HEXAPOD – AN FPGA BASED APPROACH

PROJECT REPORT

Submitted by

AZHAGAN K	**2016105017**
BHARATH RAJ V	**2016105018**
SRI HARIHARAN	**2016105592**
SURAJ SURESH	**2016105598**

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY: CHENNAI 600 025

AUGUST 2020

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report titled "**HEXAPOD – AN FPGA BASED APPROACH"** is the bonafide work of **AZHAGAN K (2016105017), BHARATH RAJ V (2016105018), SRI HARIHARAN (2016105592)** and **SURAJ SURESH (2016105598)** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr .M. MEENAKSHI	Dr .Y.V.RAMANA RAO
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Professor,	Professor,
Department of ECE,	Department of ECE,
College of Engineering, Guindy,	College of Engineering, Guindy,
Anna University,	Anna University,
Chennai-600025.	Chennai-600025.

ACKNOWLEDGEMENT

We are highly indebted to the Dean **Dr .T.V.GEETHA**, Senior Professor, for providing with sufficient facilities that contributed for the success of this endeavour. We are thankful to the Head of the Department, **Dr.M.MEENAKSHI**, Professor, for her continuous encouragement during the project work.

In performing our project, we had to take the help and guidance of some respected persons, who deserve our greatest gratitude. The completion of this project gives us much pleasure. We would like to show our gratitude to our guide **Dr .Y.V.RAMANA RAO** for giving us good guidance for the project throughout numerous consultations. We would also like to expand our deepest gratitude to all those who have directly and indirectly guided us in all our endeavours.

We would also like to thank our panel members for pointing out the mistakes and for suggesting improvements. Many people, especially our classmates and team members itself, have made valuable comments and suggestions on this proposal which gave us an inspiration to improve our project. We thank all the people for their help directly or indirectly to successfully complete our project.

Finally we would like to thank our families and friends for their constant support and guidance.

ABSTRACT

The design of any robot must be such that it is structurally robust that it could facilitate operability in rugged environment, while at the same time must be functionally efficient so as to minimize the resources and hence the cost. This proposed project involves both the dimensions giving our design the needed flexibility and stability.

This proposed project deals with the design of a Hexapod, i.e., a six legged robot, using Field Programmable Gate Array (FPGA). The usage of FPGA guarantees better speed and performance. Unlike wheeled robots, this can climb staircases and travel through uneven, rough and harsh surfaces. This bot has a fall recovery feature, which enables it to regain its position whenever it falls in any direction. The bot is manually controlled using a remote controller, which is interfaced using Wi-Fi, hence allowing a wireless control over the bot. Another added feature to the hexapod is the ability to transmit live video feed and this is achieved by using the ESP32 CAM module. By using this camera module the live video transmission from the hexapod can be viewed in a browser. This feature ensures a good quality video feed from the hexapod point of view and this helps in understanding the nature of the surroundings in which the hexapod is being operated. Another use is that it is easy to locate obstacles in every nook and corner. It is also used to take photos through the OV640 camera.

The gait locomotion of the Hexapod is implemented as a hardware module in the FPGA through Hardware Description Language (HDL) synthesis that offers considerable advantage than a conventional microcontroller based approach for the Hexapod design.

திட்டப்பணிச்சுருக்கம்

எந்தவொரு ரோபோவின் வடிவமைப்பும் கட்டமைப்பு ரீதியாக வலுவானதாக இருக்க வேண்டும், அது கரடுமுரடான சூழலில் இயங்குவதை எளிதாக்கும், அதே நேரத்தில் வளங்களை குறைக்கவும், எனவே செலவும் குறைக்க செயல்பாட்டு ரீதியாக திறமையாக இருக்க வேண்டும். இந்த முன்மொழியப்பட்ட திட்டமானது எங்கள் வடிவமைப்பிற்கு தேவையான நெகிழ்வுத்தன்மையையும் நிலைத்தன்மையையும் கொடுக்கும் பரிமாணங்களை உள்ளடக்கியது.

இந்த திட்டம் ஒரு ஹெக்ஸாபோட் வடிவமைப்பைக் கையாளுகிறது, அதாவது, ஆறு கால் ரோபோ, ஃபீல்ட் புரோகிராம் கேட் அரே (FPGA) ஐப் பயன்படுத்துகிறது. FPGA இன் பயன்பாடு சிறந்த வேகத்தையும் செயல்திறனையும் உறுதி செய்கிறது. சக்கர ரோபோக்களைப் போலல்லாமல், இது படிக்கட்டுகளில் ஏறி சீரற்ற, கடினமான மற்றும் கடுமையான மேற்பரப்புகளில் பயணிக்க முடியும். இந்த போட் வீழ்ச்சி மீட்பு அம்சத்தைக் கொண்டுள்ளது, இது எந்த திசையிலும் விழும்போதெல்லாம் அதன் நிலையை மீண்டும் பெற உதவுகிறது. போட் ஒரு ரிமோட் கன்ட்ரோலரைப் பயன்படுத்தி கைமுறையாகக் கட்டுப்படுத்தப்படுகிறது, இது வைஃபை பயன்படுத்தி இடைமுகப்படுத்தப்படுகிறது, எனவே போட் மீது வயர்லெஸ் கட்டுப்பாட்டை அனுமதிக்கிறது. ஹெக்ஸாபோடில் மற்றொரு கூடுதல் அம்சம் நேரடி வீடியோ ஊட்டத்தை அனுப்பும் திறன் மற்றும் இது ESP32 CAM தொகுதியைப் பயன்படுத்துவதன் மூலம் அடையப்படுகிறது. இந்த கேமரா தொகுதியைப் பயன்படுத்துவதன் மூலம் ஹெக்ஸாபோடில் இருந்து நேரடி வீடியோ பரிமாற்றத்தை உலாவியில் காணலாம். இந்த அம்சம் ஹெக்ஸாபோட் பார்வையில் இருந்து ஒரு நல்ல தரமான வீடியோ ஊட்டத்தை உறுதி செய்கிறது மற்றும் இது ஹெக்ஸாபோட் இயக்கப்படும் சுற்றுப்புறங்களின் தன்மையைப் புரிந்து கொள்ள உதவுகிறது. மற்றொரு பயன்பாடு என்னவென்றால், ஒவ்வொரு மூலை மற்றும் மூலையிலும் தடைகளை கண்டறிவது எளிது. இது OV640 கேமரா மூலம் புகைப்படங்களை எடுக்கவும் பயன்படுகிறது.

ஹெக்ஸாபோடின் நடை லோகோமோஷன் வன்பொருள் விளக்க மொழி (எச்.டி.எல்) தொகுப்பு மூலம் எஃப்.பி.ஜி.ஏ இல் வன்பொருள் தொகுதியாக செயல்படுத்தப்படுகிறது, இது ஹெக்ஸாபோட் வடிவமைப்பிற்கான வழக்கமான மைக்ரோகண்ட்ரோலர் அடிப்படையிலான அணுகுமுறையை விட கணிசமான நன்மையை வழங்குகிறது.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT

In this age of automation, bots are preferred for everyday chores. Robots are widely used in manufacturing, construction, assembly, packaging and many other industries. Robots serve many advantages, mainly safety, consistency, perfection, and productivity. They neither get distracted, nor need any breaks.

Whenever the term ‘Robot’ is used, people generally imagine huge structures that replace humans. But small sized bots are equally effective. Such bots can be used in agriculture, as survey bots in army camps, accident prone areas and in many other applications.

Hexapod is a bot with six legs, which comes handy while traversing rough terrains, since the wheeled bots cannot travel through them nor climb stairs. Another constraint while using wheeled bots is that it can also damage crops when being used in agriculture, whereas the hexapod can walk through the crops without yielding damage to them.

A hexapod was chosen rather than a quadra-pod because, six legs gives more stability when compared to the stability offered by a four-legged bot. Stability plays a vital role while being operated in accident prone areas or army camps.Moreover, quadrapod cant climb stairs but hexapod could. Hence a hexapod is preferred over a quadra-pod.

1.2 STRUCTURE

This project demands an unyielding and a rigid body, since it has to travel through all terrains and through uneven grounds. Such a body cannot be built using acrylics. Hence it has to be 3D printed.

3D printing is the process of building a three-dimensional object from a CAD or a digital 3D model. One of the main advantages of 3D printing is to produce complex shapes that would be impossible to be built using cardboards or acrylics. FDM (Fused Deposition Modelling) is the most commonly used 3D printing process that uses a continuous filament of a thermoplastic material that is fed from a heated printer deposited on the growing work. One such material is PLA (Polylactic acid or Polylactide).

PLA is a vegetable based plastic polymer, made from renewable resources. It is a thermoplastic aliphatic polyester and is the primary natural raw material used in 3D printing. PLA is also sought after, for its unique ability among other plastics to biodegrade in a short span of time.

Following are the design criteria,

- Symmetric legs for the bot to traverse smoothly.
- Quite a large body for the bot to hold the battery and the FPGA.
- Lengthy legs for 3 degrees of freedom.

A model satisfying all the design criteria was carefully chosen, printed and assembled.

PROBLEMS FACED

While assembling a minute dimension error had occurred, which made it difficult for the legs to fit into the body. The solution was to use acrylics to act as a house for each leg to fit in.

1.3 REQUIREMENT

1.3.1 HARWARE REQUIREMENTS

- 3D printed skeletal structure of Hexapod
- Basys 3 Artix-7 FPGA Board
- Esp32 Cam module
- Ov2640 Camera
- Joystick
- Esp8266
- Servo motors,

    1. MG995 Servo (12 nos)
        - Operating voltage: 4.8V to 7.2V
        - Stall torque @4.8V : 10 kg-cm
        - Stall torque @6.6V : 12 kg-cm
    2. MG 90s Micro Servo (6 nos)
        - Operating Voltage: 4.8V to 6V (Typically 5V)
        - Stall Torque: 1.8 kg/cm (4.8V)
        - Max Stall Torque: 2.2 kg/cm (6V)

1.3.2 SOFTWARE REQUIREMENTS

- Xilinx Vivado design suite
- Arduino Integrated Development Environment

1.4 SYSTEM OVERVIEW:

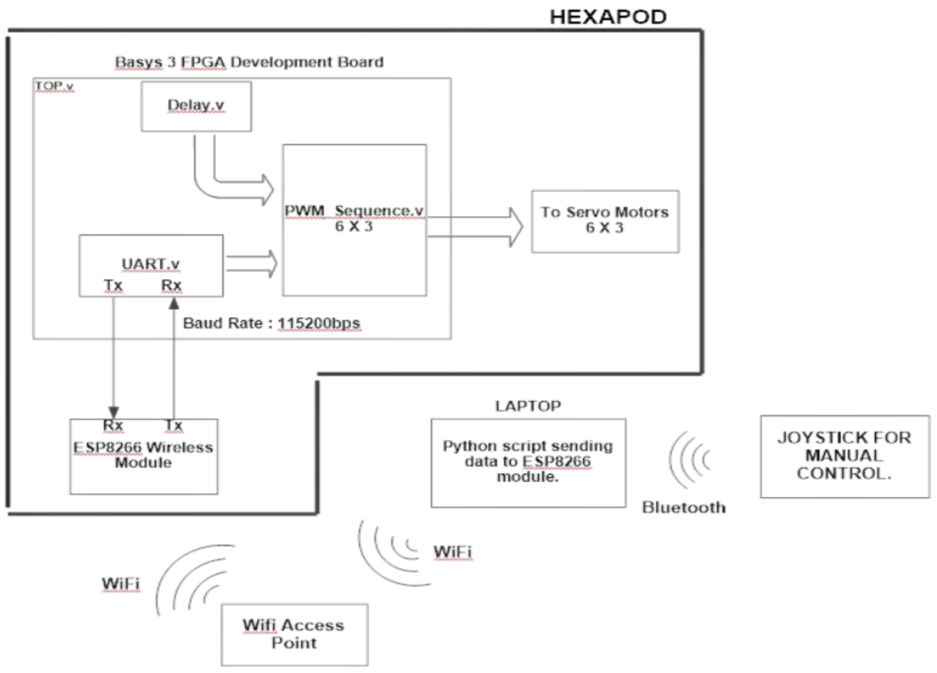


Fig 1.1 Overall Block Diagram

The user commands from the joypad is read and transmitted by the laptop to the NodeMcu through UDP. Nodemcu fixed in the Hexapod, acts as a UDP server and receives the data from laptop and sends it to Basys3 FPGA through UART at 115200bps. Uart receiver then sends it to the PWM sequence module which generates and sends appropriate control signals to the 18 servos so as to execute the command.

