FLYPORT DEVELOPMENT MANUAL

Version *1.2*

12/19/2013

Version History

Version Number	Implemented By	Revision Date	Approved By	Approval Date	Description of Change
1.0	Sneihil Gopal		Amarjeet Singh		
1.1	Nishant Sharma Mayank Garg	07/23/13	Amarjeet Singh		P2P Page State Saving, Optimization implemented, Buffer Size fixed
1.2	Nishant Sharma	12/19/13	Amarjeet Singh		Added Fyport Ethernet Module over UART

TABLE OF CONTENTS

1 INTRODUCTION

1.1 PURPOSE OF THE DOCUMENT

This manual not only helps a newbie to understand the Flyport from scratch, but also helps a professional to understand the advanced stuff implemented on it.
This is a detailed manual for all the three modules (Ethernet, Wi-Fi, GPRS) illustrating both the hardware part and the software part of all.
The manual was continuously updated during the development cycle based on the particular circumstances of the Project and on the methodology used for developing the Flyport.

2 GENERAL OVERVIEW

This section contains the overview of Flyport, it's different types, and services. It also mention the necessary software required by a beginner to setup the complete environment to start development on Flyport modules.

2.1 ABOUT THE FLYPORT

FLYPORT is an open hardware programmable system-on-module available in 3 versions*:* Wi-Fi, Ethernet and GPRS**.**

FLYPORT Wi-Fi is a compact module designed around the Microchip PIC 24FJ256GA106 processor (256K Flash, 16K Ram, 16 Mips) and MRF24WB0MA/RM WI-FI Certified Transceiver.

FLYPORT is not simply a serial to Wi-Fi solution, but a smart module with no need of an external host processor as it combines the processor power with the connectivity transceiver.

FLYPORT Ethernet is a wired LAN device embedding a Microchip PIC24FJ microcontroller and an Ethernet transceiver. Plus the same 26 pin connector of Wi-Fi version, this module has a second 26 pin connector with the RJ45 signals and some I/Os.

FLYPORT GPRS quad band (850/900/1800/1900MHz) is a microcontroller programmable system-on-module with integrated cellular GPRS connectivity and SIM card holder.

Flyport provides the easiest way to connect sensors/devices and more to the Internet.
It is able to handle voice calls, SMS, email, FTP client, TCP client and HTTP client.

A wide range of applications can be easily developed and run on FLYPORT with openPicus IDE. No Wi-Fi expertise is needed.

The IDE allows focusing on application as the openPicus framework handles the rest for you. It is based on freeRTOS, manages the Wireless stack and its events.

FLYPORT provides the following services:

Web server, TCP Socket, UDP Socket, SMTP Client, SNTP Client

2.2 GETTING STARTED

To setup the complete environment to start development on the Flyport Modules download the following software:

- C30 Microchip Compiler
- FLYPORT IDE

3 ARCHITECTURE DESIGN

This section outlines the software and hardware architecture design of the Flyport.

3.1 SCHEMAS FOR FLYBOARD

For schemas of Flyboard, go to schemas.

3.2 HARDWARE ARCHITECTURE

Flyport consists of following hardware:

- **Openpicus Flyport Module**

  Microcontroller: Microchip PIC24FJ256 16bit.

  Transceiver: Wi-Fi, Ethernet, GPRS

- **Flyport Baseboard (Custom Made)**

  Ambient Light Sensor APDS-9300 having I2C Interface.

  Ambient Temperature Sensor DS-18S20 / DS-18B20 having 1 wire Interface.

  PIR Motion Sensor (Digital O/P)

  Interface's Available: Power(5V/3V), UART, ADC, SPI, GPIO

The Baseboard for the entire three Module remains same, however since GPRS Module require extra Power, it cannot work over UART power and thus require a Power source of 3V-5V unlike Wi-Fi Module or Ethernet Module.

