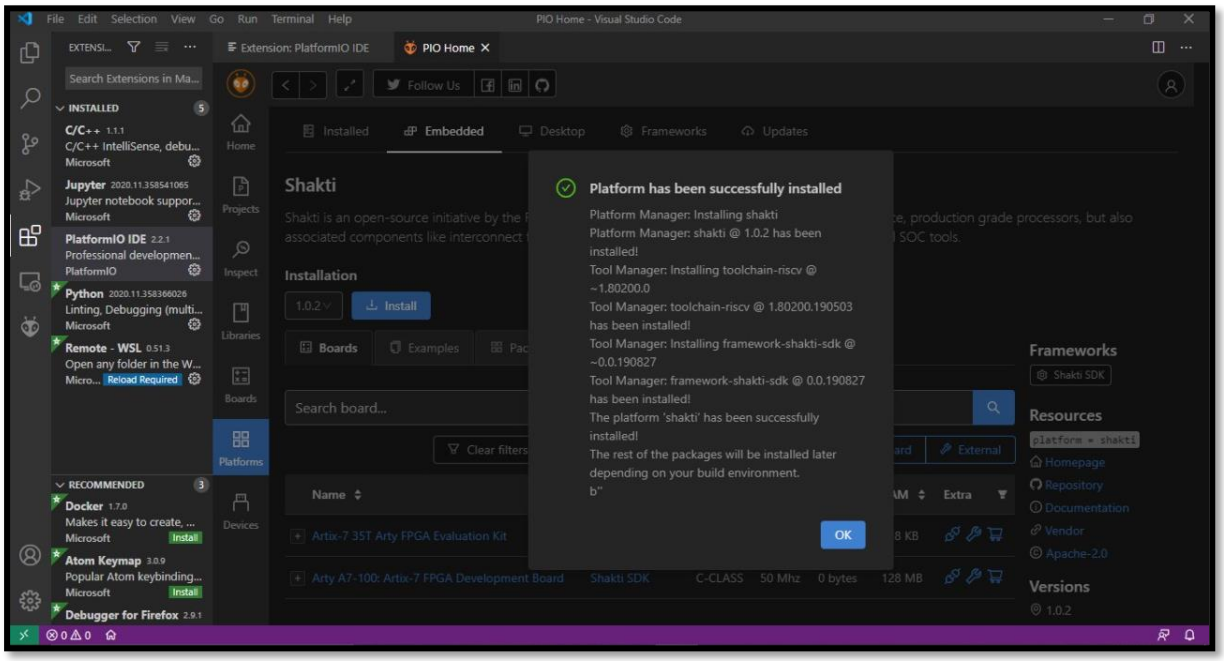


Fig 6.3: Successful porting of Shakti SDK in PlatformIO and configuration of Arty-7 100T FPGA

The firmware for the different peripherals are added in the lib folder and the generated source codes are added to the src folder. The make file is automatically generated and uploaded via UART. The same control system is also deployed in Parrot Mini Drone and APM based drone and the control system is found to be stable and the test was a success

Fig 6.4: Deployment into Custom Built APM Based drone with autonomous navigation

Fig 6.5: Test Flight of Custom Built APM Based drone with autonomous navigation

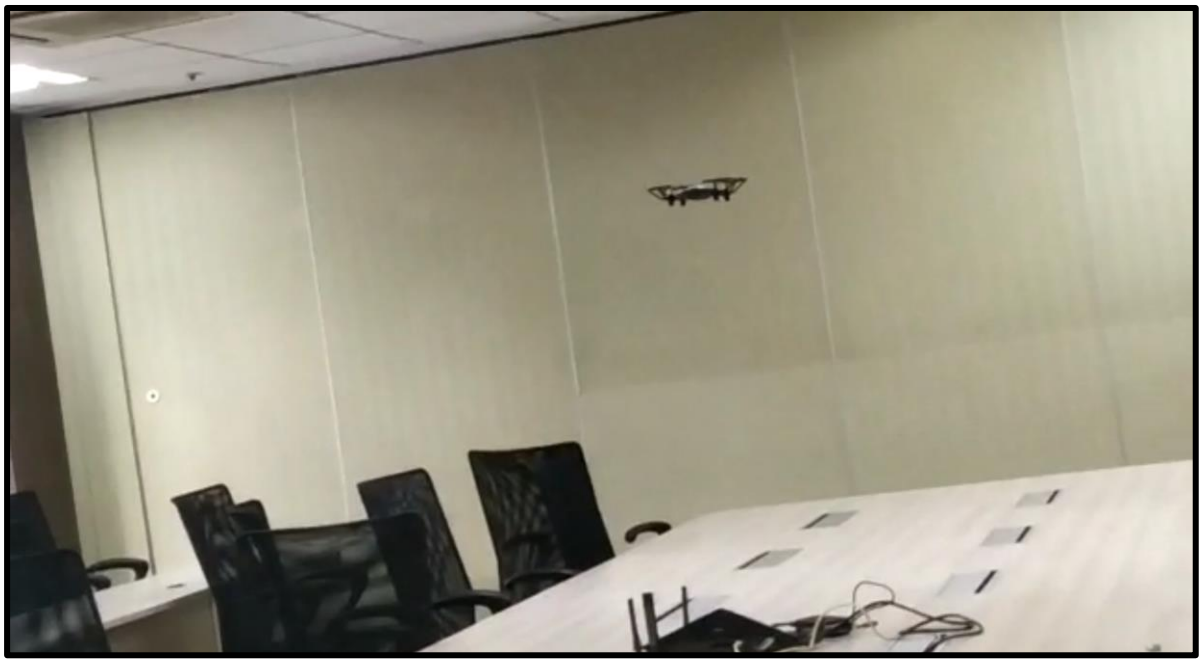


Fig 6.6: Test Flight of Parrot Mini drone with control system implemented in SIMULINK

7. Conclusion

The indigenously developed flight controller will be suitable for deployment into military and defence applications providing a secure and backdoor exploit less Controller for Mission Critical Applications. The developed control system and the proposed fight controller would prove to be an effective solution against common security flaws in proprietary processors like backdoor exploits. The indigenously developed flight controller emphasizes the “Make in India” campaign and the “Aatmanirbar Bharat” initiative.

8. Startup and Future Scope

The indigenously developed flight controller may be ported to fault tolerant version of Shakti, F- Class Processors. This can then be deployed in defense and other mission critical applications. The use of C class processor also provides the ability to implement on board image processing and machine learning algorithms so that autonomous capabilities including SLAM could be added.

This idea is implemented under startup, Technowiz Innovations, founded by Shrihari in 2018. The incubation for this project is applied at renowned incubators in Coimbatore.

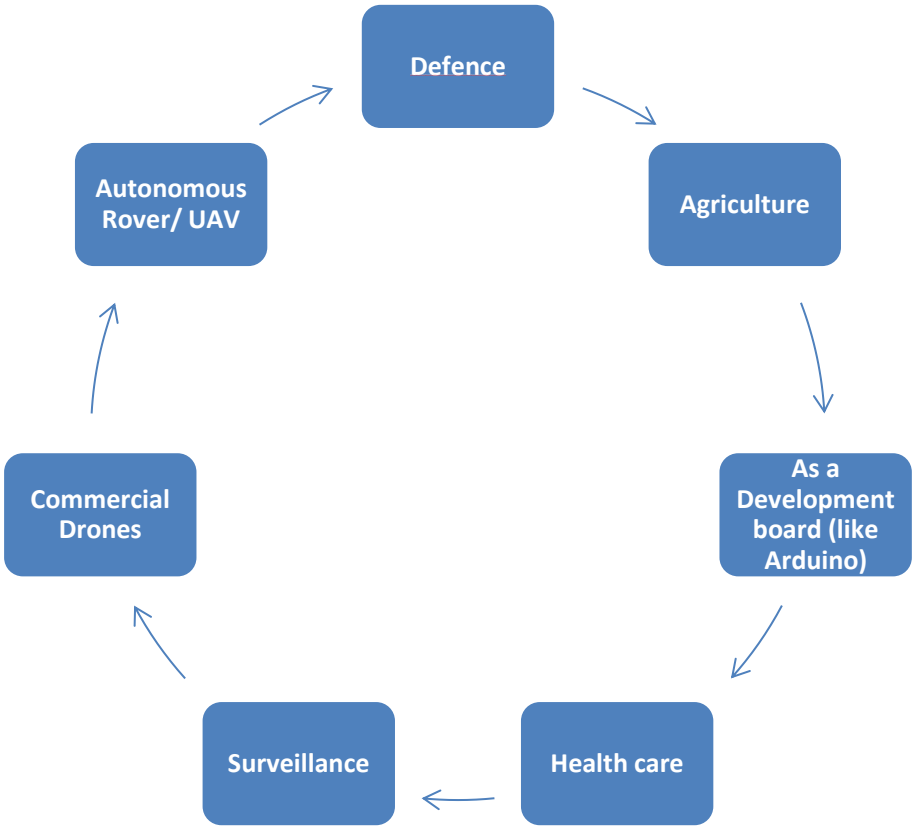


Fig 8.1: Thrust areas for Shakti based Flight Controllers

Appendix I

Memory mapping

Sl. No	Peripheral	Base Address Start	Base Address End
1.	Memory (DDR)	0x80000000	0x87FFFFFF
2.	Debug	0x00000000	0x0000000F
3.	UART0	0x00011300	0x00011340
5.	I2C	0x020C0000	0x020C00FF
4.	GPIO	0x020D0000	0x020D00FF
6.	CLINT	0x02000000	0x020BFFFF
7.	PLIC	0x0C000000	0x0C020000

Appendix II

Device Pin Mapping

Sl. No	Pin Description	Pin mapping	Peripheral
1.1	GPIO0	CKIO0 (J4[1],IO - Lower)	GPIO
1.2	GPIO1	CKIO1 (J4[3],IO - Lower)	
1.3	GPIO2	CIIO2 (J4[5],IO - Lower)	
1.4	GPIO3	CKIO3 (J4[7],IO - Lower)	
1.5	GPIO4	CKIO4 (J4[9],IO - Lower)	
1.6	GPIO5	CKIO5 (J4[11],IO - Lower)	

Sl. No	Pin Description	Pin mapping	Peripheral
1.7	GPIO6	CKIO6 (J4[13],IO - Lower)	
1.8	GPIO7	CKIO7 (J4[15],IO - Lower)	
1.9	GPIO8	CKIO8 (J2[1],IO - Higher)	
1.10	GPIO9	CKIO9 (J2[3],IO - Higher)	
1.11	GPIO10	CKIO10 (J2[5],IO - Higher)	
1.12	GPIO11	CKIO11 (J2[7],IO - Higher)	
1.13	GPIO12	CKIO12 (J2[9],IO - Higher)	
1.14	GPIO13	CKIO13 (J2[11],IO - Higher)	
1.15	GPIO14	CKIO26 (J4[2],IO - Lower)	
1.16	GPIO15	CKIO27 (J4[4],IO - Lower)	
2.1	SDA	CK_SDA (J3[1])	I2C
2.2	SCL	CK_SCL (J3[2])	
3.1	UART0 TX	J10	UART
3.2	UART0 RX	J10	

Sl. No	Pin Description	Pin mapping	Peripheral
4.1	INTERRUPT 0	CKIO28 (J4[6],IO - Lower)	PLIC
4.2	INTERRUPT 1	CKIO29 (J4[8],IO - Lower)	
4.3	INTERRUPT 2	CKIO30 (J4[10],IO - Lower)	
4.4	INTERRUPT 3	CKIO31 (J4[12],IO - Lower)	
4.5	INTERRUPT 4	CKIO32 (J4[14],IO - Lower)	
4.6	INTERRUPT 5	CKIO33 (J4[16],IO - Lower)	
4.7	INTERRUPT 6	CKIO34 (J2[2],IO - Lower)	
4.8	INTERRUPT 7	CKIO35 (J2[4],IO - Lower)	

APPENDIX III

PLATFORM IO – USAGE (EXTRACTED FROM SHAKTHI USER MANUAL)

PlatformIO IDE is the next-generation integrated development environment for IoT.

3.1 PREREQUISITES

• **Operating System**:

– Linux: Ubuntu 16.04 or later

– Windows: Windows 10

• **Python Interpreter**: Python 2.7 or Python 3.5+.

• **Access to Serial Ports (USB/UART)**:

– Windows Users: Please check that you have correctly installed USB driver from board manufacturer

– Linux Users: Please install 99-platformio-udev.rules

• **Software**: Visual Studio Code 2018-2019

• **FPGA Board**: Pio currently supports the below boards. 12

– Artix 7 100t

– Artix 7 35t

3.2 INSTALLATION

• Getting PlatformIO for Visual Studio

– Download the Visual Studio from the following link:

https://code.visualstudio.com/?wt.mc_id=DX_841432

– Once you have downloaded the visual studio, go to the extensions tab, in the search bar search for PlatformIO IDE and install it.