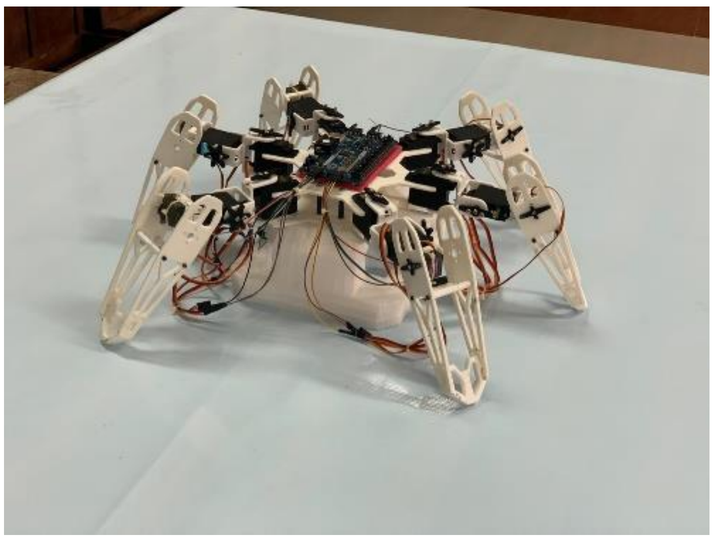


Fig 1.2 The Hexapod

CHAPTER 2

LITERATURE REVIEW

The proposed project includes the design of a Hexapod that poses as a platform for maximizing the potential of the Field Programmable Gate Arrays (FPGAs). The existence of several design projects on Hexapod has motivated us and given us the confidence to take a leap into this FPGA based approach, some of those projects and research papers have given us an in-depth knowledge on the working principle of the Hexapod. A few research papers gave us detailed insights on the gait locomotion of the bot for a much efficient and faster movement of the bot. Concisely, those projects that were in our field of work were very much inevitable and has guided us throughout the endeavour. Although these literature presents the similar field in a variety of contexts, our project differs from those in ways distinct to itself such as employing a wireless manual control of the bot, auto fall-recovery feature and along with a live video feed transmission to the user.

A hexapod design by Adam Taylor, 2018 has a similar approach through the use of an FPGA board, a Digilent Cora Z7-10 that uses the Xilinx Zynq FPGA. This was the only project that was closest to our field of work and became an indispensable reference on how to proceed in our design and gave us the basic concepts on interfacing an FPGA to the servo motors that controls the movement of the bot. This project involves an additional peripherals such as a SONAR module that facilitates the walking and navigating around its environment but we have introduced wireless manual control through a joystick that offers the user more control over the bot.

The limited literature on the FPGA based approach of the hexapod had made this attempt quite challenging but were able to overcome this through a strong fundamental understanding of the FPGA and the hardware module synthesis that made us confident on our capability to model any hardware module within the FPGA that serves to interface with the hexapod.

Another major research that has been our guidance is by, Lejla Banjanovic-Mehmedovic, Alen Mujkic, Nedim Babic and Jakub Secic, the authors of a paper titled ‘Hexapod Robot Navigation Using FPGA Based Controller, January 2020. This paper presents the Field Programmable Gate Array (FPGA) implementation of the hexapod robot navigation using the tripod gate sequence. The servo motor controller is implemented in the Cyclone IV FPGA chip by Altera using verilog as Hardware Description Language (HDL). This paper again stood as a strong fundamental that gave us sufficient information on the steps involved in coding the hardware modules through verilog along with the FPGA design flow that is essential for dumping the implemented code as a bitstream file onto the FPGA board.

All these literature has provided us with the base fundamentals on how to approach our design to which we have further extended the FPGAs’ capability by interfacing a wireless joystick for manual control that allows for operability over a longer range with much freedom. Additionally we have implemented a wireless video feed transmission that provides a visual feedback from the hexapod that assists the user in dynamic environments.

CHAPTER 3

BASYS 3 FIELD PROGRAMMABLE GATE ARRAY BOARD SPECIFICATIONS

3.1 INTRODUCTION

With the advancements in the semiconductor technologies, a number of mainstream Integrated Circuit (IC) products that use the digital circuit techniques had emerged over the recent years. The primary targets for the production of any industry standard product are circuit robustness, cost-efficiency, system integration and flexibility along with quick adaptiveness to the new emerging technologies. These necessities and requirements are fulfilled by the programmable processor cores such as the Field Programmable Gate Arrays (FPGA) chips. There are several principles such as the flexibility, scalability, reusability, portability and minimized development timespan that are taken into consideration for selecting the most suitable hardware for the design of our Hexapod.

3.2 FIELD PROGRAMMABLE GATE ARRAYS

A Field Programmable Gate Array (FPGA) is an integrated circuit designed so that it could be configured by the designer after manufacturing. The FPGA configuration is basically modelled using the Hardware Description Languages (HDLs). The basic skeletal structure of the FPGA is that it contains a large array of programmable logic blocks and a hierarchy of reconfigurable interconnects that allow these blocks to be wired together, similar to the logic gates that are inter-wired in different configurations so as to enable specific functionality. These logic blocks are configured so as to perform the functionality as that of the combinational circuits (XOR, NAND). They also include memory elements, which are simple flip-flops or a complete blocks of memory for more memory intensive applications.

More advanced FPGAs that are produced in the market in recent times have large resources of logic gates and Random Access Memory (RAM) blocks to implement complex digital computations. In general, FPGA designs employ very fast I/O rates and bidirectional data buses, enabling the designers to exploit this high speed transfers, for intensive applications where speed of computation is crucial. FPGAs have the capability to implement any functionality that an Application Specific Integrated Circuit (ASIC) chip can perform. But however, on comparison, FPGAs are found to be slower and less energy efficient than their ASIC counterparts. A study showed that, the designs implemented on FPGAs need on average 40 times as much area, draw 12 times as much power, and run at one third the speed of corresponding ASIC implementations[1]. But however given the cost-intensive and the time consuming attributes associated with ASIC design, FPGA prototyping phase became essential. Advantages of FPGAs include the ability to re-program when already deployed to fix bugs, and often include shorter time to market and lower non-recurring engineering costs.

[1] I.Kuon, J. Rose. Measuring the Gap between FPGAs and ASICs, February 2006.

3.2.1 ARCHITECHTURAL DETAILS OF THE FPGAs

The most common FPGA architecture consists of an array of logic blocks, I/O pads, and routing channels. Multiple I/O pads fit into the height of one row or the width of one column in the array. An application circuit must be mapped into an FPGA with adequate resources. The number of Computational Logic Blocks (CLB) and I/Os required is easily determined from the design but the number of routing tracks needed may vary considerably even among designs with the same amount of logic. In general, a CLB consists of a few logical cells. A typical cell consists of a 4-input Look Up Table (LUT), a full adder (FA) and a D-type flip-flop, as shown in the Figure 3.1. Modern FPGAs contain a heterogeneous mixture of different blocks like dedicated memory blocks, multiplexers. Configuration memory is used throughout the logic blocks to control the specific function of each element.

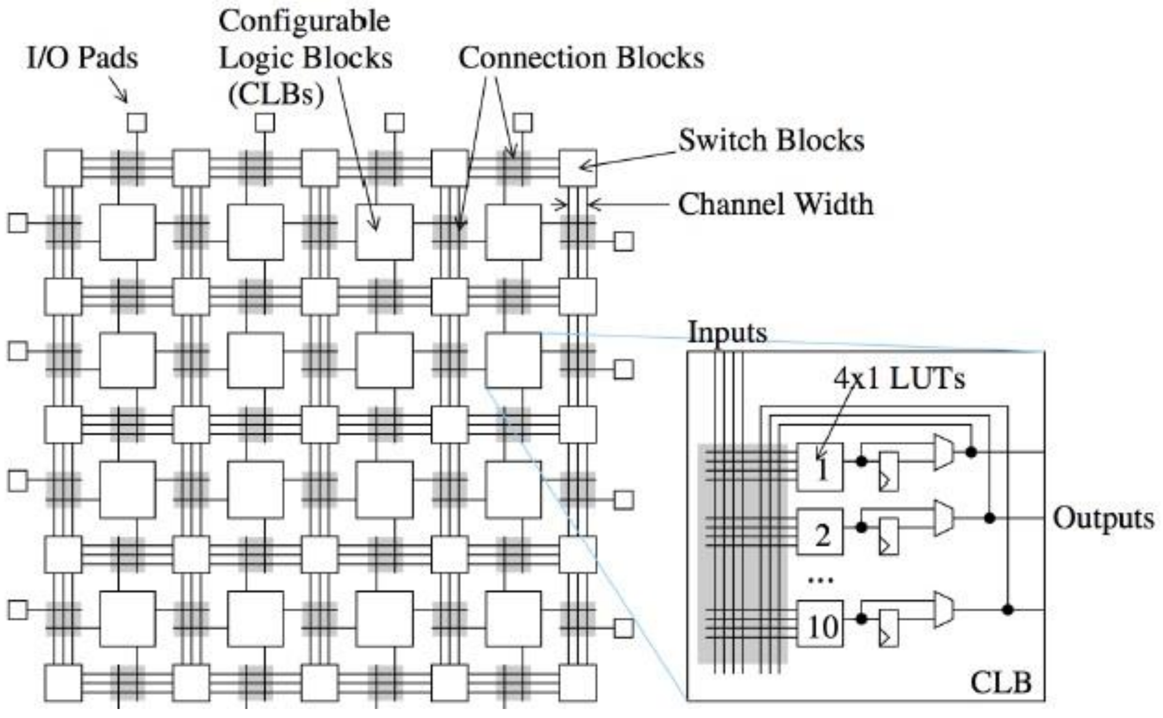


Figure 3.1. Internal Architecture of the FPGA.

The functions of an FPGA architecture module are discussed below,

- CLBs (Configurable Logic Blocks) includes digital logic, inputs, and outputs. It implements the user logic.
- Interconnects provide direction between the logic blocks to implement the user logic.
- Depending on the logic, switch block matrix provides switching between interconnects.
- I/O Pads are used for the outside world to communicate with different applications.

3.2.2 DESIGNING THE FPGA CONFIGURATION

An FPGA-based design begins by defining the required computing tasks in the development tool, then compiling them into a configuration file that contains information on how to hook up the CLBs and other modules. The process is similar to a software development cycle except that the goal is to architect the hardware itself rather than a set of instructions to run on a predefined hardware platform. Designers have traditionally used a hardware description language (HDL) such as VHDL or Verilog to design the FPGA configuration**.** Once the FPGA design has been created and verified using HDL, the compiler takes the text-based file and generates a configuration file that contains information on how the components should be wired together. Even if the HDL code has no errors, choosing the wrong FPGA may still cause the compilation to fail—for example, the FPGA runs out of a specific resource type or the compiler cannot create the required routes between components.

The advancement in process technology has greatly enhanced the logic capacity of FPGAs and has in turn made them a viable implementation alternative for larger and complex designs such as our Hexapod. Further, programmable nature of their logic and routing resources has a dramatic effect on the quality of final device’s area, speed and power consumption.

3.3 THE BASYS3 ARTIX-7 FPGA

3.3.1 DETAILS OF THE BASYS3 FPGA DEVELOPMENT BOARD

The primary goal in our design of the Hexapod is that it must be modular, flexible so as to add more features and cost-effective and in our search for the most appropriate FPGA board to satisfy as our design needs, BASYS3 Artix-7 series FPGA board from Xilinx stood the best among all the aspects. The minimal space occupancy of the board also was also an added advantage to easily incorporate the board within the Hexapod structure.

The BASYS3 board is a complete, ready-to-use digital circuit development platform based on the latest Artix-7 Field Programmable Gate Array (FPGA) from Xilinx. With its high-capacity FPGA, low overall cost, and collection of USB, VGA, and other ports, the Basys3 can host designs ranging from introductory combinational circuits to complex sequential circuits like embedded processors and controllers. It includes enough switches, LEDs and other I/O devices to allow a large number designs to be completed without the need for any additional hardware, and enough uncommitted FPGA I/O pins to allow designs to be expanded using Pmod interface[2] or other custom boards and circuits.

[2] Pmod interface is an open standard defined by Digilent Inc in the Digilent Pmod Interface Specification for peripherals used with FPGA or microcontroller development boards.

3.3.2 FEATURES OF THE BASYS3 FPGA

- 4 Pmod ports: 3 Standard 12-pin Pmod ports, 1 dual purpose XADC signal / standard Pmod port.
- 4-digit 7-segment display.
- 5 user push buttons.
- 16 user LEDs.
- 16 user switches.
- USB HID Host for mouse, keyboards and memory sticks.
- 12-bit VGA output.
- USB-UART Bridge.
- 32 Mbits Serial Flash memory.
- Digilent USB-JTAG port for FPGA programming and communication.
- On-chip analog-to-digital converter (XADC).
- Internal clock speeds exceeding 450 MHz
- 90 DSP slices.
- Five clock management tiles, each with a phase-locked loop (PLL).
- 1,800 Kbits of fast block RAM.
- 33,280 logic cells in 5200 slices (each slice contains four 6-input LUTs and 8 flip-flops).

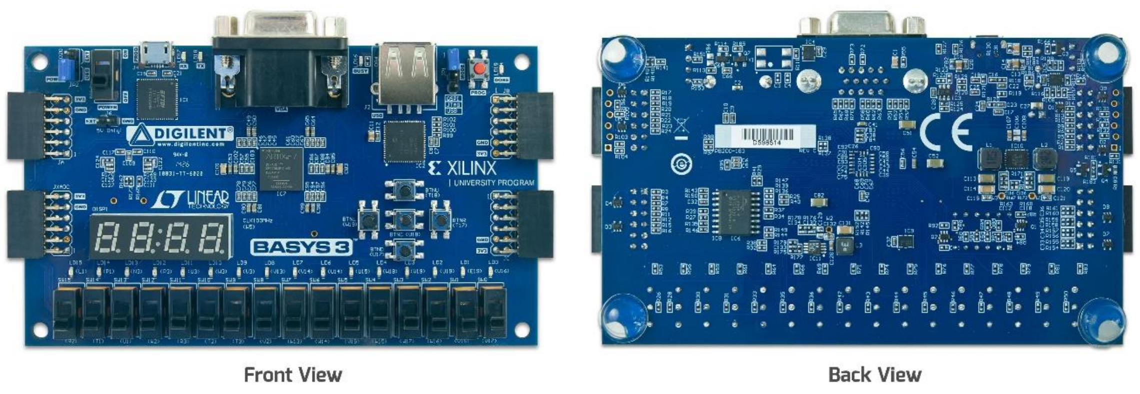


Figure 3.2. Basys3 FPGA development board

3.4 ARTIX-7 SERIES FPGA CORE

3.4.1 XC7A35T-1CPG236C SPECIFICATIONS

The BASYS3 FPGA development board supports an Artix-7 FPGA core along with certain other additional peripherals that must be configured before it could be used to interface with the external environment. Apart from the advantages the Basys3 board provides, the Artix-7 FPGA has its own unique attributes. Artix-7 family is optimized for low power applications requiring serial transceivers and high DSP and logic throughput. It provides the lowest total bill of materials cost for high-throughput, cost-sensitive applications. The Artix-7 series FPGA incorporated within the Basys3 development board is specified XC7A35T-1CPG236C.