SENSOR SPECIFICATION :

1. **Ambient Temperature Sensor DS18S20/DS18B20 :** The DS1820 Digital Thermometer provides 9–bit temperature readings which indicate the temperature of the device. Information is sent to/from the DS1820 over a 1–Wire interface, so that only one wire (and ground) needs to be connected from a central microprocessor to a DS1820. Power for reading, writing, and performing temperature conversions can be derived from the data line itself with no need for an external power source.
2. **Ambient Light Sensor APDS-9300 :** The APDS-9301 is Light-to-Digital Ambient Light Photo Sensor that converts light intensity to digital signal output capable of direct I2C interface. Each device consists of one broadband photodiode (visible plus infrared) and one infrared photodiode. Two integrating ADCs convert the photodiode currents to a

digital output that represents the irradiance measured on each channel. This digital output can be input to a microprocessor where illuminance (ambient light level) in lux is derived using an empirical formula to approximate the human-eye response.

3. **PIR Motion Sensor:** PIR sensors allow you to sense motion, almost always used to detect whether a human has moved in or out of the sensors range. We use a digital PIR sensor which gives an output 1 if the motion is detected and 0 otherwise.

For more details please see **Device Specifications**.

3.3 HARDWARE CONFIGURATION

FIRMWARE ON IIIT-DWSN NODE :

The FIRMWARE on the IIIT-D WSN node is simple and easy to understand. It features an embedded Web Server that can be modified and used for a number of purposes:
• Sending parameters to the FLYPORT module at runtime.
• Accessing the I/O ports at runtime.
• Reading analog and digital sensor values in real time.
• Connecting to a Wireless Network in infrastructure mode (allowing users to set SSID and access credentials through their browser)

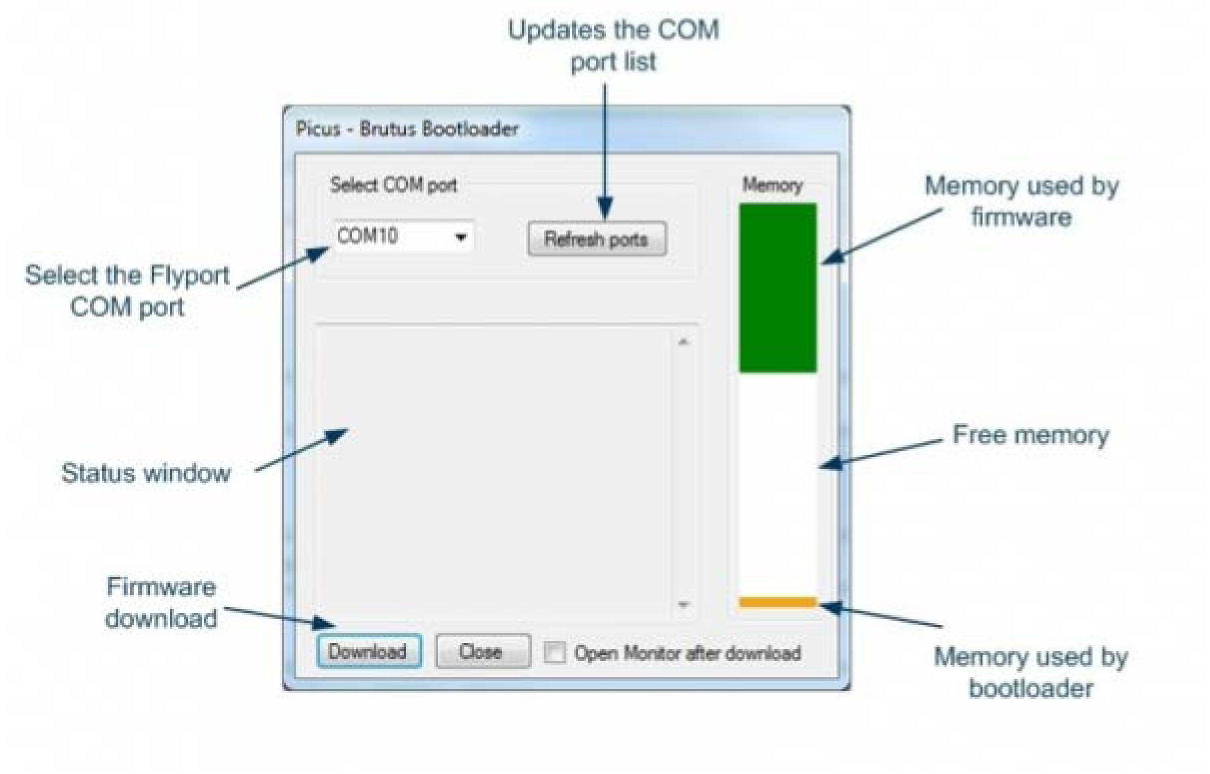
LOADING THE FIRMWARE :