– Restart the visual studio once you have installed the PlatformIO extension.

– Open the platforms tab and install the platform IO framework using the following two methods:

* Use the Advanced Installation by Clicking the Advanced Installation Button

-> Paste the link https://github.com/platformio/platform-shakti to

install the corresponding platform.

* Use the Command Line for installing the respective platform by typing the following in the terminal pio platform install https://github.com/platformio/platformshakti

– Once the above steps are complete restart the visual studio to refresh the IDE.

3.3 USING SHAKTI SDK IN PLATFORMIO IDE

The shakti software development kit is a ready to use kit where one can develop software's and projects on the respective boards available. The SHAKTI-SDK is embedded into the PlatformIO IDE, along with RISC-V tools. There by the application development process is integrated into the PlatformIO IDE.

3.3.1 GETTING STARTED WITH YOUR FIRST APPLICATION

To have a clear understanding of how the software is built and deployed one can go through a quick example by selecting Project Examples and follow the following steps:

• Select the hello application from the drop down and press import.

• Once the project is imported, you can start building the project by going to project tasks (Alien icon on left side)->Build

• To connect the board with IDE, follow the instructions

• Once you have connected the board you can proceed by Clicking on Upload and Monitor on the Project task column, which is followed by a pop-up terminal displaying the monitor rate also it uploads the program automatically.

• To execute the corresponding program to the board, go to Debug ->Start Debugging or press F5. This should insert a break point automatically and execute.

• The above two steps should upload the program and open the terminal displaying the output "HelloWorld"

3.3.2 CREATING A PROJECT FROM SCRATCH

1. Start creating a project by doing the following

• New Project -> Enter Details in the boxes as below

– Project Name: Give any name of your choice

– Board: Choose appropriate board of your choice from drop down as below

* Arty7 100t

* Arty7 35t

– Framework: By default, it will choose SHAKTI-SDK, if not select SHAKTISDK from drop box

• Click Finish

2. Creating your first application, when creating your first application one should know where to put the files, hence please refer the below structure.

• Once you have created appropriate files according to the above steps, Build the project and acquire self-output file so that you can upload to the board and test.

• Use the "Upload and Monitor " Command copy the elf to the board and execute.

APPENDIX IV

SHAKTI SDK - USAGE (EXTRACTED FROM SHAKTHI USER MANUAL)

SHAKTI-SDK ARCHITECTURE

The SHAKTI-SDK is a C/C++ platform to develop applications over SHAKTI. The SDK has the necessary firmware code and framework to develop newer applications on the hardware. The framework is light weight and customizable.

SHAKTI-SDK

DOCS BSP SHAKTI-TOOLS SOFTWARE

DRIVER

LIBS

CORE

INCLUDE

UTILS

BOARDS

EXAMPLES

BENCHMARK

PROJECTS

4.1.1 BOARD SUPPORT PACKAGE

The BSP consists of driver files for various devices and system files. It contains certain platform dependent definitions for each board. Essentially, the BSP is the layer above the hardware. It includes the following sub directories:

1. drivers

The drivers are a set of software constructs that help software applications to access the devices in the SoC. They are generally low-level API's, that execute a particular task in the hardware.

DRIVERS

PLIC GPIO

I2C

SPI

QSPI

UART

CLINT

PWM

2. include

This folder has header files for core and each driver. The board independent variable/macro definitions and declarations pertaining to each driver is included here.

3. libs

The library utilities, boot code is hosted here. Library is a common place for reusable code. The libraries can be compiled as a separate "lib" file and used.

4. core

The core usually has functions related to the startup codes, Trap handlers and interrupt vectors. The codes related to memory initialisation are also available here.

5. utils

This contains the code related to standalone mode feature of the shakti processor.

6. thirdparty

This folder provides support for external boards as well as custom boards. Includes the definitions of board specific functions such as console drivers.

4.1.2 SHAKTI-TOOLS

SHAKTI is a RISC-V based architecture. It uses the RISC-V toolchain to develop software. The shakti-tools folder has "ready to use" RISC-V tools. It has a RISC-V GNU tool chain, RISC-V instruction set simulator, OpenOCD (debugger) and RISC-V spike simulator. These tools can be downloaded, along with the SHAKTI-SDK.

4.1.3 SOFTWARE

The software folder provides a platform for developing various applications,

independent

of the underlying BSP. All the applications/projects developed in SHAKTI-SDK reside in this folder. In general, an application will involve writing high level C/C++ code that uses BSP API's. The software folder is broadly classified in to three sub-directories:

1. Projects

This folder consists of applications developed using different sensors. These are usually a combination of standalone applications.

2. Benchmarking

Applications or bare metal codes that are developed for bench-marking a core reside here. These programs usually describe the capability of the SHAKTI class of processors.

3. Examples

This is the place where any new standalone application is developed. Few example programs involving sensors are already developed for different peripherals and kept here. These programs demonstrate the integration of BSP and the core support libraries with the user programs.

4.1.4 MAKEFILE

To compile programs more efficiently the GNU's MAKE utility is used. The make utility uses the *Makefile* to compile program from source code. The output generated by the MAKE utility is in ELF format. The Makefile has support for different target boards and applications. The Makefile's are mostly non-recursive and devoid of complex expressions.

The supported make commands are listed below:

- **make help**

Lists the possible commands supported in Makefile.

- **make list_targets**

List the boards that are supported.

- **make list_applns**

Lists the samples that are available in SHAKTI-SDK

- **make software PROGRAM=? TARGET=?**

-PROGRAM can be found from "make list_applns"

-TARGET= artix7_35t or artix7_100t or aardonyx

-Default TARGET is artix7_35t

- **make debug PROGRAM=? TARGET=?**

-PROGRAM can be found from"make list_applns"

-TARGET= artix7_35t or artix7_100t or aardonyx

-Default TARGET is artix7_35t

-debug command adds the debug support to applications.

- **make all**

TARGET= artix7_35t

All the applications under example folder are compiled for above target.

- **make clean**

cleans all the executable and the design overrides the executable generated by the last target with current target.

- **make clean CLEAR=?**

CLEAR?= any application under list_applns

clean the executable for an application.

4.2 SETTING UP THE SHAKTI-SDK

4.2.1 PRE-REQUISITES

Ensure that the following packages are installed in the host system. To solve the software

dependencies, copy paste the below command in terminal and press enter.

```
sudo apt-get install autoconf automake autotools-dev curl make-guile
sudo apt install libmpc-dev libmpfr-dev libgmp-dev libusb-1.0-0-dev bc
sudo apt install gawk build-essential bison flex texinfo gperf libtool
sudo apt install make patchutils zlib1g-dev pkg-config libexpat-dev
```

sudo apt install libusb-0.1 libftdi1 libftdi1-2
sudo apt install libpython3.6-dev

4.2.2 DOWNLOAD THE SHAKTI-SDK REPOSITORY

SHAKTI-SDK repository contains scripts, board support packages to build your application. It can be cloned by running the following command:

git clone https://gitlab.com/shaktiproject/software/shakti-sdk.git

4.2.3 DOWNLOAD THE SHAKTI-TOOLS REPOSITORY

The SHAKTI-TOOLS repository contains both 64bit and 32bit toolchain, for building your application. It can be cloned by running the following command:

git clone --recursive https://gitlab.com/shaktiproject/software/shakti-tools.git

If you had omitted --recursive option earlier, then run the below command to clone the submodules repository:

git submodule update --init –recursive

4.2.4 SETTING UP SHAKTI TOOL-CHAIN

SHAKTI uses RISC-V tools. The tool-chain can be installed in two ways:

• Manual method

¢ Build and install toolchain from RISCV-tools.

¢ The RISCV-tools repository has the readme to install RISC-V toolchain for SHAKTISDK.

• Automatic method (Recommended)

¢ The SHAKTI-SDK repository provides the board support packagers and an environment in which the applications can be developed.

¢ The Tool-chain executables are hosted in the SHAKTI-TOOLS repository. (shakti-tools).

¢ The absolute path of the tool-chain has to be added to "PATH" variable and exported, to use it across the file system.

¢ The steps to export the tool chain to the PATH variable is provided below, Assuming, one is in SHAKTI-TOOLS repository. Copy, paste the following commands in the same terminal. This will essentially set the $PATH variable to the exact tool-chain path for that

particular session.

SHAKTITOOLS=/path/to/shakti-tools
export PATH=$PATH:$SHAKTITOOLS/bin
export PATH=$PATH:$SHAKTITOOLS/riscv64/bin
export PATH=$PATH:$SHAKTITOOLS/riscv64/riscv64-unknown-elf/bin
export PATH=$PATH:$SHAKTITOOLS/riscv32/bin
export PATH=$PATH:$SHAKTITOOLS/riscv32/riscv32-unknown-elf/bin

Things to know:

1. The availability of the tool chain across the file system is ensured by the **export** command.

2. The above commands will export both **64-bit and 32-bit tool-chains**. If only one of the tool-chains is required, it can export separately.

3. Please add the above lines in **.bashrc** file in the home folder to set the PATH **permanently** instead of that particular session.

4. The variable $SHAKTITOOLS has the location of SHAKTI-TOOLS in the file system.

5. The command **which riscv64-unknown-elf-gcc** helps you to verify whether toolchain path exported correctly.

Steps:

Automatic method

$ pwd

/home/user

$ git clone https://gitlab.com/shaktiproject/software/shakti-sdk.git

$ git clone --recursive https://gitlab.com/shaktiproject/software/shakti-tools.git

$ cd shakti-tools

$ pwd

/home/user/shakti-tools

$ SHAKTITOOLS=/path/to/shakti-tools

$ export PATH=$PATH:$SHAKTITOOLS/bin:$SHAKTITOOLS/riscv64/bin

$ export PATH=$PATH::$SHAKTITOOLS/riscv32/bin

$ export PATH=$PATH:$SHAKTITOOLS/riscv64/riscv64-unknown-elf/bin

$ export PATH=$PATH:$SHAKTITOOLS/riscv32/riscv32-unknown-elf/bin

$ which riscv64-unknown-elf-gcc

/home/user/shakti-tools/riscv64/bin/riscv64-unknown-elf-gcc

Add export commands to .bashrc, save and close the file.

4.2.5 UPDATE THE SDK OR TOOLS

To update your SDK or TOOLS repository to the latest version, move to the respective repository **(cd shakti-tools or cd shakti-sdk)** and please use the command below.

$ git pull origin master

$ git submodule update --init --recursive

This updates the required repository to its latest version

4.3 APPLICATION DEVELOPMENT

As discussed earlier, SHAKTI-SDK helps in developing applications for SHAKTI class of processors. We are listing the steps to develop a small application using SHAKTI-SDK. SHAKTI-SDK comes with a separate repository for Applications and Projects. A project usually has its own design environment, except that it imports the BSP. An application is a simple program that demonstrates the working of sensors using SHAKTI class of processors. Any program with a smaller memory footprint is put under Applications. Before developing an application, make sure that the pre-requisites mentioned below are ready.

- Board is up and running with SHAKTI.
- Download and install SHAKTI-SDK.
- Install the RISC-V tool chain.
- set PATH variable for the tool chain.

4.3.1 STEPS TO ADD A NEW APPLICATION TO SHAKTI-SDK

An application program has to use the BSP API's for any device access. The steps followed to develop a simple program is listed below

• The new application is created under one of the *examples/XYZ_applns* folder.

• XYZ should be a device type in the SoC. For example, it can be UART, I2C, etc...

• Let's assume, we are under the XYZ_applns directory.

• Create a folder for the new application and name it accordingly.

• Inside the folder, create source and header files for the application.

• Create and edit a newMakefile for the application (refer existing examples).

• The name of the application folder corresponds to the name of the application.

• Make an entry in the existing Makefile under*/shakti-sdk/software/examples* for the new application.

• Now typing *make list-applns* will list the new application as one under SHAKTISDK.

4.3.2 MY FIRST PROGRAM!

Follow the steps given below to compile and run a program to print "HelloWorld!"

• The device required is UART. Include UART device headers for the program.

• Write your program under *software/examples/uart_applns/*.

• Create a folder called 'first'.

• Create a first.c file and write a program to print "HelloWorld !".

• Create and edit aMakefile for the program and save in the 'first' folder. Use existing programs in example folder for reference.

• Make a new entry for the program in the 'existingMakefile' under examples folder.

4.3.3 BUILD

The make commands in SHAKTI-SDK gives various options to build and run a program. The list of *make* commands can be found by typing **make help** in the terminal. Once the

program is built usingMAKE command, the ELF file is generated. The ELF file is the final executable that can be loaded into the memory and run.

$ cd shakti-sdk

$ make software PROGRAM=hello TARGET=artix7_35t

Interpreting above commands:

• PROGRAMis the new one created. It is listed by typing "make list_applns".

• TARGET= artix7_35t or artix7_100t or aardonyx.

• Default TARGET is artix7_35t.