The 7 series FPGAs from Xilinx is built on a state-of-the-art, high-performance, low-power (HPL), 28 nm, high-k metal gate (HKMG) process technology, 7 series FPGAs enable an unparalleled increase in system performance with 2.9 Tb/s of I/O bandwidth, 2 million logic cell capacity, and 5.3 TMAC/s (trillion multiply-accumulates per second) DSP, while consuming 50% less power than previous generation devices to offer a fully programmable alternative to ASSPs and ASICs.

3.4.2 KEY CAPABLITIES AND FEATURES

- 6.6Gb/s transceivers enabling 211Gb/s peak bandwidth (full duplex).
- Single and double differential I/O standards with speeds of up to 1.25Gb/s.
- 740 DSP slices with up to 930 GMAC/s (giga multiply-accumulates/s) of signal processing.
- 1,066Mb/s DDR3 memory, including SODIMMs (Small-Outline Dual Inline Memory Module) support.

- 200+ DMIPs MicroBlaze processor in microcontroller, real time processor, or application processor configuration.
- Integrated memory interface for streamlined access.
- 50% lower total power compared to previous generation.
- Sub-watt performance ranging from 13K–2,000K logic cells.
- 2X logic, 2.5X block RAM, 5.7X more DSP slices compared to the previous generation FPGAs.
- Lowest-power Industrial speed grade offering.

CHAPTER 4

FPGA DESIGN FLOW IMPLEMENTATION

4.1 INTRODUCTION

The core of the development of the Hexapod and the wireless control of the structure involves the implementation of various different modules within the FPGA board using the Hardware Description Language (HDL), Verilog that follows a certain hierarchy in the FPGA design flow. The control of the motors in the arms of the hexapod structure, the proper sequence of motor actuation for specific movements and the wireless control through a joystick all must be implemented as hardware modules following the protocols and syntax of the Verilog language. Vivado Design Suite is the environment in which the synthesis and analysis of the HDL designs are undertook. It is user friendly in that, the processes involved in the FPGA design flow are sequentially carried out. The inbuilt Vivado Simulator also enables for the verification of the design before being implemented onto the FPGA board. The BASYS3 board works with the Vivado Design Suite that includes many tools and design flows that facilitate and enhance the latest design methods. It runs faster, allows better use of FPGA resources, and allows designers to focus their time evaluating design alternatives.

4.2 FPGA DESIGN FLOW

The implementation of the hardware modules with the required specifications on the FPGA development board basically involves a sequential flow of processes as given below.

Design Entry

The hardware modules are designed as per the specifications using the Hardware Description Language, also facilitates design through schematic approach or state-machine approach.

⇩

Design Synthesis

The input HDL is essentially converted into a netlist which lists the logic elements that will be needing for the project and interconnects needed in the specific hierarchy. Also eliminates redundant logic.

⇩

Design Implementation

The layout of your design will be determined and this consists of three steps: translate, map, and place & route. Constraints such as the position and assignment of the pins, timing requirement such as maximum delay or input clock period are set by the user.

⇩

Device Programming

The last step in the design flow is to load the mapped out and completely routed design into the BASYS3 FPGA board by generating a Bitstream file.

4.3 PRIMARY HARDWARE MODULES FOR HEXAPOD

4.3.1 PWM SIGNAL GENERATION

The basic design of the Hexapod constitutes a total of 18 servo motors that need to be interfaced independent of each other so as to individually control every servo motor in the structure and this independent control facilitates a much fluid movement of the bot and added unique movements such as scaling short heights is made possible. Each leg of the Hexapod comprises of 3 servo motors providing a 3 DOF (degrees of freedom) movement per leg that facilitates faster and smooth movement. The BASYS3 board provides with sufficient I/O Pmod (Peripheral Module) ports necessary for the independent control of all 18 servo motors.

Servo motors are preferred as they are compact and offering sufficiently high torque while being energy-efficient. It is comprised of a DC motor, a potentiometer and a control circuitry. As the motor rotates, the potentiometer's resistance changes, so the control circuit can precisely regulate how much movement there is and in which direction.

Servo motors are controlled by sending an electrical pulse of variable width, or pulse width modulation (PWM), through the control wire. There is a minimum pulse, a maximum pulse, and a repetition rate. A servo motor can only turn 90° in either direction for a total of 180° movement. The PWM sent to the motor determines position of the shaft, and based on the duration of the pulse sent through the control wire; the rotor will turn to the desired position. The servo motor expects to see a pulse every 20 milliseconds (ms) and the length of the pulse will determine how far the motor turns.

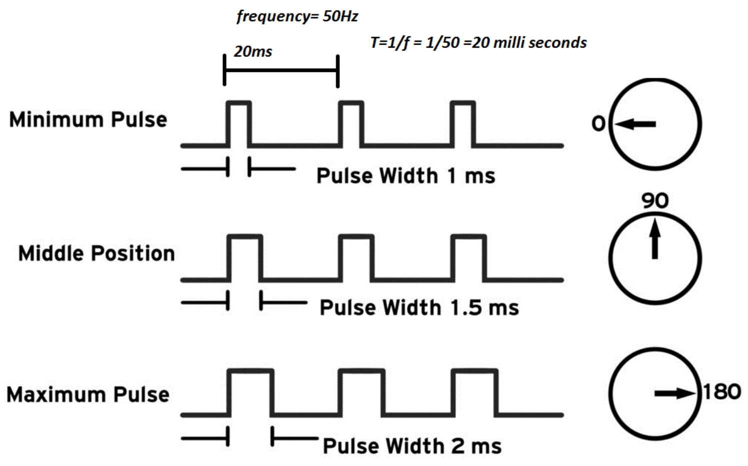


Figure 4.1. Variable pulse width control of servo position

Here as per the manufacturer specifications of the servo, a 1.5ms pulse will make the motor turn to the 90° position. Shorter than 1.5ms moves it in the counter clockwise direction toward the 0° position, and any longer than 1.5ms will turn the servo in a clockwise direction toward the 180° position.

IMPLEMENTATION AS HARDWARE MODULE

The control signals necessary to actuate the servo motors of the Hexapod are created as hardware modules using the HDL synthesis. The BASYS3 FPGA has an internal clock with a frequency of 100MHz and this could be used as the global clock for all the hardware modules to be designed as a part of the project. The PWM control signal is a periodic signal that must repeat once every 20ms. Thus a pulsating signal of frequency 50Hz needs to be generated initially.

Frequency = 1/ Time Period;

1/20ms = 0.05KHz => 50Hz.

Henceforth the internal clock must be downscaled to 50Hz with the help of a counter. Before instantiating the counter (C1) the required count value up to which the counter must count before it complements has to be determined for the accurate downscaling, as even single clock error propagates leading to failure in actuation of motors.

$$\text{Count\_Value} = \frac{[\text{System\_Clock\_Frequency/ Required\_Clock\_Frequency}]}{2}$$

$$\Rightarrow \frac{[100\text{ MHz}/5\text{ KHz}]}{2}$$

Count_Value= 10000.

Thus a the counter that need to be instantiated in the Verilog code must be capable of counting in decimals up to 9999 before it complements. In order for this counting to be feasible a 14-bit counter is required after analysing the count value in binary notation. The output of the above designed counter is now a clock signal pulse with time period of 200µs.

This clock signal is further used to count a separately instantiated counter (C2) independent of C1, that counts up to 100 as this facilitates an easier way of generating the pulses of required width and gives the user the at most control over the Duty Cycle of the pulse width modulated signals by simply using the counter C2. That is, to actuate the servo motor to the neutral position of 90° the required pulse width is 1.5ms and it is generated by making C2 to count up to 15 and pulse is asserted to 0 until the next periodic pulse after 20ms is generated.

4.3.2 SEQUENTIAL CONTROL FOR THE MOVEMENTS

With the generation of all the 18 individually controllable PWM signals it is now feasible to define the movements of the hexapod as and how we require. Now as all the PWM signals are independent of one another the movement of the servo motors in each of the legs can be actuated concurrently thereby enabling a much fluid or smooth motion of the bot. Apart from the generic motions on a levelled surface, this advantage of having able to control each of the 18 motors can be exploited so as to enable to Hexapod to scale an elevation or tread through uneven surfaces.

Even in such events where the Hexapod topples over it is now possible to regain its state of motion and stay active as our design facilitates a complete 180° rotation of the servo motors across all the six legs of the bot. This complete flexibility makes the bot suitable for any harsh terrain.

Now making use of the generated variable pulse width signals from the created hardware module, the necessary order or the sequence in which each of the motors (3 DOF of each leg) must be activated is written in the form of the Verilog code. One primary challenge in this design as a hardware module is that the synthesisable codes are concurrent and not sequential, that is all the signal pulses will be forever present and a particular signal cannot be stopped while another signal requires a certain delay. This challenge was overcome by making use of a Multiplexer (MUX) based hardware implementation.

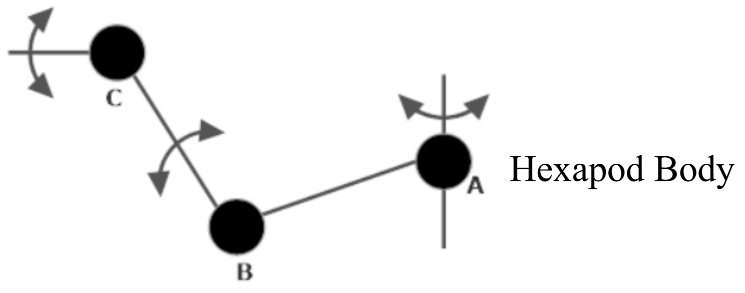


Figure 4.2. 3-DOF (degrees of freedom) representation

IMPLEMENTATION AS HARDWARE MODULE

This concurrent nature of execution of the synthesisable RTL code for the correct sequence of motor actuation has resulted in finding alternative ways of hardware implementation. One of the ways of implementation found to be solving this is by employing a MUX based design as shown below.

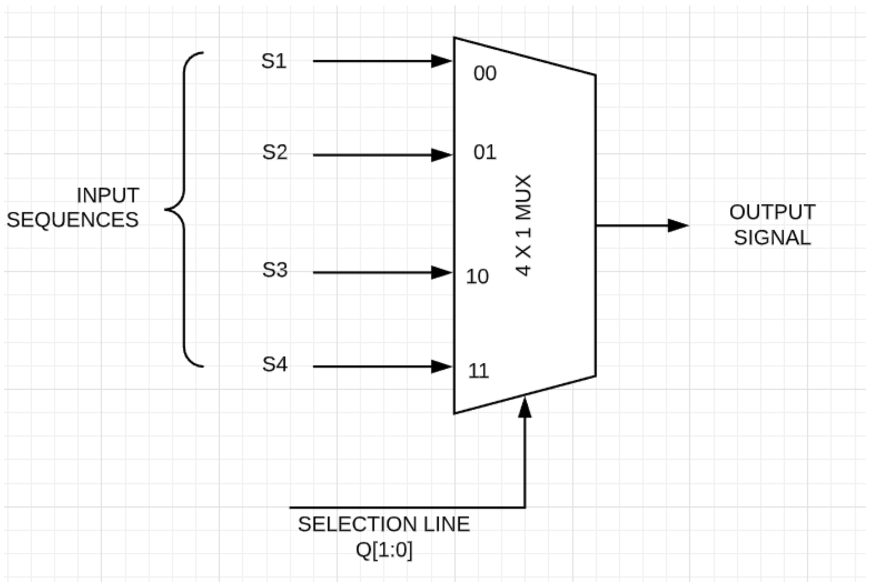


Figure 4.3. MUX based design with PWM signal sequence as input.

From the above design it is clear that a 4 X 1 MUX component is to be implemented as a module on the FPGA. This MUX comprises of 4 inputs wherein, each of the inputs corresponds to a particular PWM signal sequence having a specific duty cycle. Now due to the concurrent nature all the input PWM signals S1, S2, S3, S4 are continuously available and the execution of one particular signal cannot be stopped. To solve this we make use a SELECTION LINE, here defined as Q [1:0] a 2 input selection line.

Now this Q [1:0] is implemented as a register in Verilog, and basically as a 4-bit counter that resets after 11 (binary notation). The count value in this register only increments after a certain amount of delay that is defined by the user.

Thus as the delay is also under the user's control this gives complete modularity and flexibility to control when exactly the motor must actuate. Thus each of the PWM signals are latched to the output in the same order S1, S2, S3, and S4 with a delay between each signal.