The Firmware customized for IIIT-D WSN node can be check out at
CODE FOR IIIT-DWSN NODE.

- Install Tortoise SVN client to checkout latest code from https://iiitd-flyport-development.googlecode.com/svn/trunk/
- Correspondingly two folders named **“*Wi-Fi*”** (code folder for FLYPORT Wi-Fi) and **“*Eth*”** (code folder for FLYPORT ETHERNET) will be checked out. You can check out the read-only version since you are not required to commit back the code.
- Now follow the steps mentioned at **Load Firmware into IIIT-DWSN Node**. (**Note:** Please skip the first two steps as you have already checkout the code from the Google Code repository as described above).

SERIAL BOOTLOADER

Each Flyport module embeds a serial Bootloader (a customized version of the ds30 loader) that enables the user to download the firmware simply using the serial port, instead a programmer. This makes everything easier and cheaper. The download tool integrated in the IDE communicates with the device and downloads the firmware directly on the Flyport, simply selecting the serial com port and clicking “download”. On the right is shown firmware size (green bar), free rom memory (white bar) and bootloader size (orange bar, at the end of the program memory).


Updates the COM port list
Picus - Brutus Bootloader
Select COM port
COM10
Refresh ports
Memory
Memory used by firmware
Select the Flyport COM port
Free memory
Status window
Firmware download
Download
Close
Open Monitor after download
Memory used by bootloader

USING FLYPORT IDE

- IMPORT WEB PAGES (NOT VALID FOR FLYPORT GPRS)

IDE includes a tool to automatically import new webpages inside the project. Importing webpages doesn't mean only importing the HTML code, but also images, sounds and dynamic code (like JavaScript). Using **Flyport Ethernet** and **Flyport Wi-Fi G** there is the possibility to choose between internal flash memory (consumes microcontroller's ROM memory), or external one (uses onboard 16Mbit Flash memory).

To start the process, just click on the button Web Pages, and the import window will appear.

In the new form, you must select the folder in which are located all the webpages files (HTML, images, media, javascript...). Inside the folder you can order the data as you like, also with other subfolders. All the files located in the selected folder and subfolders will be included inside the project.

IMPORTANT: the WebPages of the webserver not only contain static data, but also dynamic contents that interacts with the firmware. The integration with the firmware is managed by the file HTTPApp.c. So when the webpages are changed, also the code in HTTPApp.c must be changed (if there are differences with the previous dynamic contents), otherwise, compiling errors can arise.

- **IDE WIZARD**

IDE wizard is a tool to properly configure some Hardware and Firmware parameters. For Flyport Ethernet and Wi-Fi it permits also to configure the TCP/IP stack, the service available on the device, and the network configuration (like IP address, SSID, security...). The network configuration is needed to make the Flyport run in the right way, and connect to the desired network, the selection of the services is needed to minimize the resources used by the TCP/IP stack. IDE wizard changes with the model of Flyport module you choosen.