4.3.4 RUN

Once the application is built, the executable is generated in the output folder. The executable is in ELF file format and they have the extension.*shakti*. There are two modes to run an application on the arty 35t board. The two modes are discussed in the following sections.

4.4 RUNNING APPLICATION IN DEBUG MODE

After the ELF for the target application is generated, the program can be run in Debug or Standalone mode. Debug mode helps in incremental development. It also helps to understand the program flow and debug applications easily.

The Arty35T board should be connected to the OpenOCD debugger, in order to debug your program using the RISC-V GNU Debugger (GDB) software. The standard GDB commands supported by RISC-V GDB can be used to debug. Since we have built the application, already. We can start loading it to the Arty 35T board and test. The following steps list out the actions to be taken.

4.4.1 STEPS TO RUN

PREREQUISITES

1. Installminiterm

$ sudo apt-get install python-serial

2. Open three terminals, one for each of the following

a. One terminal for UART terminal display.

b. Another for GDB server.

c. And the last one for OpenOCD.

Follow the steps below to set up and run programs:

1. In the first terminal, open a serial port to display output from UART.

$ sudo miniterm.py /dev/ttyUSB0 19200. Open the terminals in the above-mentioned order.

2. In the second terminal, launchOpenOCDwith super user (sudo) permission. Please ensure that, you are in the SHAKTI-SDK folder.

For example,

$ pwd

/home/user/shakti-sdk ———> *you are in the right folder*

Press reset in the board and run the following commands.

$ cd ./bsp/third_party/artix7_35t

$ sudo $(which openocd) -f ftdi.cfg

3. In the third terminal launch RISC-V GDB. Applications will be loaded and run in this terminal

$ riscv32-unknown-elf-gdb

(gdb) set remotetimeout unlimited

(gdb) target remote localhost:3333

(gdb) file path/to/executable

(gdb) load

(gdb) c

Note:

1. "/dev/ttyUSB0" - USB means "USB serial port adapter" and the "0" ("0" or "1" or whatever) is the USB device number.

2. For C class (64 bit) applications, please use riscv64-unknown-elf-gdb instead of riscv32-unknown-elf-gdb.

3. The **default** baud rate is **19200** but if the target is **AARDONYX** the baud rate is **7668.**

4. While using **Aardonyx** as a Target the corresponding peripheral PINMUXRegisterValue has to be included in the application program. For Example, if one want to run applications of the PWM,in **Aardonyx** and want to make use of all GPIO pins as PWM pins initialize PinMux Register with 0x0002AA80.

5. The values of the Pin MUX register for different peripherals is mentioned under Aardonyx in Device mapping section of the user manual.

4.5 RUNNING APPLICATION IN STANDALONEMODE

Until now, we have been running Arty-35T board in a Debug mode. We need a Host PC to build and run the application every time. In standalone mode, the Arty-35t board when

RESET HARDWARE BOOT CORE INIT APPLICATION

DRIVER

MEM INIT REG INIT DRIVER INIT

I/O OPERATION

Execution flow, after every reset booted starts executing the code autonomously. The application is no longer downloaded from the PC through a debugger and executed. Instead, it is stored in the flash memory. When the system starts, the boot loader loads the application from the flash memory to the physical memory (RAM). Then the control transfers to the application residing in RAM. This mode of running the application is usually used for standalone systems.

4.5.1 STEPS TO GENERATE STANDALONE USER APPLICATION

The make upload command is used to build and upload the application to the flash automatically. The SHAKTI-SDK has an *uploader* tool that is used to load a content (such as ELF) to flash, after building the image.

$ cd shakti-sdk

$ make upload PROGRAM= <bare metal appln> TARGET=artix7_35t

Interpreting above commands:

• PROGRAM is the new bare metal user application that is created. It is listed by typing "make list_applns".

• TARGET= artix7_35t, refers to the board.8

APPENDIX V

Link to Project Files

Website - Details about the project

https://www.shrihari.ml/projects/

Source Code

https://drive.google.com/drive/folders/1SVnb7kaQ_TGSpaheMe8ieQj36t9Zv0z7?usp=sharing

GitLab Repository

https://gitlab.com/shariethernet/risc-v-based-flight-controller-with-shakthi-vajra-c64

APPENDIX VI

HARDWARE COMPONENT AND ITS SPECIFICATIONS

6.1 SHAKTI MICROPROCESSOR

SHAKTI is the first open-source initiative by the Reconfigurable Intelligent Systems Engineering (RISE) group at Indian Institute of Technology, Madras to develop the first indigenous industrial-grade processor.

The aim of SHAKTI initiative includes building an opensource production-grade processor, complete System on Chips (SoCs), development boards and SHAKTI based software platform. The primary focus of the team is architecture research to develop SoCs, which is competitive with commercial offerings in the market concerning area, power and performance.

All the source codes for SHAKTI are open-sourced under the Modified BSD License of the University of California, Berkeley. The project was funded by Ministry of Electronics and Information Technology, Government of India.

RISC-V is an open standard instruction set architecture (ISA) based on established reduced instruction set computer (RISC) principles. Unlike most other ISA designs, the RISC-V ISA is provided under open source licenses that do not require fees to use.

A number of companies are offering or have announced RISC-V hardware, open source operating systems with RISC-V support are available and the instruction set is supported in several popular software toolchains.

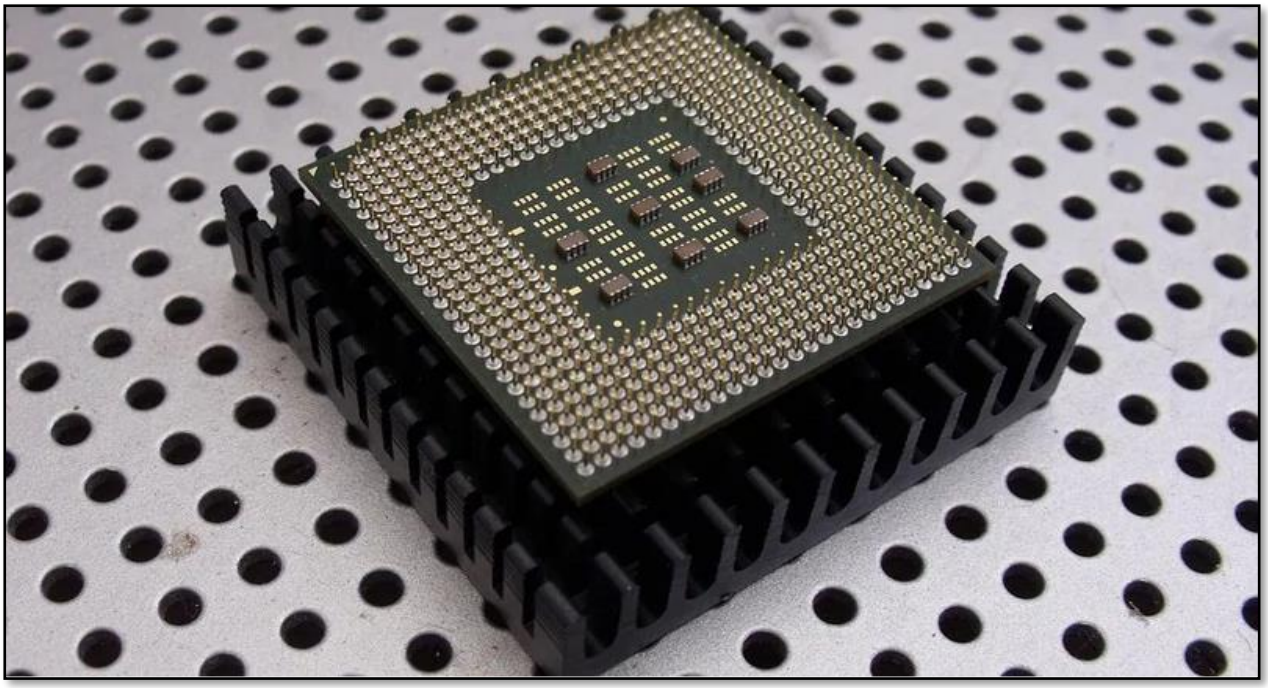
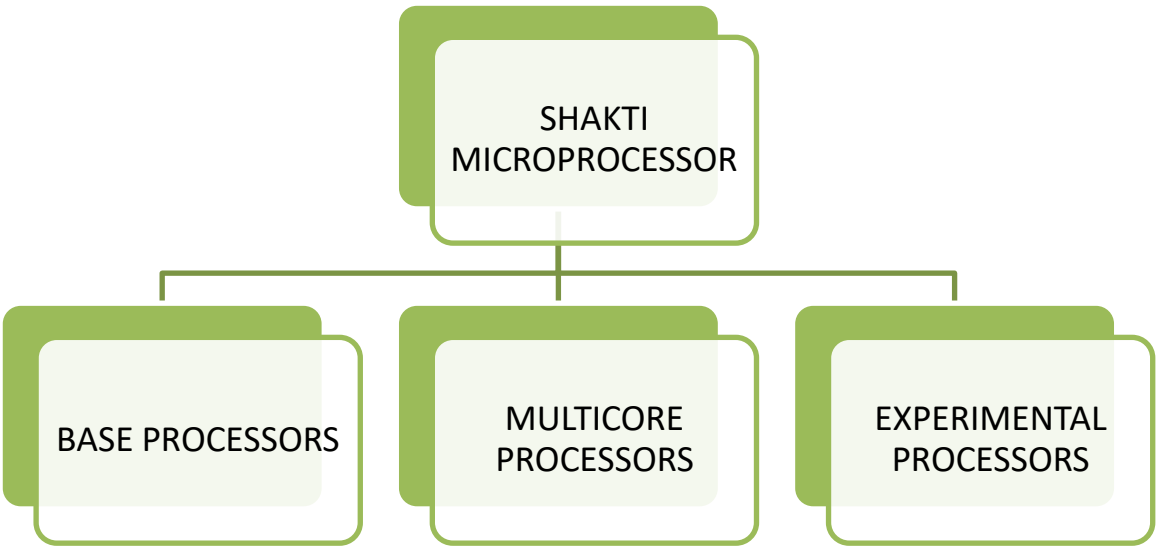


Fig 6.1: Shakti Microprocessor

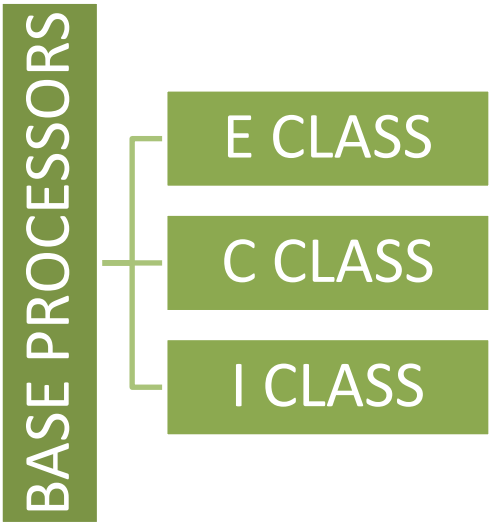
6.2 APPLICATIONS OF SHAKTI MICROPROCESSOR

1. Researchers at Indian Institute of Technology Madras (IIT-M) have designed and booted up India's first microprocessor as shown in Figure 3.1, Shakti, which could be used in mobile computing and other devices.
2. According to IIT-M, the Shakti microprocessor can be used in low-power wireless systems and networking systems besides reducing reliance on imported microprocessors in communication and defence sectors.
3. This development will assume huge significance when systems based on Shakti processors are adopted by strategic sectors such as defence, nuclear power installations, government agencies and departments.
4. According to IIT-M, Shakti processor family targets clock speeds to suit various end-user application devices such as various consumer electronic devices, mobile computing devices, embedded low-power wireless systems and networking systems, among others.