BASIC MOVEMENTS SEQUENCE AS TABULATION

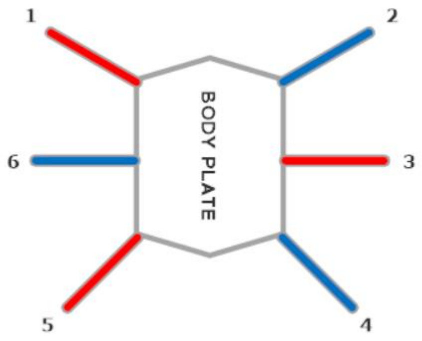


Figure 4.4. Legs (1, 3, 5) and (2, 4, 6) with synchronized movement.

FORWARD MOVEMENT

	Leg 1	Leg 2	Leg 3	Leg 4	Leg 5	Leg 6
Action 1	C1 - 60°		C3 - 60°		C5 - 60°	
Action 2	A1 - 30°		A3 - 30°		A5 - 30°	
Action 3	C1 - 0°	C2 - 60°	C3 - 0°	C4 - 60°	C5 - 0°	C6 - 60°
Action 4	A1 - 90°		A3 - 90°		A5 - 90°	
Action 5		A2 - 30°		A4 - 30°		A6 - 30°
Action 6	C1 - 60°	C2 - 0°	C3 - 60°	C4 - 0°	C5 - 60°	C6 - 0°
Action 7		A2 - 90°		A4 - 90°		A6 - 90°
Action 8	A1 - 30°		A3 - 30°		A5 - 30°	
Action 9	C1 - 0°	C2 - 60°	C3 - 0°	C4 - 60°	C5 - 0°	C6 - 60°

Table 4.1. Sequence of actions for forward motion.

BACKWARD MOVEMENT

	Leg 1	Leg 2	Leg 3	Leg 4	Leg 5	Leg 6
Action 1	C1 - 60°		C3 - 60°		C5 - 60°	
Action 2	A1 - 150°		A3 - 150°		A5 - 150°	
Action 3	C1 - 0°	C2 - 60°	C3 - 0°	C4 - 60°	C5 - 0°	C6 - 60°
Action 4	A1 - 90°		A3 - 90°		A5 - 90°	
Action 5		A2 - 150°		A4 - 150°		A6 - 150°
Action 6	C1 - 60°	C2 - 0°	C3 - 60°	C4 - 0°	C5 - 60°	C6 - 0°
Action 7		A2 - 90°		A4 - 90°		A6 - 90°
Action 8	A1 - 150°		A3 - 150°		A5 - 150°	
Action 9	C1 - 0°	C2 - 60°	C3 - 0°	C4 - 60°	C5 - 0°	C6 - 60°

Table 4.2. Sequence of actions for backward motion.

RIGHT MOVEMENT

	Leg 1	Leg 2	Leg 3	Leg 4	Leg 5	Leg 6
Action 1	C1 - 60°		C3 - 60°		C5 - 60°	
Action 2	A1 - 30°		A3 - 150°		A5 - 30°	
Action 3	C1 - 0°	C2 - 60°	C3 - 0°	C4 - 60°	C5 - 0°	C6 - 60°
Action 4	A1 - 90°		A3 - 90°		A5 - 90°	
Action 5		A2 - 150°		A4 - 150°		A6 - 30°
Action 6	C1 - 60°	C2 - 0°	C3 - 60°	C4 - 0°	C5 - 60°	C6 - 0°
Action 7		A2 - 90°		A4 - 90°		A6 - 90°
Action 8	A1 - 30°		A3 - 150°		A5 - 30°	
Action 9	C1 - 0°	C2 - 60°	C3 - 0°	C4 - 60°	C5 - 0°	C6 - 60°

Table 4.3. Sequence of actions for right motion.

LEFT MOVEMENT

	Leg 1	Leg 2	Leg 3	Leg 4	Leg 5	Leg 6
Action 1	C1 - 60°		C3 - 60°		C5 - 60°	
Action 2	A1 - 150°		A3 - 30°		A5 - 150°	
Action 3	C1 - 0°	C2 - 60°	C3 - 0°	C4 - 60°	C5 - 0°	C6 - 60°
Action 4	A1 - 90°		A3 - 90°		A5 - 90°	
Action 5		A2 - 30°		A4 - 30°		A6 - 150°
Action 6	C1 - 60°	C2 - 0°	C3 - 60°	C4 - 0°	C5 - 60°	C6 - 0°
Action 7		A2 - 90°		A4 - 90°		A6 - 90°
Action 8	A1 - 150°		A3 - 30°		A5 - 150°	
Action 9	C1 - 0°	C2 - 60°	C3 - 0°	C4 - 60°	C5 - 0°	C6 - 60°

Table 4.4. Sequence of actions for left motion.

4.3.3 UART RECEIVER MODULE FOR MANUAL CONTROL

UART stands for Universal Asynchronous Receiver/Transmitter is a serial communication device that performs parallel to serial data conversion at the transmitter side and serial to parallel data conversion at the receiver side. It is universal because the parameters like transfer speed and data speed are configurable. This is a serial communication device wherein only two wires are required to establish the communication between the two UARTs. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART. The word asynchronous implies that there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits. When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the baud rate.

Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps).Both UARTs must operate at about the same baud rate. The baud rate between the transmitting and receiving UARTs can only differ by about 10% before the timing of bits gets too far off. Both UARTs must also must be configured to transmit and receive the same data packet structure.

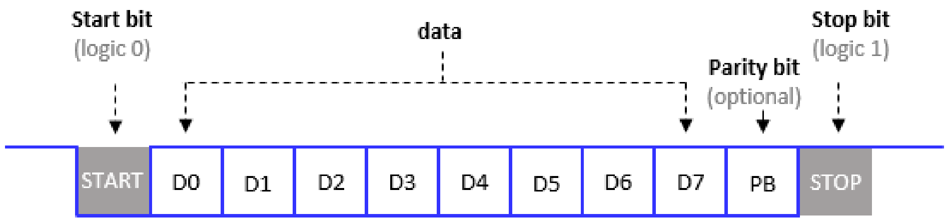


Figure 4.5. Structure of UART data packet/frame.

- Start bit is a synchronisation bit that is added before the actual data. Start bit marks the beginning of the data packet. In order to start the data transfer, the transmitting UART pulls the data line from high voltage level to low voltage level (from 1 to 0). The receiving UART detects this change from high to low on the data line and begins reading the actual data.

- The data frame contains the actual data being transferred. It can be 5 bits up to 8 bits long if a parity bit is used. If no parity bit is used, the data frame can be 9 bits long. The data is sent with the least significant bit (LSB) first.

- Parity describes the evenness or oddness of a number. The parity bit is a way for the receiving UART to tell if any data has been changed during transmission.

- To signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage for at least two bit durations.

ADVANTAGES

- Requires only two wires for full duplex data transmission
- No necessity of the clock signal or any other timing signal.
- Parity bit ensures basic error checking is integrated in to the data packet frame.

LIMITATIONS

- Size of the data in the frame is limited.
- Speed for data transfer is less compared to parallel communication.
- Transmitter and receiver must agree to the rules of transmission and appropriate baud rate must be selected.

IMPLEMENTATION AS HARDWARE MODULE

The primary requirement for the manual control of the bot is the UART Receiver module that receives the packets from the ESP8266 wireless module and these data packets are processed by the FPGA wherein, depending on the received data packets the sequence of motion that are predefined are carried out. So only the UART Receiver module is being implemented here to solve the communication that establishes manual control of the bot.

UART Receiver module here is designed through the state machine based approach that is common for designing complex modules and the various states as per the UART specifications are shown below.

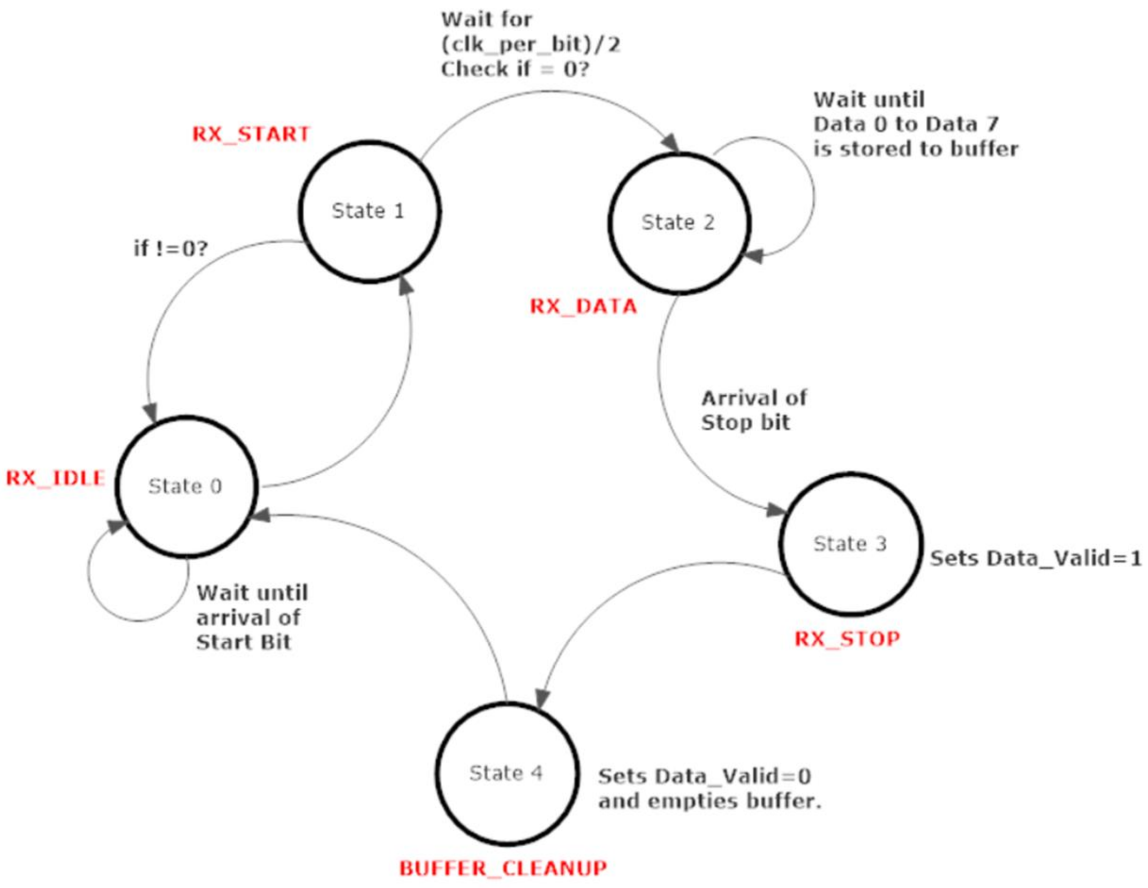


Figure 4.6. UART Receiver state diagram.

Totally it has five different states.

- **State 0:** Idle State: The module stays in this state by default before the start of reception of any data. It continually checks the Rx pin and stays in this state as long as the pin is high. When this pin goes low, it indicates the arrival of the start bit, meaning that a UART data transmission to this module has begun and it should start listening. So it goes to the next state -Start state

- **State 1:** Start state: Since the baud rate is fixed priorly as 115200bps, a clock with that time period is generated. To get reliable and stable data, it is read at the middle portion of data waveform instead of the starting or the ending portion of it. Due to transient effects, the data might be varying at the starting and the end portions, reading it in those portions would lead to an unreliable, wrong data. So it is preferred to read the data in the middle where the transients would have died and the data would have become stable. To sample at the middle portion, in this start state, a counter runs for half of the time period and samples the data. It should still be low for a valid start bit. It then switches to the next state. If it was high, then there was some glitch and rx pin might have become low due to some noise and goes back to idle state.
- **State 2:** Data state: This is state where data is sampled and read. A counter runs for the time period derived from the baud rate set priorly. Once the counter reaches the time period value, it resets. The Rx pin is read and stored in an 8 bit buffer. This repeats for 8 cycles and all the data are read and stored in a buffer. In the ninth cycle it goes to the stop state.

- **State 3:** Stop state: Now the rx pin is checked if it has high value, indicating the stop bit. Now a flag called Rx_Flag is set to indicate the completion of receiving and to indicate that a valid data is present in the buffer. It then goes to the clean-up state.
- **State 4:** Clean-up state: In this state, all the counters are reset. The Rx_Flag goes low. Whenever there is high to low transition on the Rx_Flag, the buffer is read into a variable which could used as a command to the Hexapod. It returns to the idle state after this state, and keeps waiting for the next UART data to receive.

CHAPTER 5
CONNECTIVITY AND CONTROL

5.1 INTRODUCTION

The Hexapod is controlled wirelessly using a joypad interfaced with a laptop. The Laptop and the Hexapod is connected to the same Wi-Fi network. Wi-Fi network was chosen because it is faster and covers long range. UDP/IP protocol is used for communication. ESP-8266, a Wi-Fi SoC built with Tensilica XTensa LX106 CPU core, will host a UDP server at port 1883.

The Laptop will act as a UDP client and send joypad commands to the ESP, which is in turn relayed to the Basys-3 FPGA through UART protocol. The Detailed implementation of these modules are as follows.

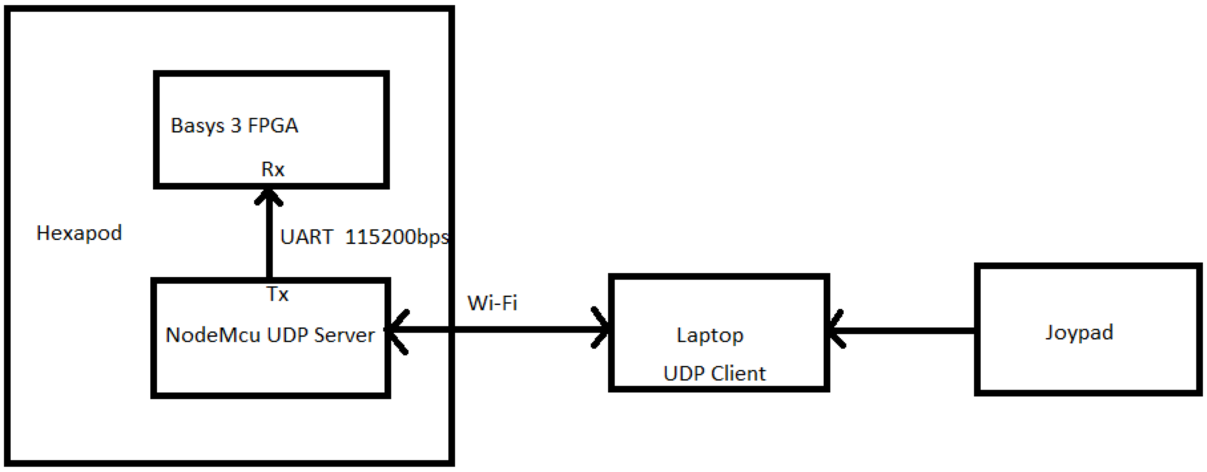


Figure 5.1 Basic Schematic containing the functional blocks.

5.2 THE UDP PROTOCOL:

The User Datagram Protocol (UDP) is a connectionless, light-weight protocol with less overhead used to establish low latency, faster and loss tolerant communication between two application over the internet. Unlike TCP (Transmission Control Protocol), there is no handshake involved between the sender and the receiver before transmitting the data. Hence no connection link is established between the sender and the receiver before the data transmission. Once the server receives a request from a client, it immediately sends the response containing the data.

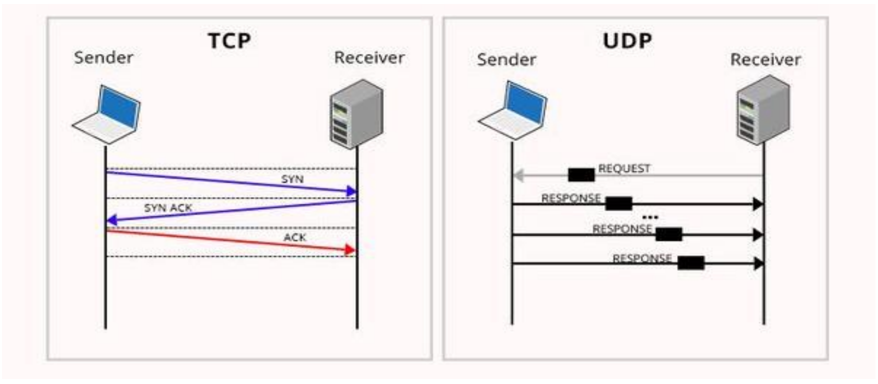
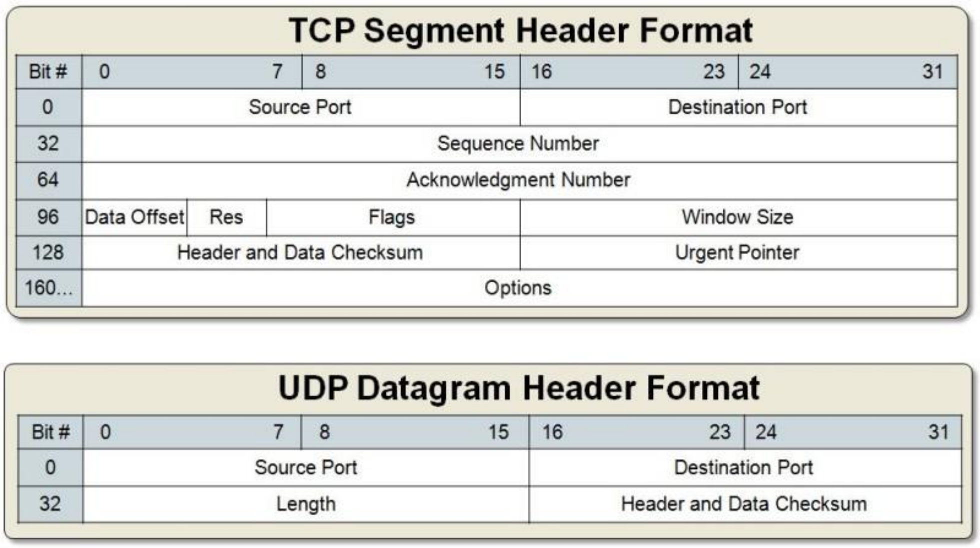


Figure 5.2 TCP vs UDP Comparison.

Since there is no link established prior, there is no guarantee for the data to be received. It might be lost and there is no provision for retransmission of the lost data. Since different datagrams can take different routes, they might be received out of order .Hence UDP doesn’t provide reliable data transmission like TCP. On the other hand, UDP datagram is very less in size compared to that of TCP.

The UDP header consists of source and destination port numbers to indicate the specific source and destination applications in the sending and receiving host machines. It also consists of 16 bits checksum to verify if the data has been corrupted. This leads to a reduced header size of only 8 bytes. TCP header consists of many other fields leading to a bigger packet size, varying from 20 to 60 bytes. It is due to this small packet size, UDP is faster, requires less bandwidth and provides low latency.



TCP Segment Header Format

Bit #	0	7	8	15	16	23	24	31
0	Source Port				Destination Port			
32	Sequence Number							
64	Acknowledgment Number							
96	Data Offset	Res	Flags		Window Size			
128	Header and Data Checksum				Urgent Pointer			
160...	Options							

UDP Datagram Header Format

Bit #	0	7	8	15	16	23	24	31
0	Source Port				Destination Port			
32	Length				Header and Data Checksum			

Figure 5.3 TCP and UDP header format.

5.3 THE TOTAL NETWORK STACK

Transport layer	**UDP**
Network layer	**IP**
Data link layer - LLC MAC	**Wi-Fi -802.11.n (CSMA/CA)**
Physical layer	

This figure defines the network stack of the communication. It consists of four layers. The Fifth layer called application layer is not required. Transport layer, which is the fourth layer has UDP protocol, followed by IP on the network layer. The Wi-Fi standard also called as 802.11.n standard covers the last two layers, the data link layer and physical layer. It uses CSMA/CA which stands for Carrier Sense Multiple Access with Collision Avoidance, to sense and transmit the data through a physical network with many devices and avoids collision of data.

- The joypad data generated by the user is first given to the UDP layer, which adds source and destination port numbers, enabling process to process connection.
- It makes the UDP datagram by adding UDP header to the data from the joypad.
- This UDP datagram is passed to IP, which adds the source and destination IP address of the devices to enable the data to reach the correct devices.
- It makes the IP packet by adding IP header to the UDP datagram received.
- The datagram is the payload of the IP protocol.
- This is then sent to the Wi-Fi stack adds the MAC address of the devices and uses CSMA/CA algorithm to transmit the packet without collision from other packets sent by other devices connected to the same physical network.
- At each layer, the data from the previous layer becomes the payload of this layer.
- It adds ups its own header and sends this packet to the next layer which considers this whole packet as its payload and the packet size grows on increasing as we reach lower layers

			Data
		UDP header	**Data**
	IP header	**UDP header**	**Data**
Wi-Fi header	**IP header**	**UDP header**	**Data**

Figure 5.4 Data Packets

5.4 THE NECESSITY FOR UDP

UDP was preferred over TCP, because

- Our application required faster, low latency and real time data transmission.
- Live video streaming requires high speed transmission and the order of the data is not very important.
- The joypad buttons are read multiple times, same data is generated and transmitted multiple times, ensuring reliable and correct reception of at least one of them.
- The connection is over a local Wi-Fi network rather than the internet, the number of nodes within this network is quite less. Thus, there isn’t many diverge paths for the transmitted data to get lost. Network traffic is also very less.

All these reasons make UDP the best suitable protocol for our application.

5.5 NODEMCU AS UDP SERVER

NodeMCU is an ESP8266 Wi-Fi SoC based development board with inbuilt Wi-Fi connectivity (802.11.b/g/n).It supports 2.4Ghz band. It is based on Tensilica Xtensa Lx106 32-bit CPU core, with 128 KB RAM and 4MB flash memory, operating at 80 MHz clock speed.

UDP server is hosted in NodeMCU at port 1883. This is done with “ESP8266WiFi” and “WiFiUDP” libraries available on Github. It receives data from the laptop and sends it to Basys3 through UART protocol.

Initially, an object, udp, of class WiFiUDP is created. The SSID and password of the WiFi network is passed to WiFi.begin () function to connect the nodemcu with it. The udp server port number, 1883, is passed to the udp.begin () function to setup the server at that port. UART hardware is initialized with 115200 as the baud rate .Its configured to have 1 start bit, 1 stop bit, and no parity bit.

On successful completion of these steps, an UDP server is started and its IP address is printed on a serial port. This IP address is required by the laptop to connect with it as a client.

In an infinite loop, the following steps are performed indefinitely.

- Check if any UDP packet is sent by the client.
- If any, read the data into a buffer.
- Send a dummy reply to the client.
- Send the received buffer to Basys3 through UART.

NodeMCU is used only for wireless connectivity and none of the Hexapod servos are controlled by it. It only forwards the commands sent from the laptop. All hexapod functions are done by Basys3 FPGA board only.

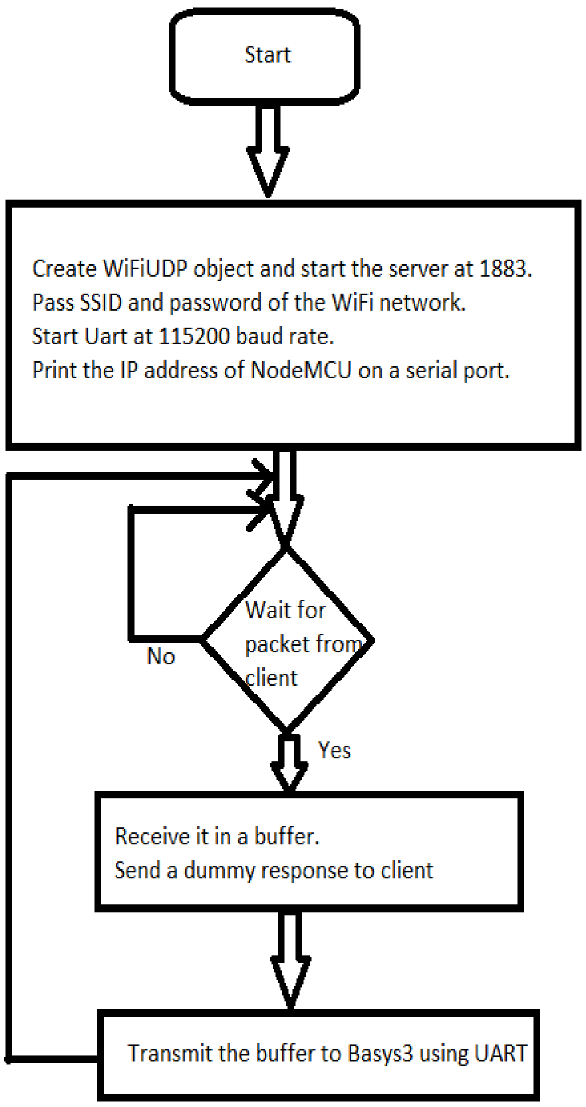

Start
Create WiFiUDP object and start the server at 1883.
Pass SSID and password of the WiFi network.
Start Uart at 115200 baud rate.
Print the IP address of NodeMCU on a serial port.
Wait for packet from client
No
Yes
Receive it in a buffer.
Send a dummy response to client
Transmit the buffer to Basys3 using UART

Figure 5.5 Flow diagram for the UDP Server

5.6 JOYPAD INTERFACING AND UDP CLIENT

A python script running on a laptop, is used to interface the Joypad and send the data to the Nodemcu Udp server on the Hexapod. For interfacing the Joypad, Pygame module and to send data through UDP, sockets module are used respectively. The Dual-shock joypad consists of 4 axes, 10 buttons, and one hat inputs.

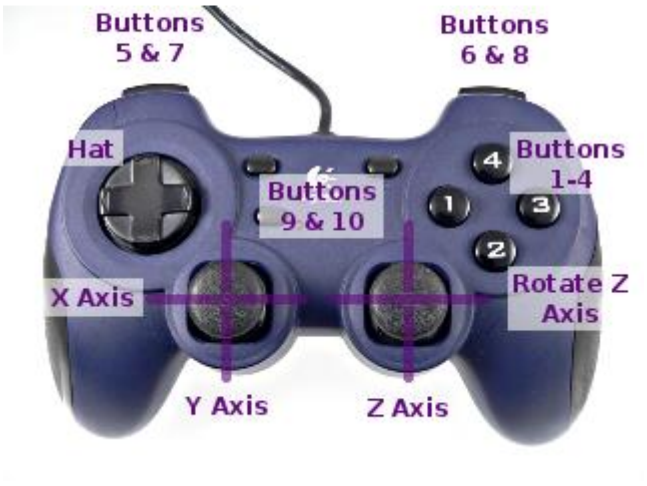


Figure 5.6 Joypad input mapping.