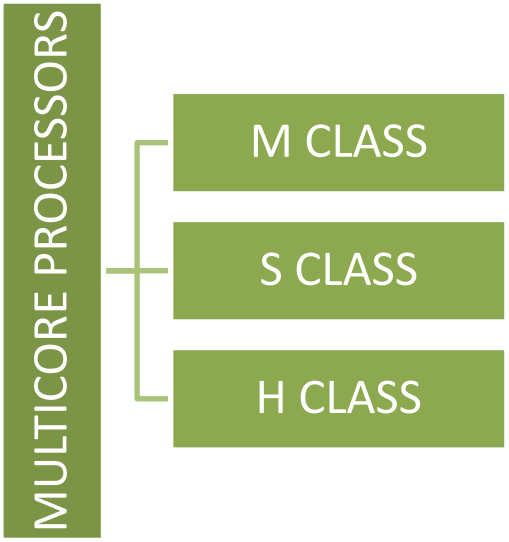
6.3 DIFFERENT CLASSES OF SHAKTI PROCESSORS



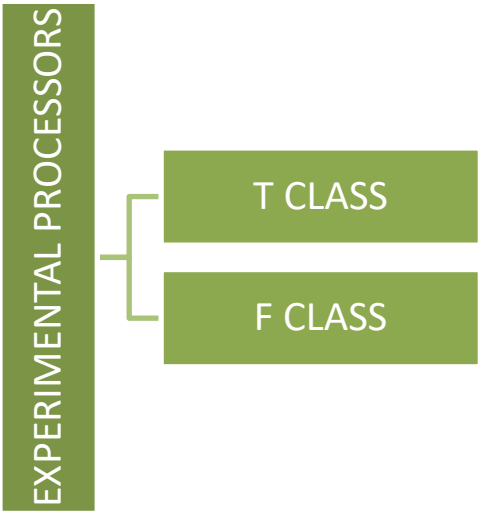
BASE PROCESSORS are further subdivided into



MULTICORE PROCESSORS are further subdivided into



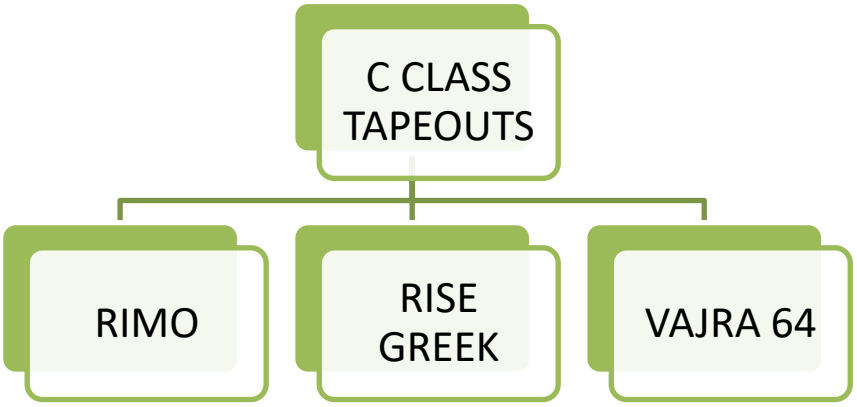
EXPERIMENTAL PROCESSORS are further subdivided into



C Class processor is chosen out of these processors because:

- C-Class is a member of the SHAKTI family of processors.
- The core generator in this repository is capable of configuring the core to generate a wide variety of design instances from the same high-level source code.
- It targets the mid-range compute systems running over 200-800 MHz.
- It can also be customized up to 2 GHz.

- It is positioned against ARM's Cortex A35/A55.
- The design instances can serve domains ranging from embedded systems, motor-control, IoT, storage, industrial applications all the way to low-cost high-performance Linux based applications such as networking, gateways etc.



6.4 PRINCIPLE OF DESIGN

There are different modules in flight controller which are to be designed individually. In order to end up with the design, the following steps are to be followed:

1. Select the required drone type
2. Choose different modules to be included
3. Design the motor control algorithm

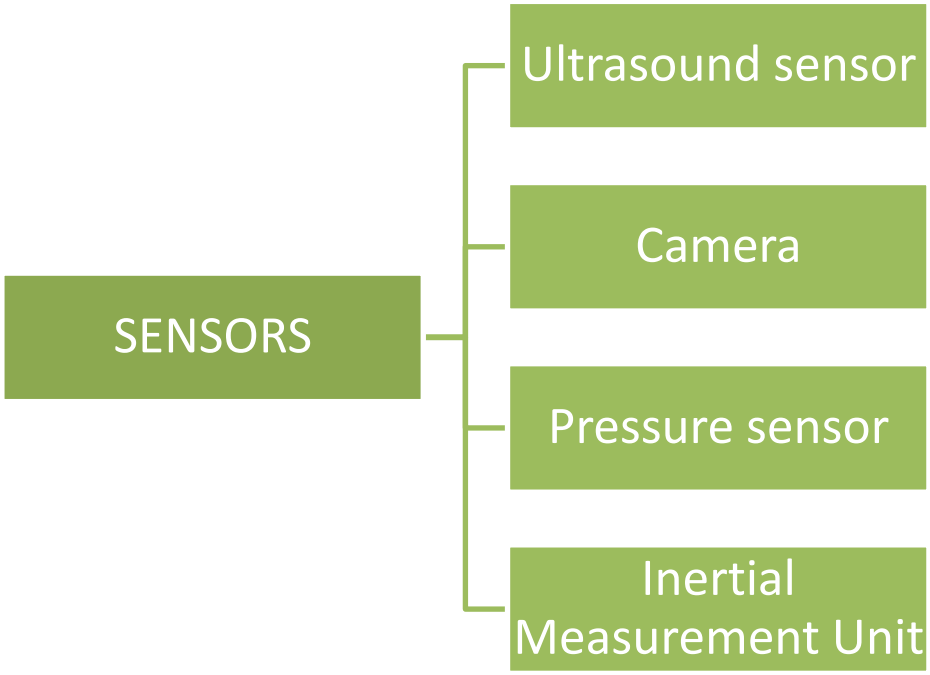
6.4.1 DRONE TYPE

Propellers spin to control the mini drone in 6 degrees of freedom. They can be divided into various types according to the number of rotors present in the drone.

6.4.2 MODULES TO BE INCLUDED

(a) SENSORS

The following are the different sensors employed in our model.



(a) ULTRASOUND SENSOR

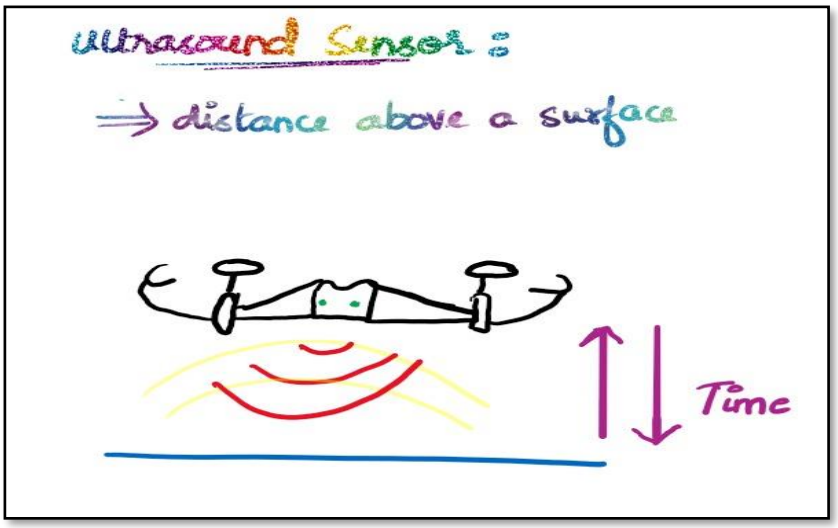


Fig 6.2: Working of Ultrasound Sensor

The Figure 6.2 depicts the working of ultrasound sensor. Ultrasound sensor could be used to measure vertical distances by sending high frequency sound pulse and measures the time it takes to bounce off the floor.

(b) CAMERA

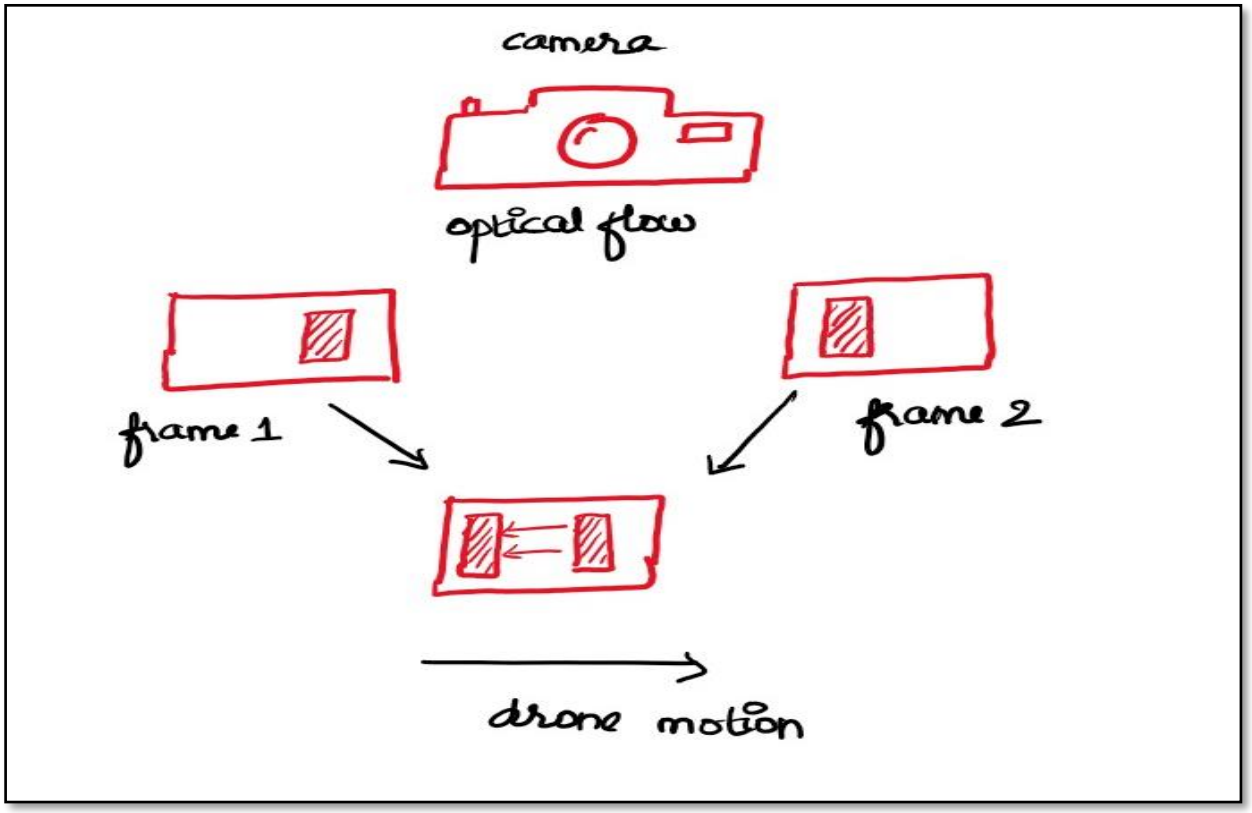


Fig 6.3: Working of Camera

The Figure 6.3 depicts the working of Camera. Camera takes 60 images per second and uses an image processing technique called optical flow to determine how objects are moving from one frame to the next. From this apparent motion, the minidrone could estimate the horizontal motion and speed.

(c) PRESSURE SENSOR

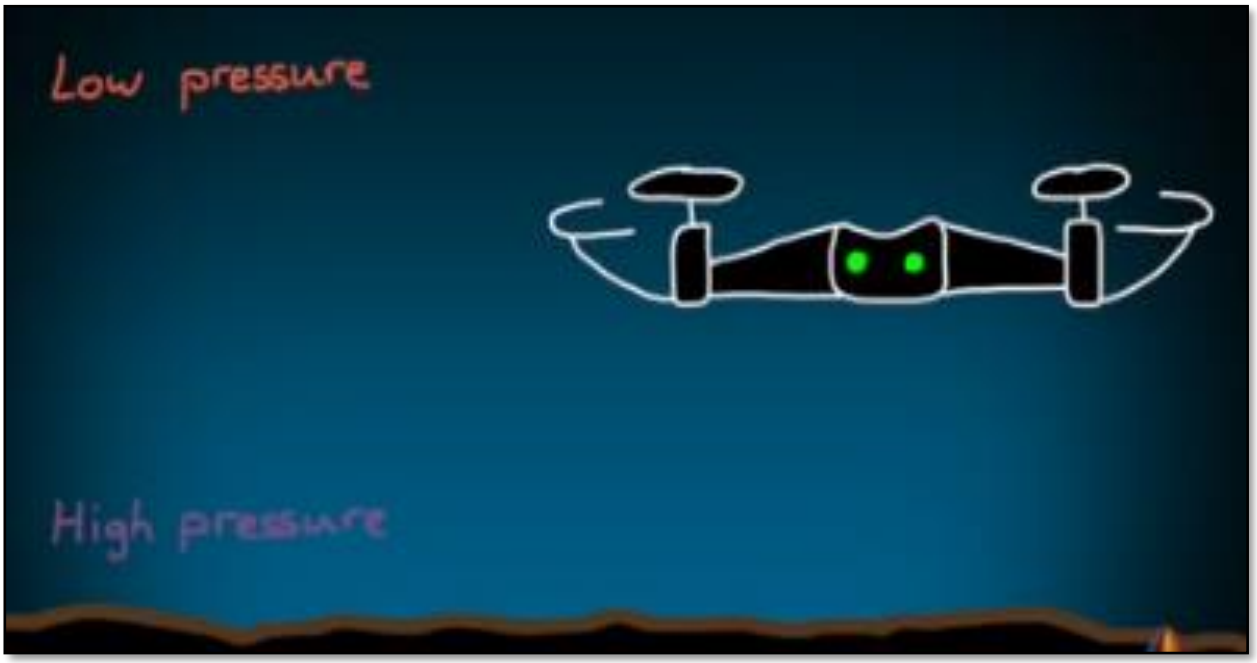


Fig 6.4: Working of Pressure Sensor

The Figure 6.4 shows the working of Pressure Sensor. Pressure sensor indirectly measures the altitude. As the drone climbs in altitude, the air pressure drops slightly. This slight change in pressure could be used to estimate the altitude.