This joystick generates five events namely JOYAXISMOTION, JOYBALLMOTION, JOYBUTTONDOWN, JOYBUTTONUP and JOYHATMOTION. Every input triggers a corresponding event which gets stored in a FIFO buffer in the order they were generated. Data is read from this FIFO buffer which ensures that inputs are read in the order they were given by the user.

The Python script does the following:

- Initialize the joystick module of Pygame package.
- Create an object of joystick class.

- Each joystick object represents each joystick connected to the computer.
- Initialise the created joystick object.
- The IP address and the port number of the UDP server is provided to a variable.
- The sockets module is initiated to set up udp communications.
- In the infinite loop, the events buffer is updated.
- The x and Y axis values are read using Joystick.get_axes () function.
- Each axis gives 0 at idle position, value within 0 to -1 range when pushed upwards, value within 0 to 1 range when pushed downwards.
- If x axis is 0 and Y axis is less than -0.5, the command is forward.
- If x axis is 0 and y axis is greater than 0.5, the command is backwards.
- Similarly if Y axis is 0 and x axis is less than -0.5, the command is left and vice-versa.
- If both the axes are 0, the command is stop.
- According to the values of the axes Forward, Back, Right, Left or Stop command is set.
- Finally, an UDP packet addressing the UDP server on the NodeMCU is sent with this command.

CHAPTER 6

LIVE VIDEO TRANSMISSION USING ESP-32 CAM

6.1 INTRODUCTION:

One of the key features of the hexapod is the ability to transmit live video feed. And this is achieved by using the ESP32 CAM module. By using this camera module we are able to view the live video transmission from the hexapod in a browser. This feature allows us to get good quality video feed from the hexapod point of view and this helps us to easily understand nature of the surrounding in which the hexapod is being operated. Another use is that it is easy for us to locate obstacles in places when we are not able to see it in person. It is also used to take photos through the OV640 camera. Let us discuss more about the ESP32 CAM module in this chapter.

The ESP32 CAM Module with OV2640 Camera has a very competitive small-size camera module that can operate independently as a minimum system with a footprint of only 40 x 27 mm. It is widely used in various IoT applications. This module adopts a DIP package It is suitable for home smart devices, industrial wireless control, wireless monitoring, and other IoT applications. This is a very small camera module with the ESP32-S chip. Besides the OV2640 camera, and several GPIOs to connect peripherals, it also features a microSD card slot that can be useful to store images taken with the camera.

6.2 FEATURES OF THE MODULE

Here is a list of the ESP32-CAM features

- The smallest 802.11b/g/n Wi-Fi BT SoC module
- Low power 32-bit CPU,can also serve the application processor
- Up to 160MHz clock speed, summary computing power up to 600 DMIPS

- Built-in 520 KB SRAM, external 4MPSRAM
- Supports UART/SPI/I2C/PWM/ADC/DAC
- Support OV2640 and OV7670 cameras, built-in flash lamp
- Support image Wi-Fi upload
- Support TF card
- Supports multiple sleep modes
- Embedded Lwip and FreeRTOS
- Supports STA/AP/STA+AP operation mode
- Support Smart Config/AirKiss technology
- Support for serial port local and remote firmware upgrades (FOTA)

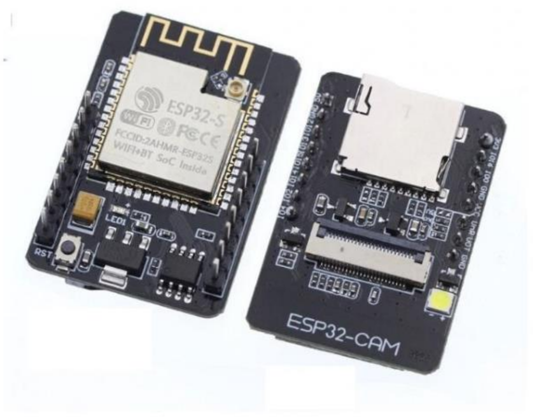


Figure 6.1 ESP-32 CAM module.

This module constitutes an OV2640 camera with the following features

- Can be used in MCU, Raspberry Pi, ARM, DSP, FPGA platforms
- 2 megapixels image sensor OV2640
- M12 mount or CS mount lens holder with changeable lens options
- Build-in 650nm IR block filter, visible light only
- Array size: UXGA 1622X1200

- Power supply: 3.3V
- IO voltage level: 1.7V~3.3V DC
- Output formats:
  - YUV(422/420)/YCnCr422
  - RGB565/555
  - 8-bit compressed data
- Max image transfer rate:
  - UXGA/SXGA 15fps
  - UXGA/SXGA 30fps
  - SVGA 30fps
  - CIF 60fps

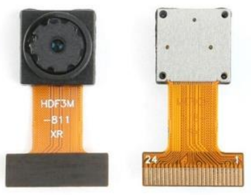


Figure 6.2 OV2640 camera

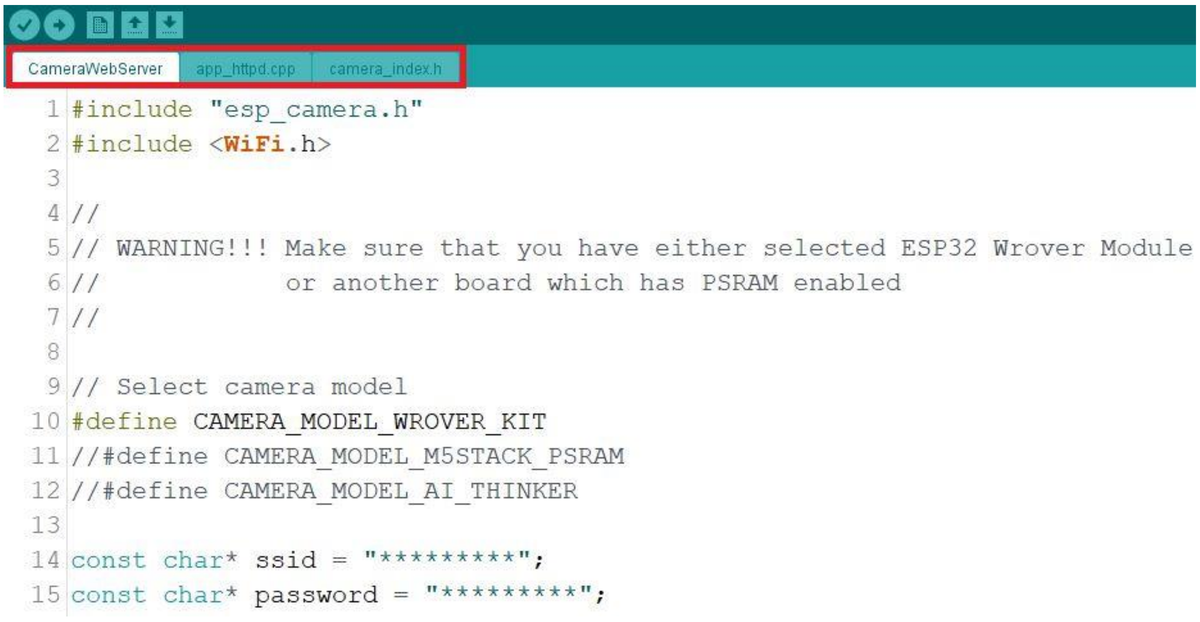
6.3 GENERAL SPECIFICATIONS OF THE MODULE

- Wireless Module: ESP32-S Wi-Fi 802.11 b/g/n + Bluetooth 4.2 LE module with PCB antenna, u.FL connector, 32Mbit SPI flash, 4MBit PSRAM.
- External Storage: micro SD card slot up to 4GB.
- Camera
  - FPC connector.
  - Support for OV2640 (sold with a board) or OV7670 cameras.
  - Image Format: JPEG (OV2640 support only), BMP, grayscale.
  - LED flashlight.
- Expansion: 16x through-holes with UART, SPI, I2C, PWM.

- Reset button.
- Power Supply: 5V via pin header.
- Dimensions (ESP32): 40 x 27 x 12 (LxWxH) mm.
- Here is a list with the OV2640 specifications:
- Active array size: 1600×1200
- Shutter: rolling shutter
- Lens: 1/4 inch
- SPI speed: 8MHz
- Frame buffer Size: 384KB

6.4 VIDEO STREAMING WITH ESP-32:

On initialization of the module the first step is to connect this module to a local host network and this is enabled by setting up the network credentials of a remote access point and on resetting the ESP-32 module gets connected in this network automatically.



```
#include "esp_camera.h"
#include <WiFi.h>

//
// WARNING!!! Make sure that you have either selected ESP32 Wrover Module
//            or another board which has PSRAM enabled
//

// Select camera model
#define CAMERA_MODEL_WROVER_KIT
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_AI_THINKER

const char* ssid = "*********";
const char* password = "*********";
```

Figure 6.3 Entry of network credentials.

Before uploading the code, you need to enter your network credentials in the following variables according to the remote access point that is available.

- const char* ssid = "REPLACE_WITH_YOUR_SSID";
- const char* password = "REPLACE_WITH_YOUR_PASSWORD";

Now, you can upload the code to your ESP32-CAM board.

PROGRAMMING THE MODULE

The ESP-32 CAM module is programmed through either a FTDI programming cable. This FTDI module acts as an adapter that facilitates the USB to TTL logic conversion as the ESP module supports a 5V TTL logic this conversion becomes necessary.

Many FTDI programmers have a jumper that allows you to select 3.3V or 5V and for this ESP module the 5V TTL logic is to be used.

- GPIO 0 needs to be connected to GND so that you're able to upload code.

ESP32-CAM	**FTDI Programmer**
GND	GND
5V	VCC (5V)
U0R	TX
U0T	RX
GPIO 0	GND

- Go to **Tools** > **Board** and select **ESP32-CAM**.
- Go to **Tools** > **Port** and select the COM port the ESP32 is connected to.
- Then, click the upload button to upload the code.
- After the code has been successfully uploaded the RST button is pressed to enable the ESP32 module to connect to the network.

CHAPTER 7

SIMULATION AND RESULTS

7.1 SIMULATION USING THE VIVADO HLS SUITE

The necessary hardware modules that were designed are simulated using the Vivado Simulator that is in-built within the software VIVADO HLS Design Suite 2017.4, produced by the Xilinx for the synthesis and analysis of HDL designs. This software provides the complete FPGA design flow in a user friendly way. Starting from the design of the hardware modules using the RTL codes to the generation of the bit stream file the flow is well defined by this software.

Generation of the Pulse Width Modulates (PWM) signal which a periodic signal having a time period (Tp) = 20ms, necessary for the actuation of the servo motors is designed as a hardware module and is simulated and the output is as shown in Figure 7.1.

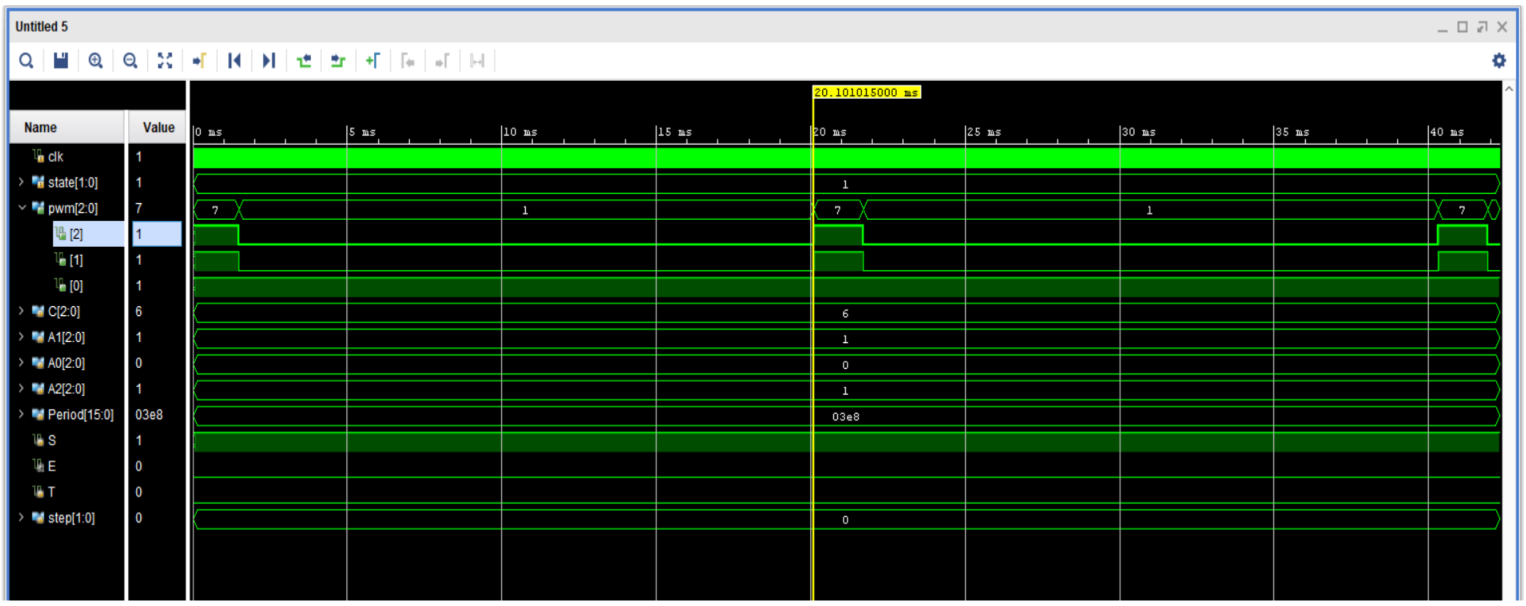
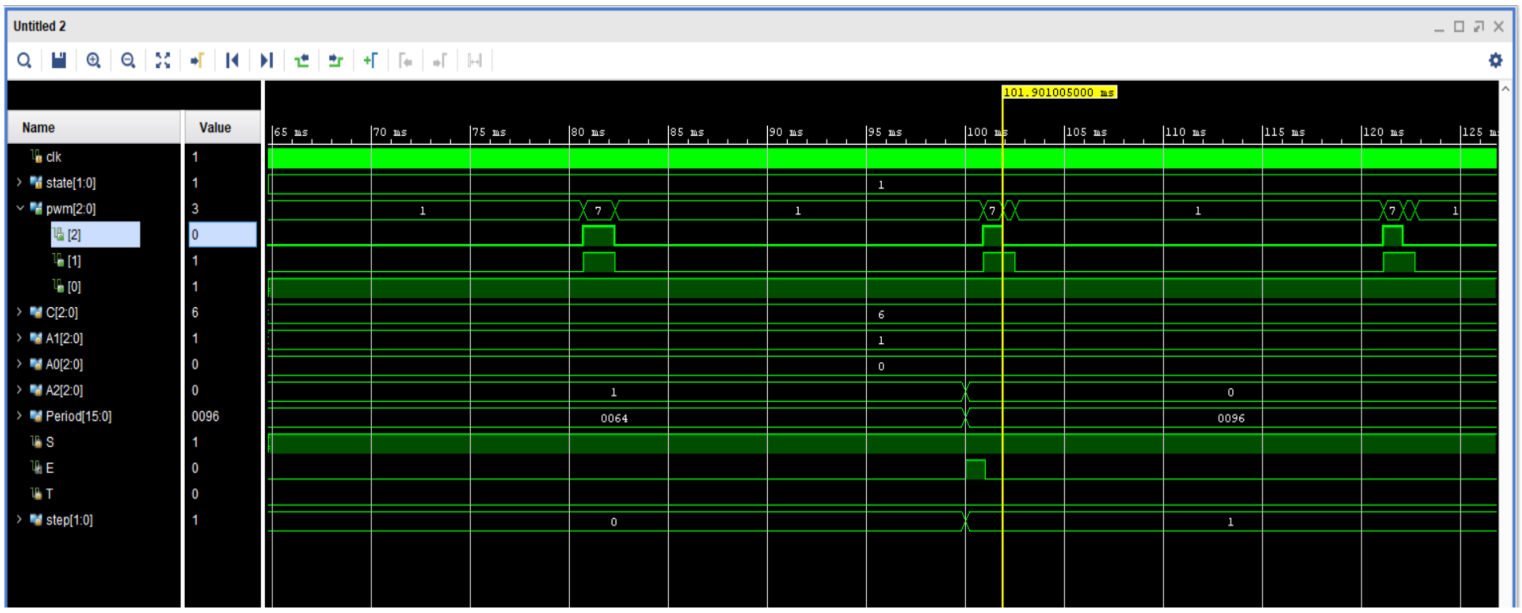


Figure 7.1 PWM signal generation output.

Following the PWM signal output the movement of the hexapod is facilitated through an ordered sequence of pulses given to actuate the servo motors. This sequences of pulses are basically composed of the PWM signals generated prior along with a controllable duty cycle that alters the ON-OFF duration of the PWM signals.

Generation of this sequence is simulated and the obtained output is as shown in Figure 7.2.

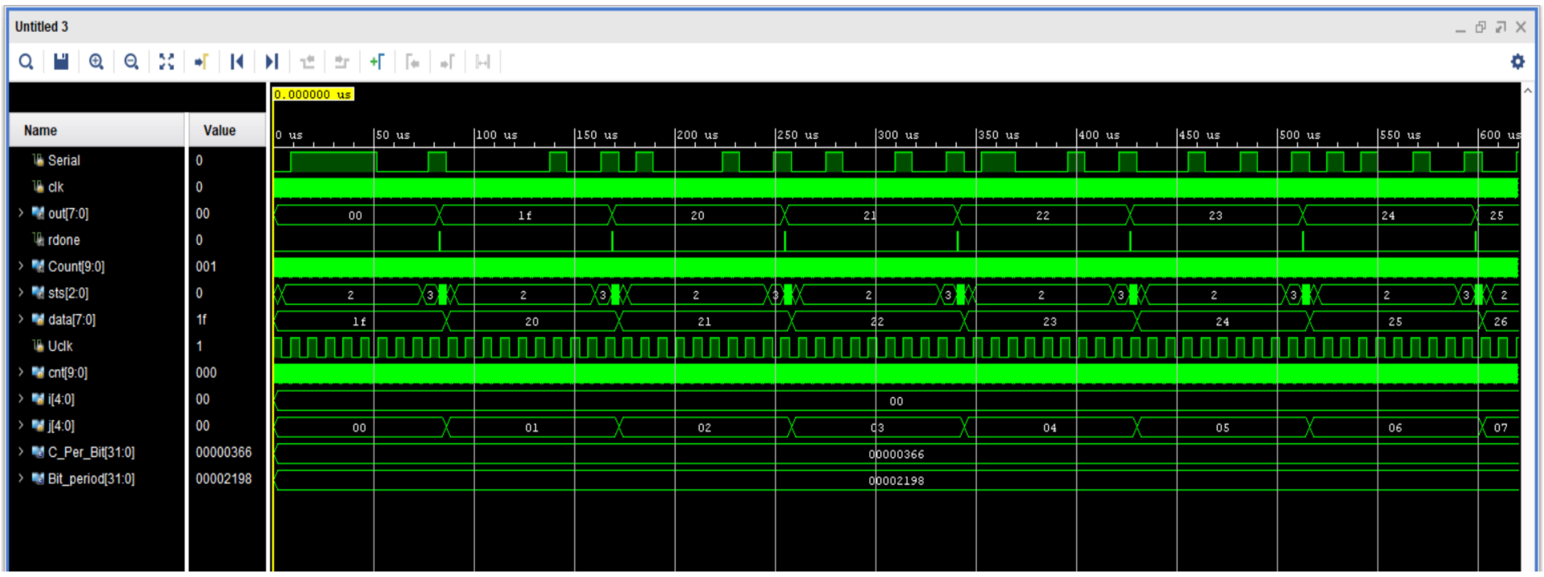

Untitled 2

Name	Value
clk	1
state[1:0]	1
pwm[2:0]	3
[2]	0
[1]	1
[0]	1
C[2:0]	6
A1[2:0]	1
A0[2:0]	0
A2[2:0]	0
Period[15:0]	0096
S	1
E	0
T	0
step[1:0]	1

Figure 7.2 Sequence generation output for motor control.

The hexapod movement is established through a manual wireless control using a joystick for which there is a necessity for the BASYS3 FPGA to interface with the ESP8266 module. This interface is established through a Universal Asynchronous Receiver/Transmitter. Only the receiver module is required and is designed as a state machine.

UART Receiver module implementation is simulated and the obtained output is as shown in Figure 7.3.



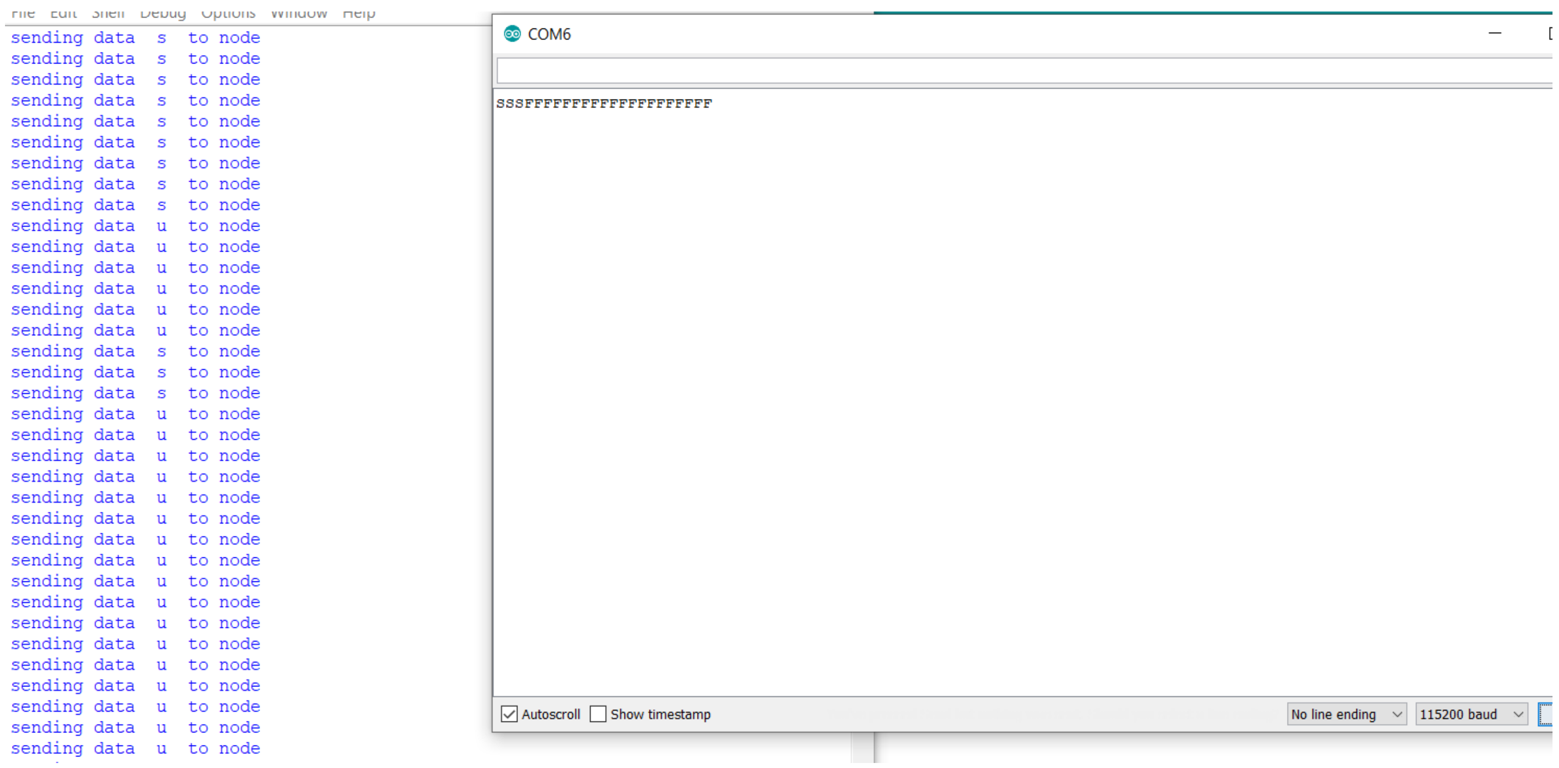
Name	Value
Serial	0
clk	0
out[7:0]	00
rdone	0
Count[9:0]	001
sts[2:0]	0
data[7:0]	1f
Uclk	1
cnt[9:0]	000
i[4:0]	00
j[4:0]	00
C_Per_Bit[31:0]	00000366
Bit_period[31:0]	00002198

Figure 7.3 UART Receiver module output.

In VIVADO all the above simulations are run by giving inputs through a module that is referred to as the testbench. The testbench is again a verilog module that gives the stimulus input to the Design Under Test (DUT) and is generally used for the simulation while designing hardware modules to ensure that the hardware modules’ functionality remains the same for various stimulus. The only major difference is that these testbench modules are not synthesizable and cannot be implemented onto the FPGA and can only be employed for software simulation.

7.2 UDP SERVER IMPLEMENTATION

The UDP server is hosted in NodeMcu board which is fixed in the Hexapod. The laptop is connected to the joypad and acts as a UDP client. A python script gets the value from the joypad and sends it to nodemcu through UDP. Here, the window on the left shows the value sent by the laptop when no button was pressed (Stop command letter: s) and forward button was pressed (forward command letter: u). On the right, the serial monitor of the nodemcu displays the corresponding command it sends to the FPGA through UART, when it received the corresponding data from laptop.



```
sending data  s  to node
sending data  s  to node
sending data  s  to node
sending data  s  to node
sending data  s  to node
sending data  s  to node
sending data  s  to node
sending data  s  to node
sending data  s  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  s  to node
sending data  s  to node
sending data  s  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
sending data  u  to node
```

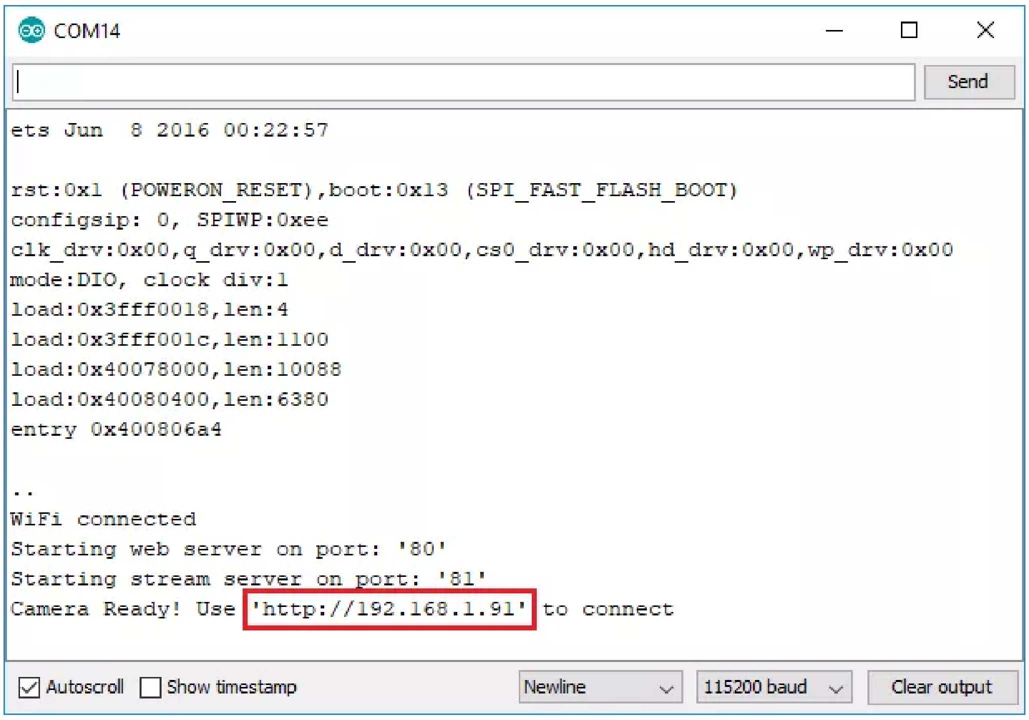
```
SSSFFFFFFFFFFFFFFFFFFFF
```

Figure 7.4 Visualization of transmitted data packets.