(d) INERTIAL MEASUREMENT UNIT

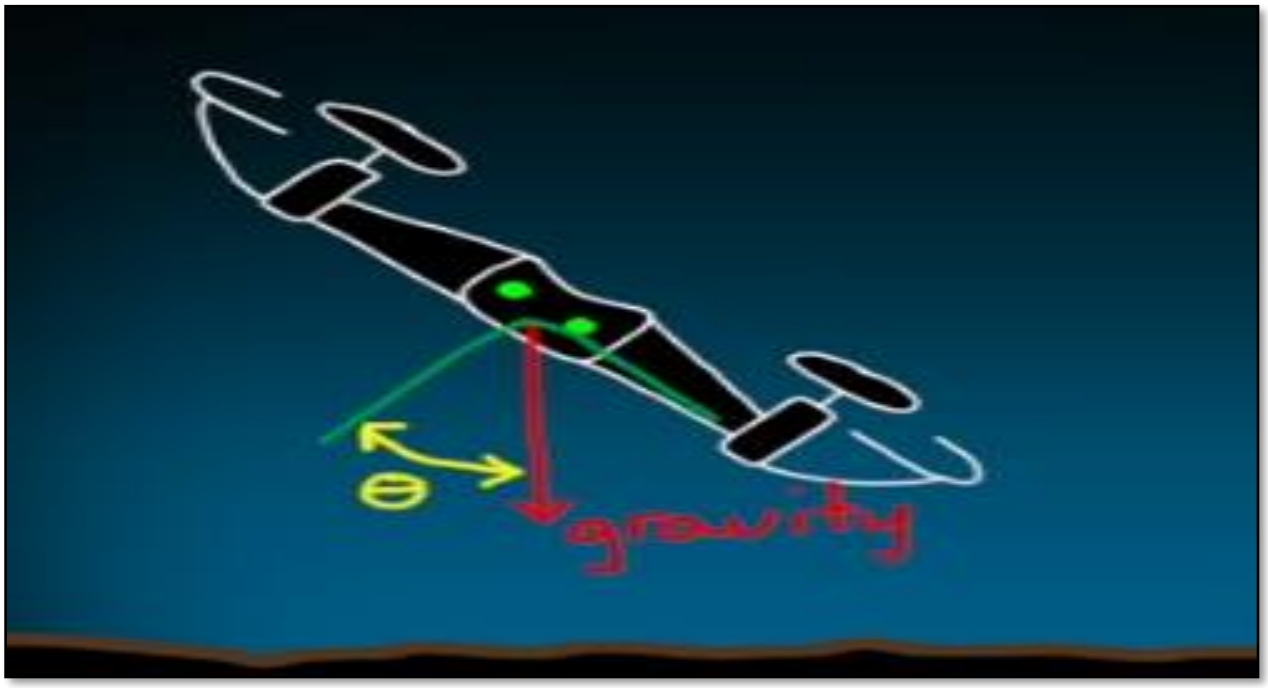


Fig 6.5: Working of IMU Sensor

The figure 6.5 depicts the working of IMU Sensor which is made up of 3-axis accelerometer which measures linear acceleration and a 3-axis gyroscope which measures angular rate.

B. COMMANDS TO MOTORS

In order to design the control system, the following characteristics have to be taken into consideration:

(a)Spin direction

(b)Roll

(c)Pitch

(d)Yaw

Opposing motors spin in the same direction and opposite direction as the other pair with roll, pitch and yaw commanded independently of each other where rotation around

the front-to-back axis is called **roll** and rotation around the side-to-side axis is called **pitch** and rotation around the vertical axis is called **yaw** as shown in the figure 6.6.

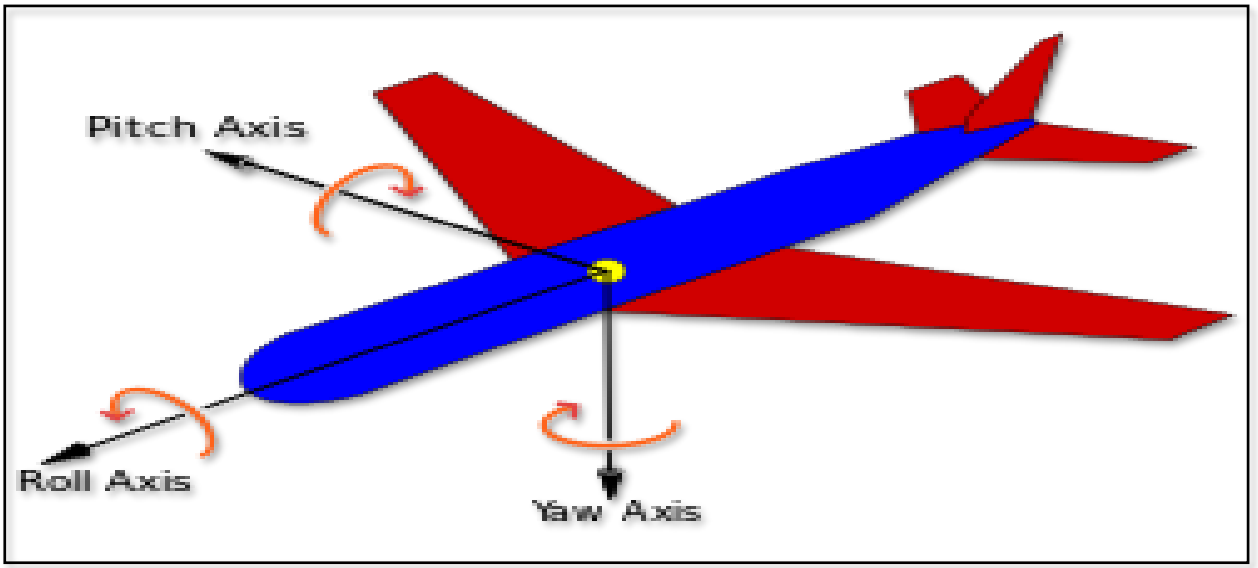


Fig 6.6: Controller Flight describing Roll, Yaw and Pitch

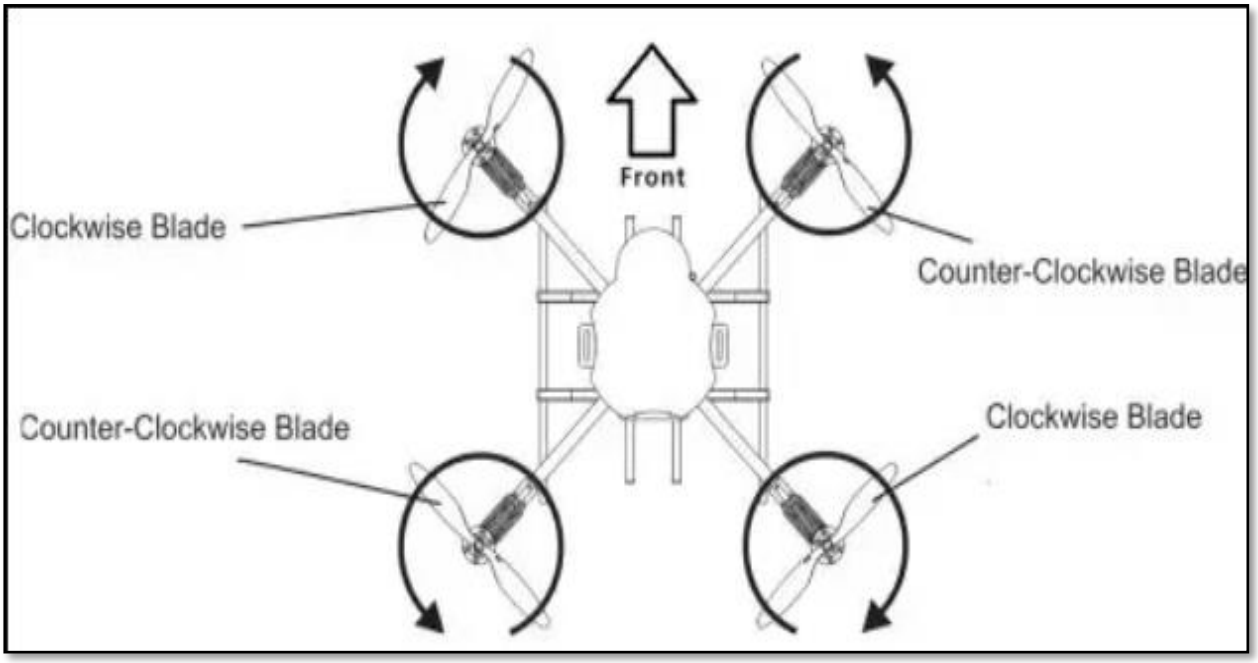


Fig 6.7: Spin Directions

3.5 MOTOR CONTROL ALGORITHM

Our main objective is to manipulate the four motors in a very precise way so that the drone can rotate and maneuver around in 3D space.

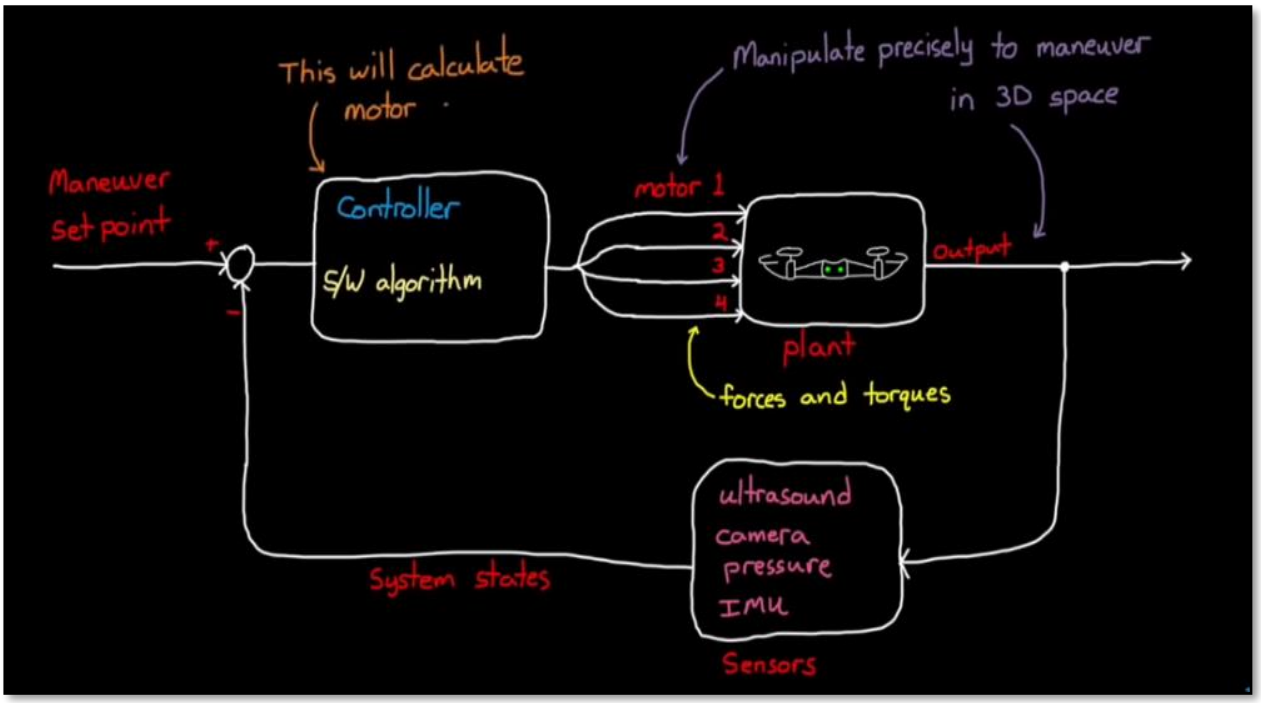


Fig 6.8: Overview of the Control Algorithm

The Figure 6.8 shows the Overview of the Control Algorithm and the explanation of each block is stated below:

- The plant in our design is the drone itself.
- There are four actuators(motors) that inject forces and torques into the system.
- To help us manipulate the four motors, sensors are used in the feedback loop which could be used directly or indirectly to estimate the state of our mini drone. The state maybe angular position, rates, altitude or horizontal velocity.
- The controller algorithm is basically an algorithm that runs in software that takes in our set point and estimated state and calculate those precise motor commands that will inject the necessary forces and torques.

Our system is basically an **underactuated system** because there are only four motors for six degrees of freedom – 3 translational directions [up and down, Left and right, Forward and backward] and 3 rotational directions [Roll, Pitch, Yaw]. Since there is no actuator for every motion, some directions are uncontrollable at any given time.

In order to make sure that the commands sent to the motor are independent of each other and also that the forces balance each other out, the following has to be included in our design.

TO ROLL

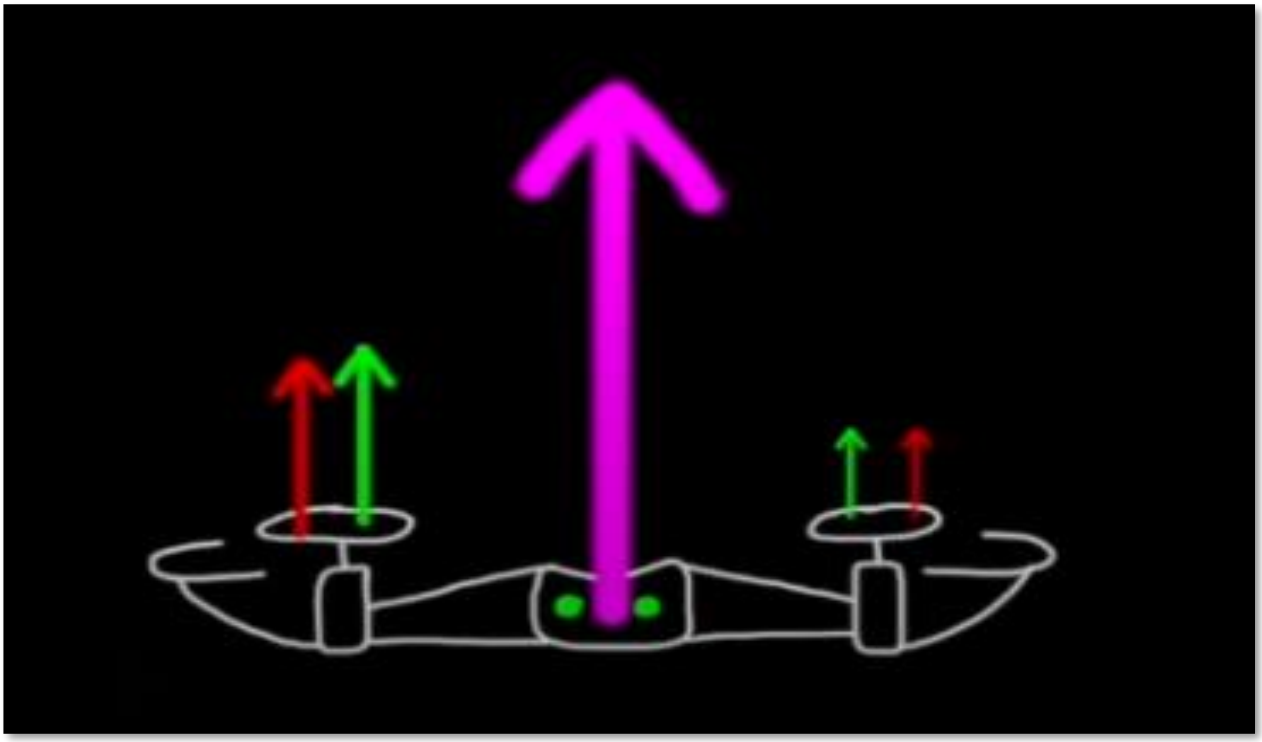


Fig 6.9: In order to Roll we decrease one of the Left/Right pairs and increase other causing a Rolling Torque

TO PITCH

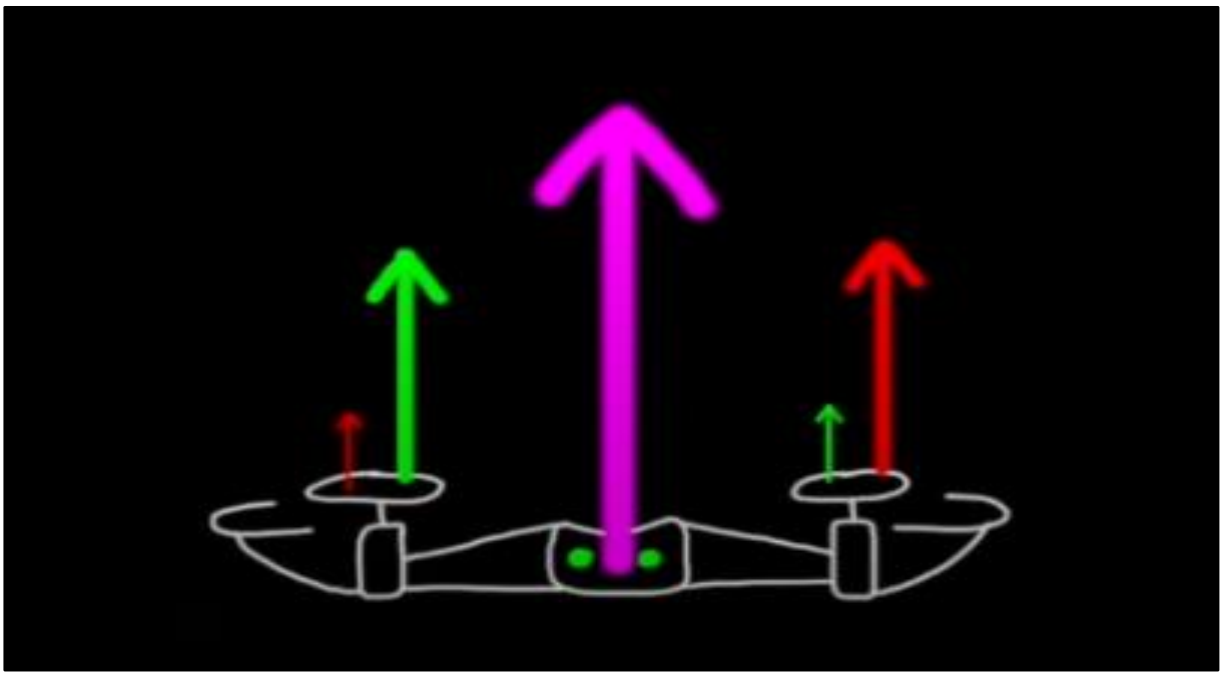


Fig 6.9: In order to Pitch we decrease one of the Front/Back pairs and increase other causing a Pitching Torque

TO YAW

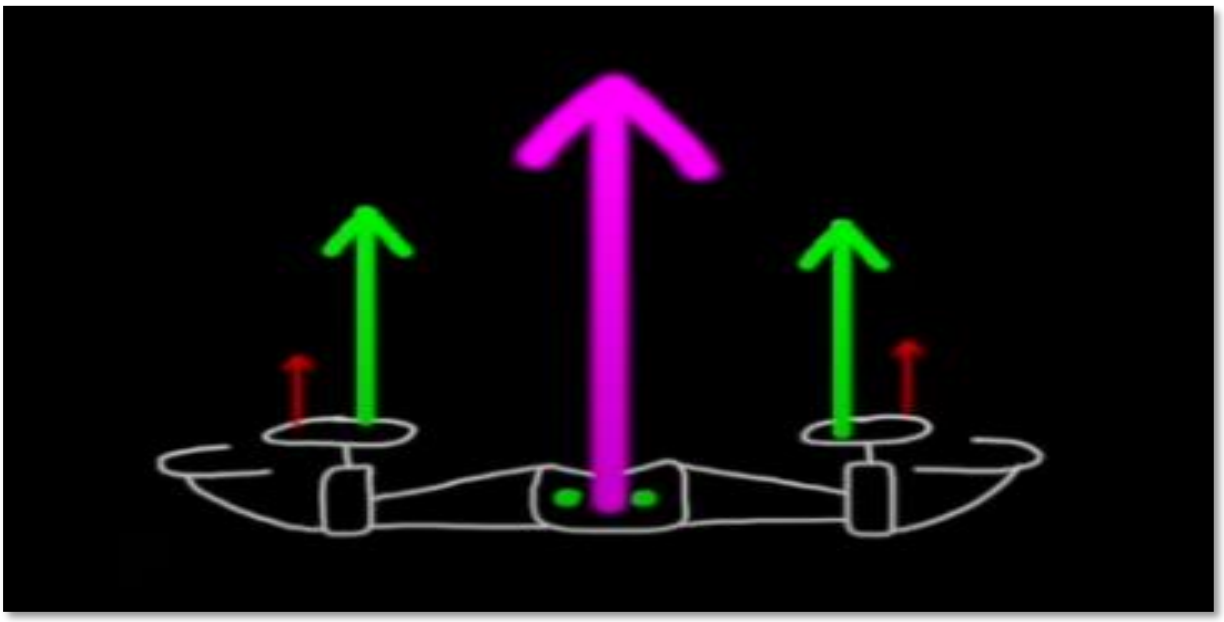


Fig 6.10: In order to Yaw we decrease the Front/Back pairs and increase other causing a Yaw

FOR THRUST

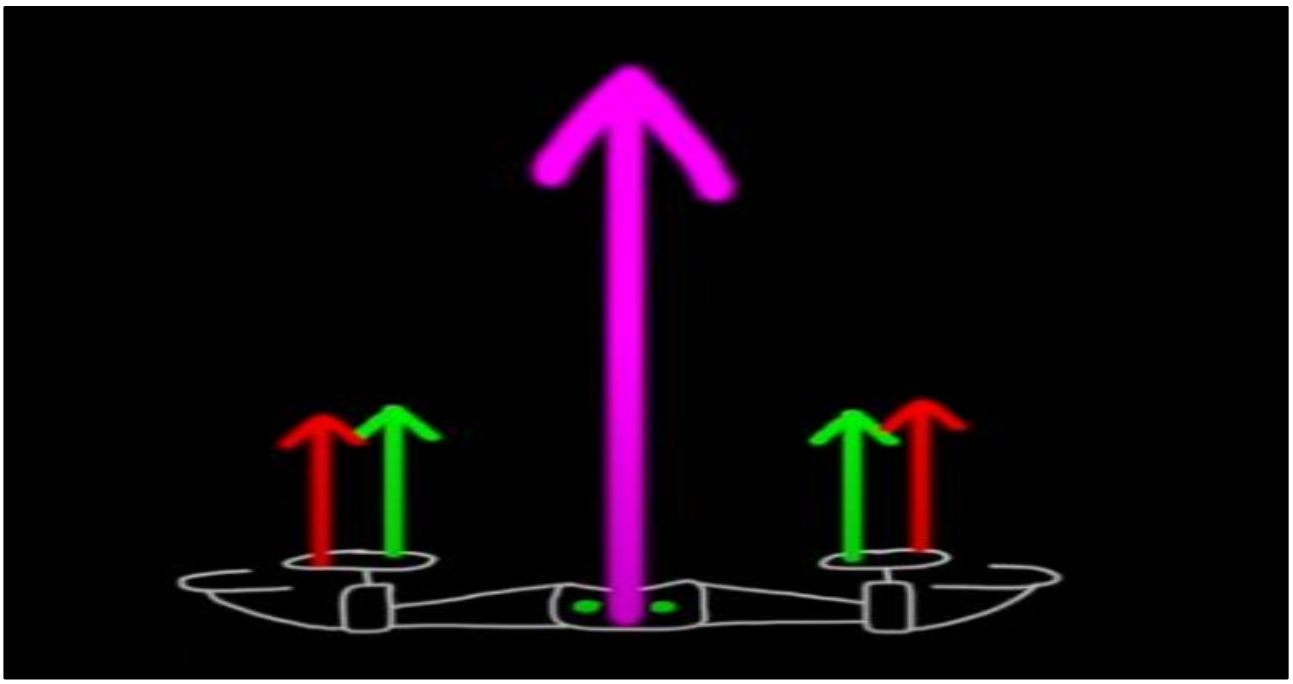


Fig 6.11: In order to produce Thrust, we decrease/ increase simultaneously

Having these into consideration, the motor mixing algorithm can be formulated as:

$$\text{MOTOR}_{\text{front, right}} = \text{THRUST}_{\text{cmd}} + \text{YAW}_{\text{cmd}} + \text{PITCH}_{\text{cmd}} + \text{ROLL}_{\text{cmd}}$$
$$\text{MOTOR}_{\text{front, left}} = \text{THRUST}_{\text{cmd}} - \text{YAW}_{\text{cmd}} + \text{PITCH}_{\text{cmd}} - \text{ROLL}_{\text{cmd}}$$
$$\text{MOTOR}_{\text{back, right}} = \text{THRUST}_{\text{cmd}} - \text{YAW}_{\text{cmd}} - \text{PITCH}_{\text{cmd}} + \text{ROLL}_{\text{cmd}}$$
$$\text{MOTOR}_{\text{back, right}} = \text{THRUST}_{\text{cmd}} + \text{YAW}_{\text{cmd}} - \text{PITCH}_{\text{cmd}} - \text{ROLL}_{\text{cmd}}$$