7.3 LIVE VIDEO TRANSMISSION

Live video transmission is also employed in our hexapod design making use of the ESP-32 CAM module that facilitates a wireless transmission of the image frames. A web server is set-up to which this ESP-32 module is connected and establishes a live video feed through the browser.

To set-up the Camera Web Server we must enable a remote access point and give in the network credentials on connection ESP-32 cam displays the IP address on the COM port that is used to connect through the browser as shown in the Figure 6.5.



```
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1100
load:0x40078000,len:10088
load:0x40080400,len:6380
entry 0x400806a4

..
WiFi connected
Starting web server on port: '80'
Starting stream server on port: '81'
Camera Ready! Use 'http://192.168.1.91' to connect
```

Figure 7.5 Serial monitor output displaying IP address.

The ESP-32 CAM module is now connected and active and the live video feed can be enabled using the IP address in the browser. The video transmission output is obtained and along with additional features that could be modified as per the requirement is as shown in Figure 6.6 and 6.7 respectively.

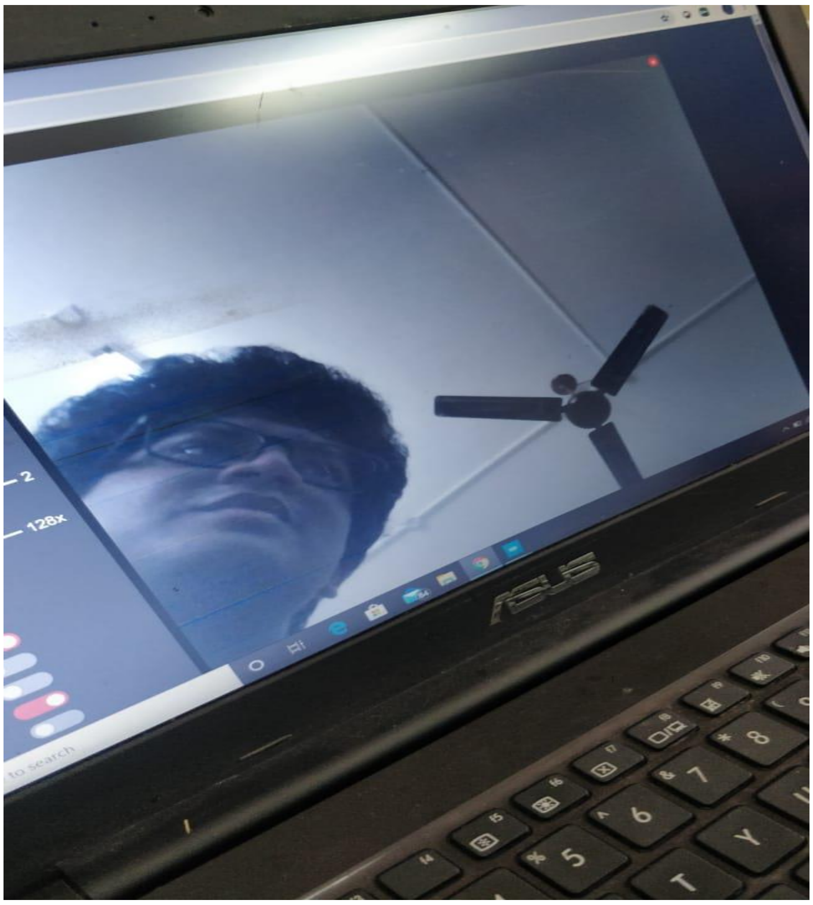
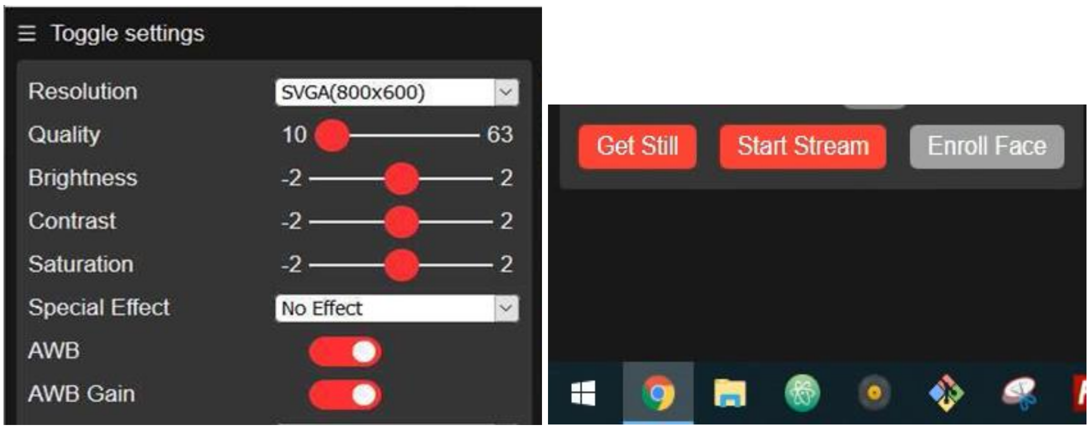


Figure 7.6 Live Video Transmission running in browser.



☰ Toggle settings

Resolution	SVGA(800x600)
Quality	10 — 63
Brightness	-2 — 2
Contrast	-2 — 2
Saturation	-2 — 2
Special Effect	No Effect
AWB	●
AWB Gain	●

Figure 7.7 Additional features for video feed modification.

CHAPTER 8

CONCLUSION

The project has comprehensively dealt with every aspect in designing of the Hexapod and the element of uniqueness is introduced through a Field Programmable Gate Array (FPGA) board based approach that has offered us several advantages in comparison to the conventional microcontroller based approach. This FPGA based design using a wireless manual control is the first of its kind, displaying the extent of capability possible through the use of an FPGA. Another important aspect of the proposed design is its capability of self-recovery that is made possible through mindful choice of the design that could facilitate this distinctive feature. The design is then completed with an additional feature of live video feed transmission to the user in a wireless setup allowing a wide range of operability of the Hexapod.

The high level of modularity and adaptability of the FPGA has given the design freedom to the user for the inclusion of several other features to the Hexapod. Currently, our Hexapod design is capable of movements if all directions through manual control, a fall-recovery capability that enables the bot by itself to recover in the event of a topple and a wireless live video stream for a visual feedback to the user.

We have demonstrated the movement of the bot through manual control along with the live video streaming running in the browser. These features are implemented as hardware modules in the FPGA through Hardware Description Language (HDL) synthesis.

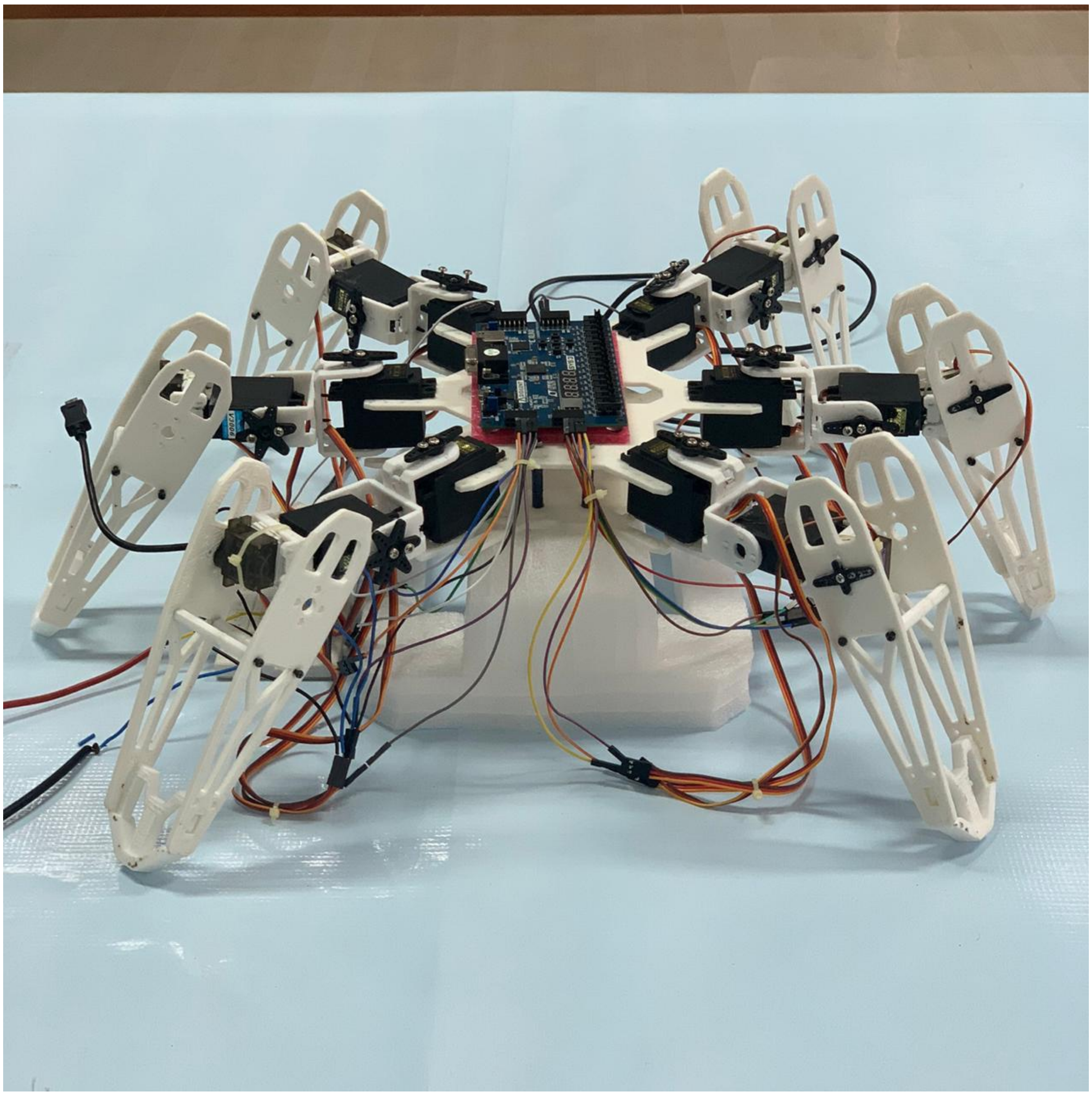


Fig 8.1 The Hexapod

CHAPTER 9

FUTURE SCOPE

Moving further, the following developments can be made to enhance the robot.

- Automated Waypoint Navigation: The bot can be made to move around autonomously in a predefined path by the user. This will be very much useful to survey and collect information about a hazardous site .The onboard camera could provide live feed of the surveying environment.
- A new user button to take a picture of the surrounding and store it in an onboard memory card or transmit wirelessly then and then.

REFERENCES

[1] Ali, N., Salim, S., Abd, R., Rosman, Anas, S., Noh, Z.M. and Samsudin, S. (2013) ‘PWM controller design of a hexapod robot using FPGA’, pp.310-314, 10.1109/ICCSCE.2013.6719980.

[2] Banjanovic-Mehmedovic, L., Mujkic, A., Babic, N., and Secic, J. (2020) ‘Hexapod Robot Navigation Using FPGA Based Controller’, 10.1007/978-3-030-18072-0_5.

[3] Hendricks, (2014), ‘PhantomX Hexapod Mark II CAD files’, viewed November 2019, [https://grabcad.com/library/hexapod-crawler-robot-1].

[4] Kumar, S. (2020). A mechanism to overcome the toppling problem of a hexapod. 10.1007/s41315-020-00126-3.

[5] Tamboli, M. (2018) ‘ESP32+OV7670 — WebSocket Video Camera’, viewed January 2020, [https://medium.com/@mudassar.tamboli/esp32-ov7670-websocket-video-camera-26c35aedcc64].

[6] Taylor, A. (2018) ‘Hexapod Robot: FPGA-Based Solution’, viewed November 2019, [https://www.hackster.io/adam-taylor/hexapod-robot-fpga-based-solution-2c41b5].