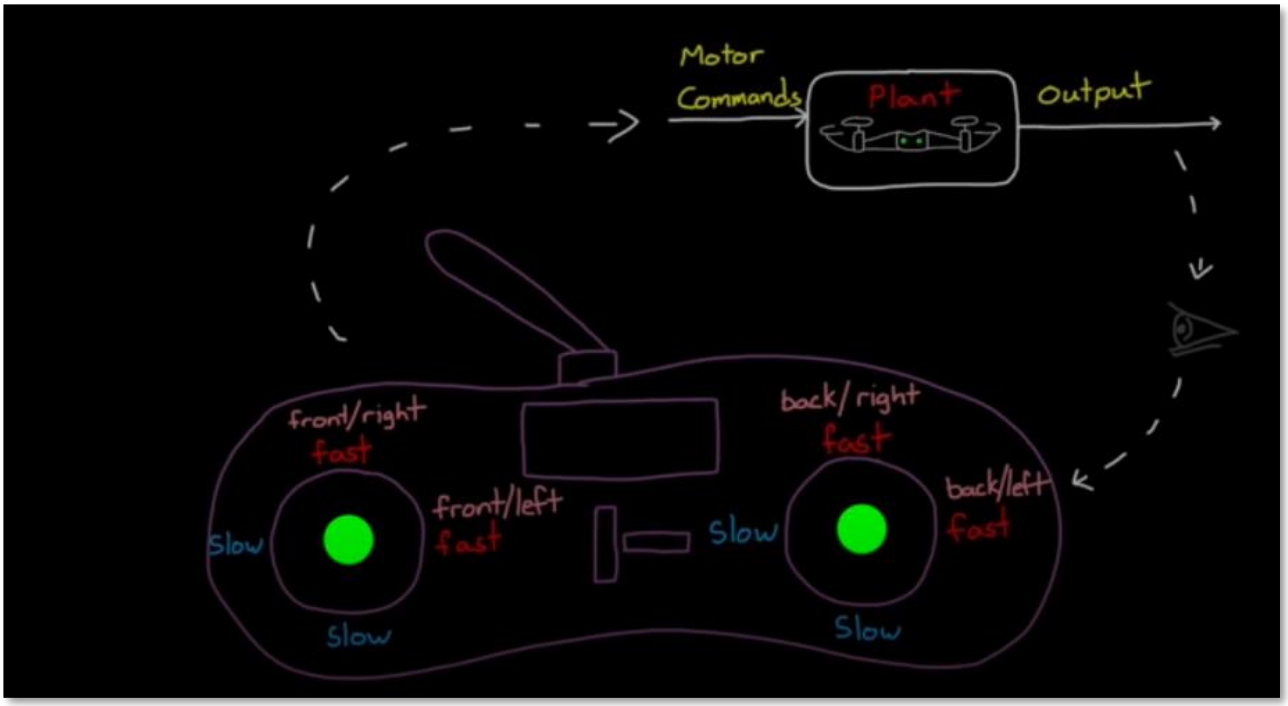


Fig 6.12: Control Design without Motor Mixing Algorithm

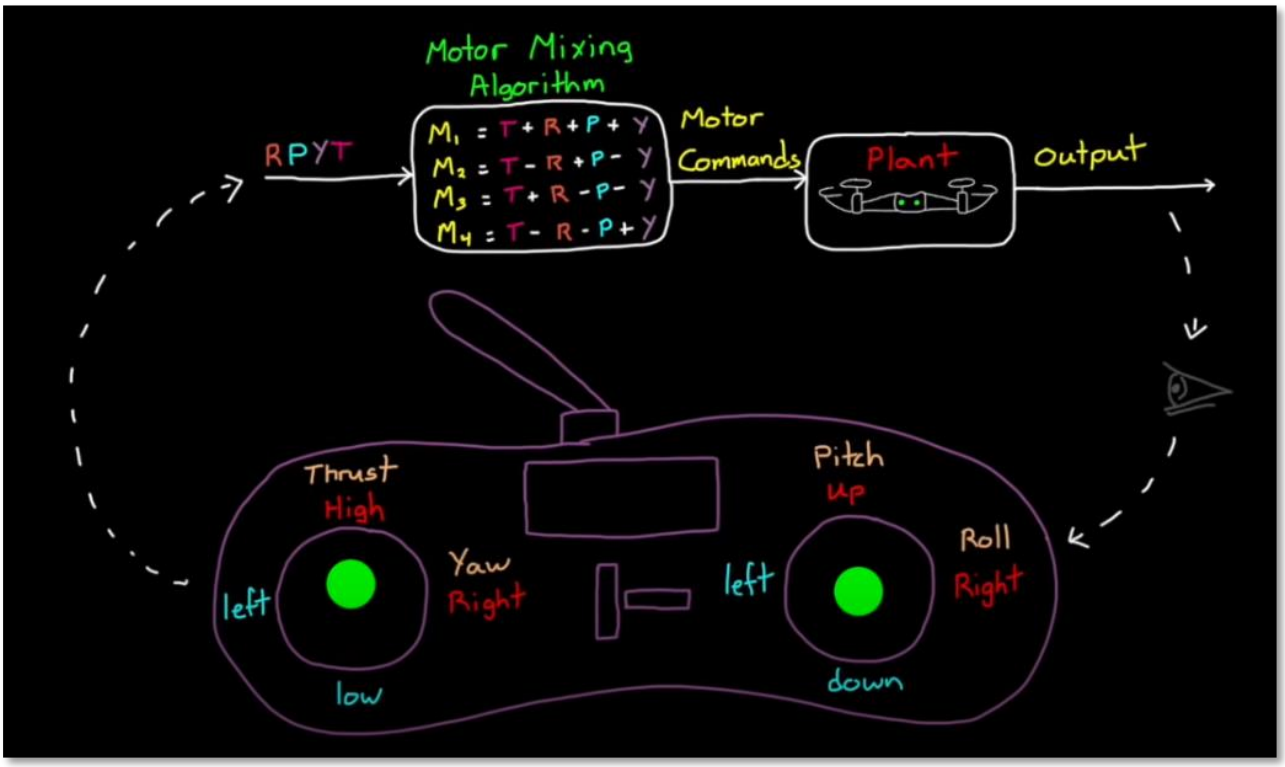


Fig 6.13: Control Design with Motor Mixing Algorithm

The Figure 6.12 shows the Control Design without Motor Mixing Algorithm and Figure 6.13 shows the Control Design with Motor Mixing Algorithm. Our design is to make the drone hover at a specified altitude. Hence the current altitude is fed back to the system as shown in the Figure 6.14.

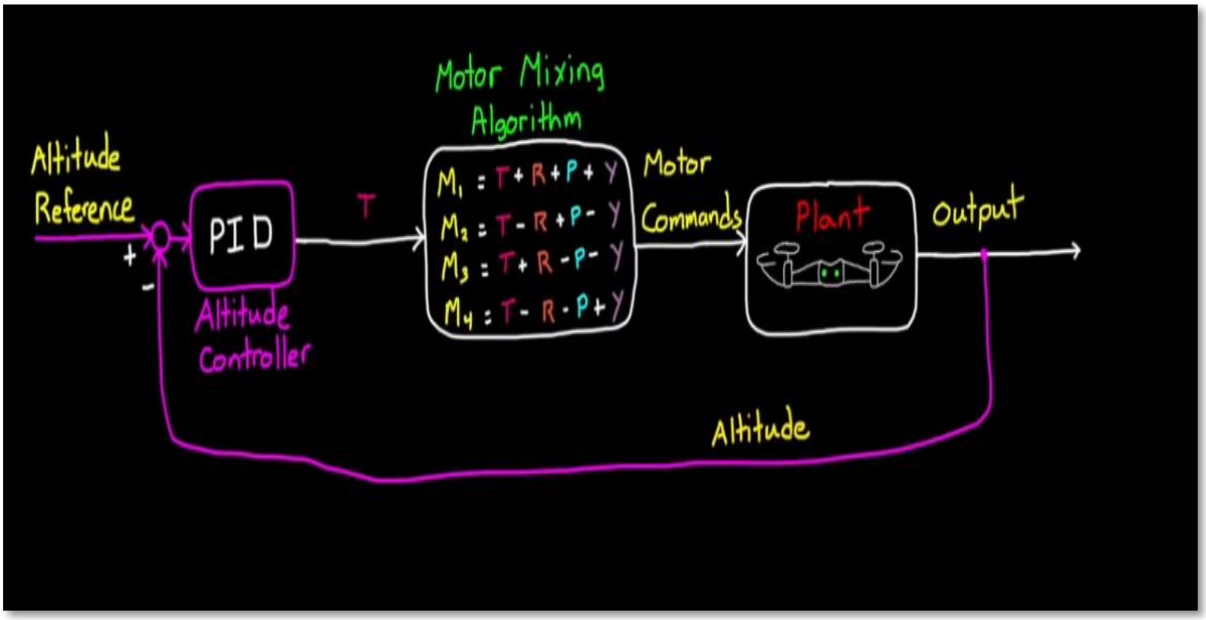


Fig 6.14: Control Design with Altitude as Feedback

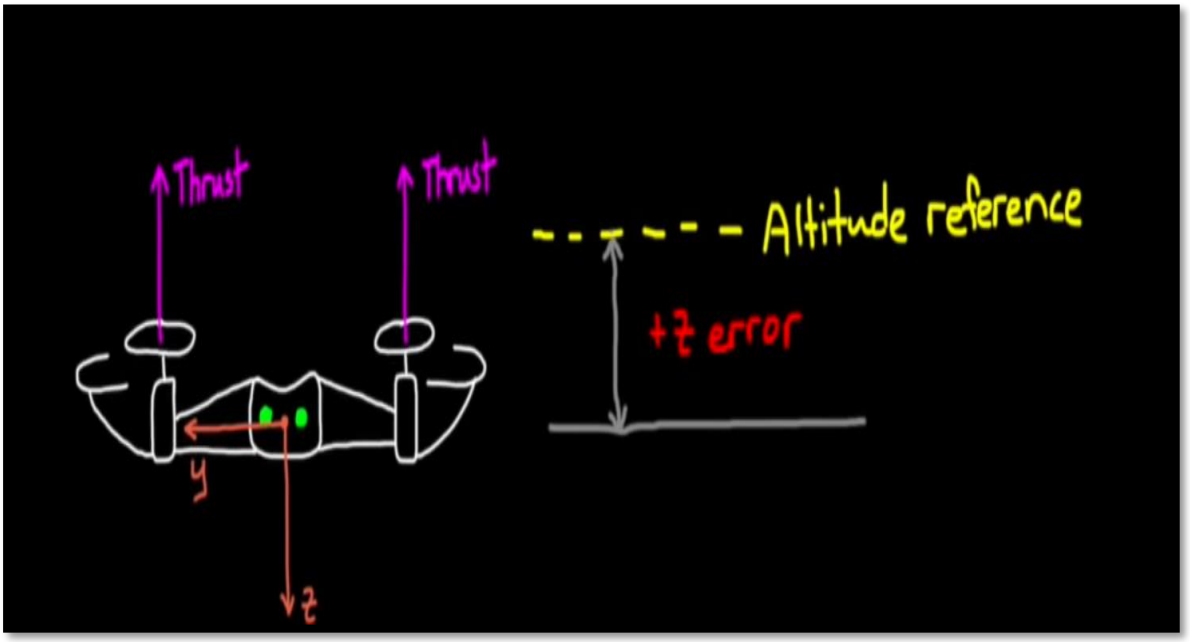
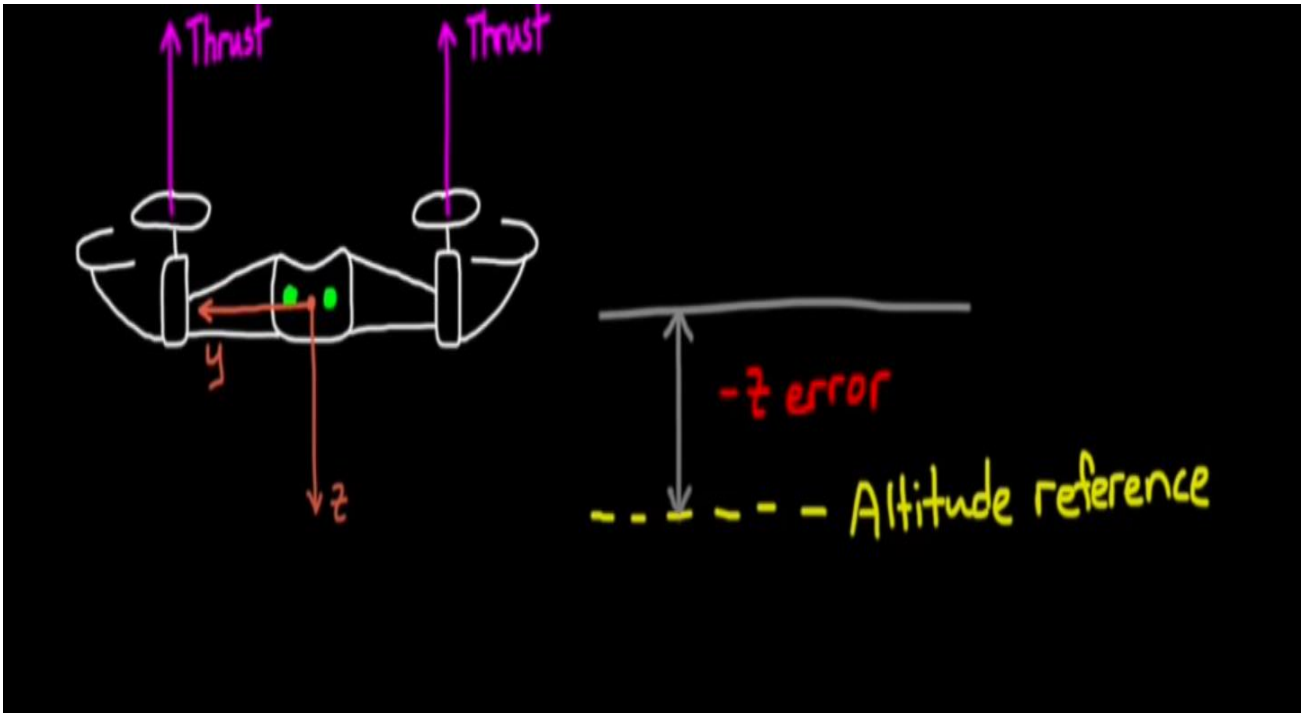


Fig 6.15: Drone at a Level Lower than the Altitude Reference



<u>Fig 6.16</u>: Drone at a Level Higher than the Altitude Reference

From the Figure 6.15 and Figure 6.16, if the drone is at a level lower than the altitude reference, it returns a positive +z error value and if the drone is at a level higher than the altitude reference, it returns a negative –z error value.

DISTURBANCES

Wind gust will induce roll or pitch externally. Thus thrust not only affects the altitude but also brings in some horizontal motion. Therefore, we need to design a control system which would maintain the roll and pitch to some position reference in accordance with the world reference frame as shown in the 6.17. This could be accomplished using three other feedback controllers. The Figure 6.18 shows the Final Control Design of the Project and the Figure 6.19 shows the implementation of the same in *SIMULINK*.

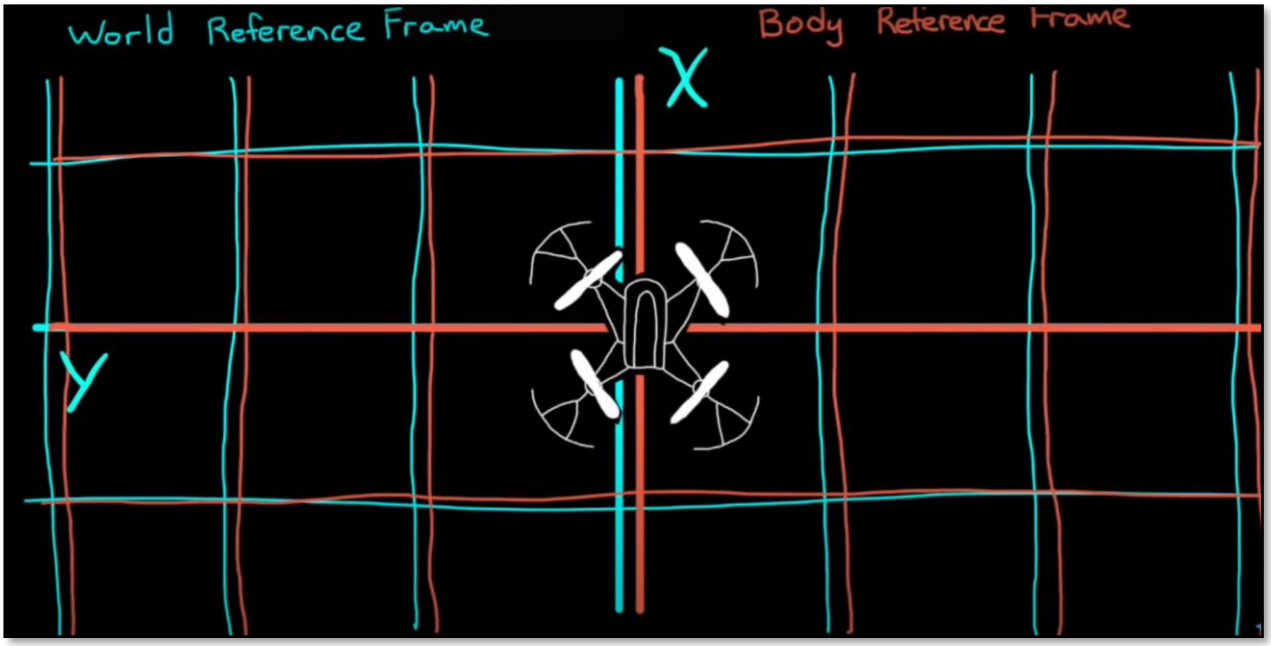


Fig 6.17: Drone's Position in accordance with World Reference Frame

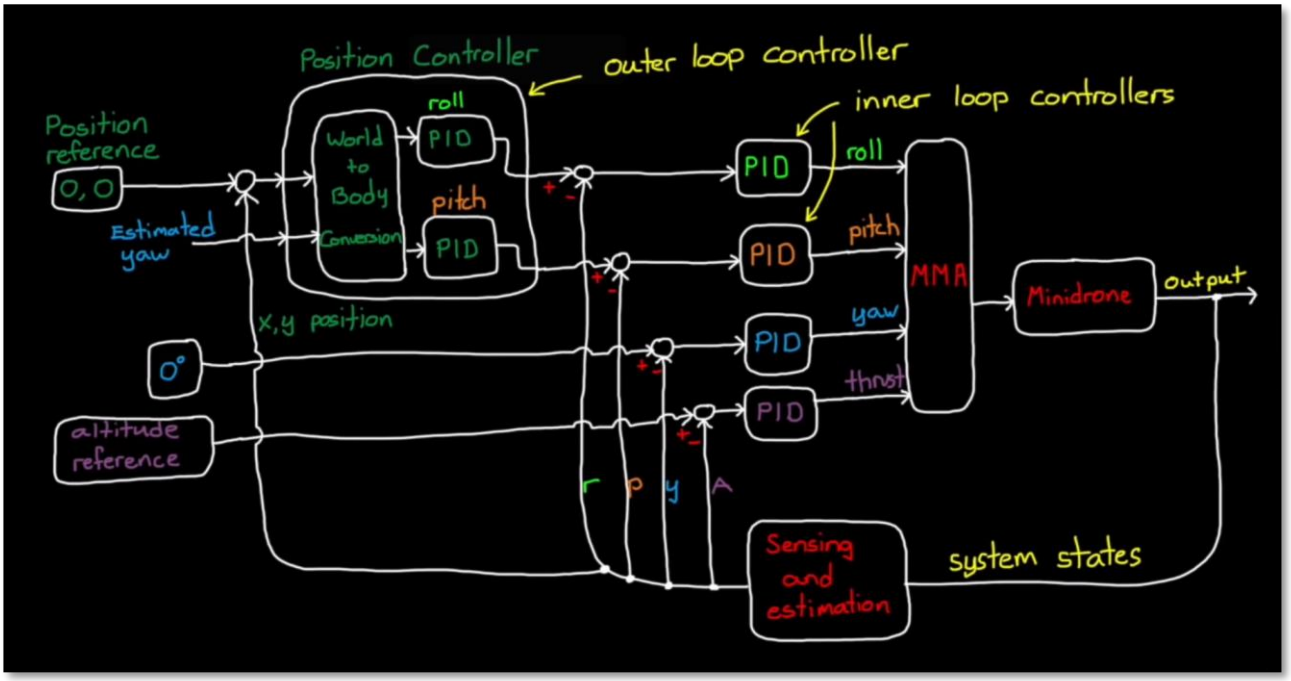


Fig 6.18: Final Control Design

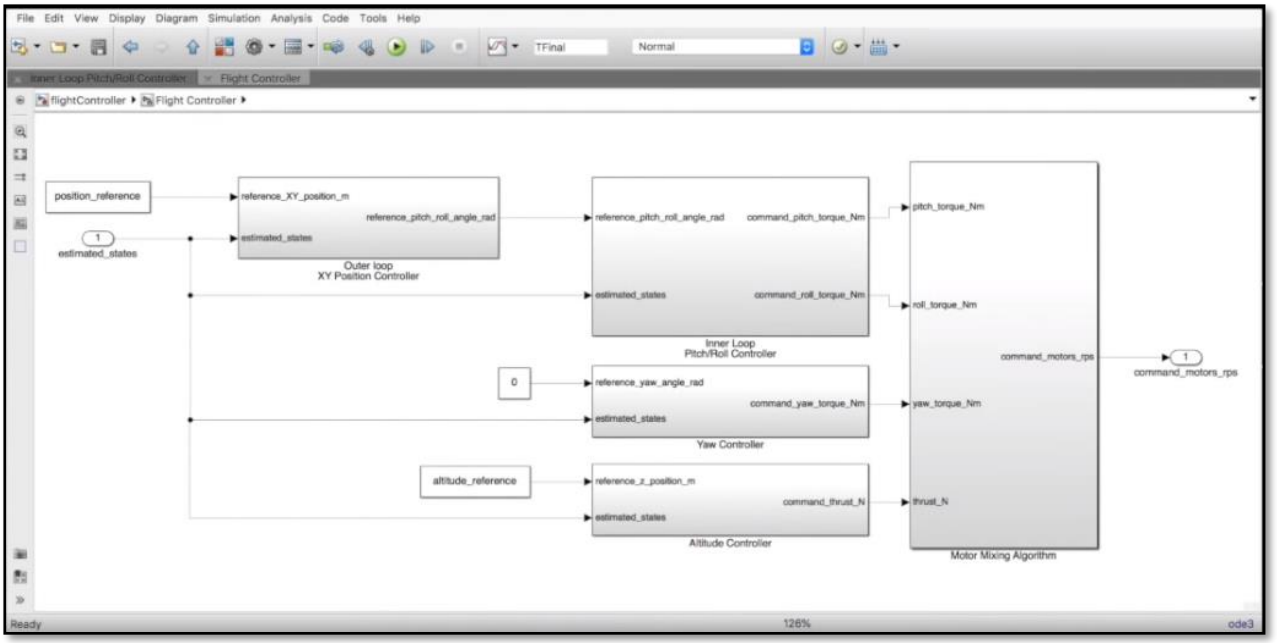


Fig 6.19: Implementation of the Designed Control System in SIMULINK

BIBLIOGRAPHY

[1] N. Gala, A. Menon, R. Bodduna, G. S. Madhusudan and V. Kamakoti, "SHAKTI Processors: An Open-Source Hardware Initiative," 2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), Kolkata, 2016, pp. 7-8. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7434907&isnumber=7434885

[2] Design of the RISC-V instruction set architecture https://riscv.org/specifications/

[3] SHAKTI Processor Program Open-source Processor Development Ecosystem https://shakti.org.in

[4] Shakti software development kit https://gitlab.com/shaktiproject/software/shakti-sdk

[5] SHAKTI C class micro architecture design https://gitlab.com/shaktiproject/cores/c-class

[6] SHAKTI E class micro architecture design https://gitlab.com/shaktiproject/cores/e-class

[7] RISC-V cores https://riscv.org/risc-v-cores/

[8] RISC V tool chain https://gitlab.com/shaktiproject/software/riscv-tools

[9] Generated RISC V tool chain https://gitlab.com/shaktiproject/software/shakti-tools

[10] Zephyr project and Zephyr OS kernel, [online], organization="Linux Foundation" https://www.zephyrproject.org/what-is-zephyr/

[11] ProgramSHAKTI on Arty A7-35T https://gitlab.com/shaktiproject/cores/shakti-soc/tree/master/fpga/boards/artya7-35t/e-class/pre-built-mcs

[12] ProgramSHAKTI on Arty A7-100T https://gitlab.com/shaktiproject/cores/shakti-soc/tree/master/fpga/boards/artya7-100t/c-class/pre-built-mcs

[13] Arty A7-100T and 35T with RISC-V https://www.digikey.in/en/product-highlight/x/xilinx/arty-a7-100t-and-35t-with-risc-v

[14] SHAKTI E class SoC on Artix 35T https://gitlab.com/shaktiproject/cores/shakti-soc/tree/master/fpga/boards/artya7-35t/e-class

[15] SHAKTI C class SoC in Artix 100T https://gitlab.com/shaktiproject/cores/shakti-soc/tree/master/fpga/boards/artya7-100t/c-class

[16] Generated RISC V tool chain https://gitlab.com/shaktiproject/software/shakti-tools

[17] PlatformIO extensions for VSCODE https://platformio.org/

[18] SHAKTI support on PlatformIO
https://github.com/platformio/platform-shakti
[19] SHAKTI support on FreeRTOS
https://gitlab.com/shaktiproject/software/FreeRTOS