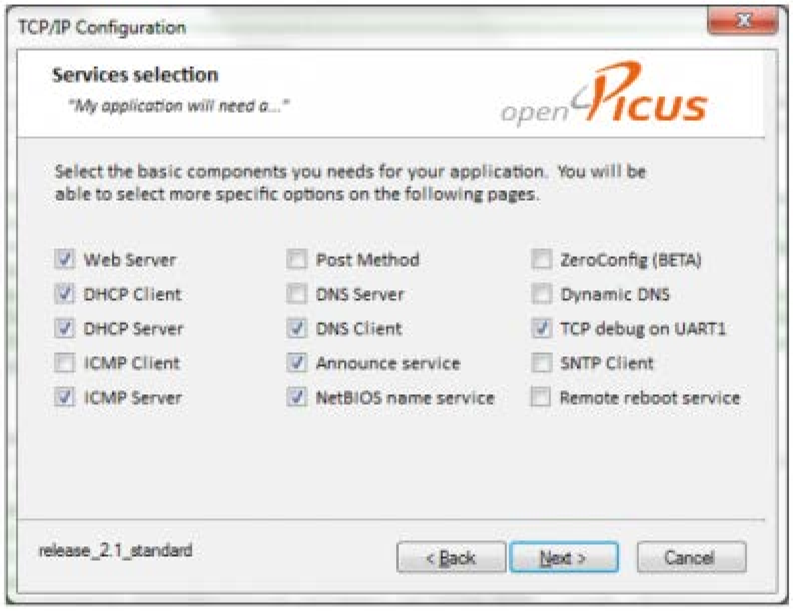
Two different wizards are generated if you are using

1. **Flyport Wi-Fi and Ethernet**
2. **Flyport GPRS**

FLYPORT WI-FI AND ETHERNET

➢ Select Services

This is the first page of the Wizard. Here it is possible to select the service to include inside the Flyport. In this way it is possible to reduce at the minimum the size of the code for the TCP stack according to your needing.

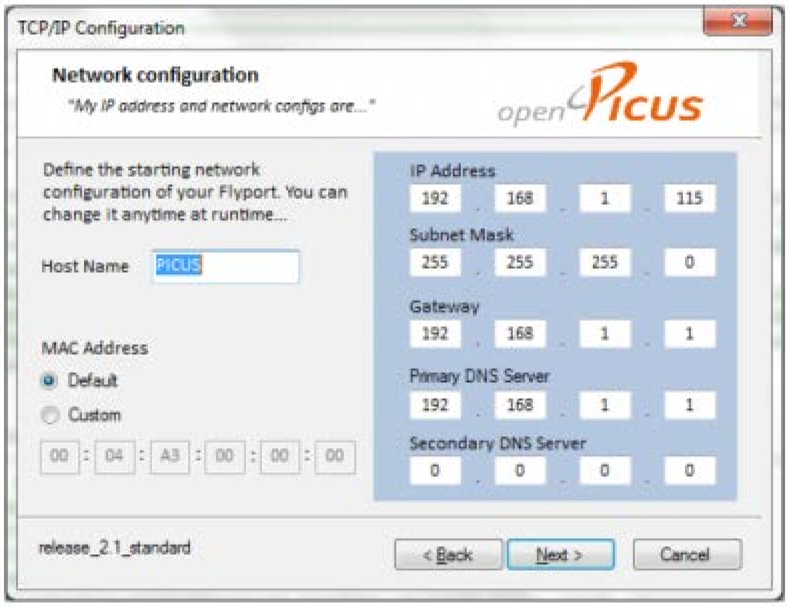


Select the basic components you needs for your application. You will be able to select more specific options on the following pages.

✓ Web Server	Post Method	ZeroConfig (BETA)
✓ DHCP Client	DNS Server	Dynamic DNS
✓ DHCP Server	✓ DNS Client	✓ TCP debug on UART1
ICMP Client	✓ Announce service	SNTP Client
✓ ICMP Server	✓ NetBIOS name service	Remote reboot service

➢ Network Configuration

Next, comes the Network Configuration screen. Here you can configure options like Host Name, MAC Address, IP Address, Subnet Mask, Gateway and DNS Server.



Network configuration

"My IP address and network configs are..."

Define the starting network configuration of your Flyport. You can change it anytime at runtime...

Host Name: PICUS

MAC Address
- [✓ Default]
- Custom

00 : 04 : A3 : 00 : 00 : 00

IP Address: 192 . 168 . 1 . 115

Subnet Mask: 255 . 255 . 255 . 0

Gateway: 192 . 168 . 1 . 1

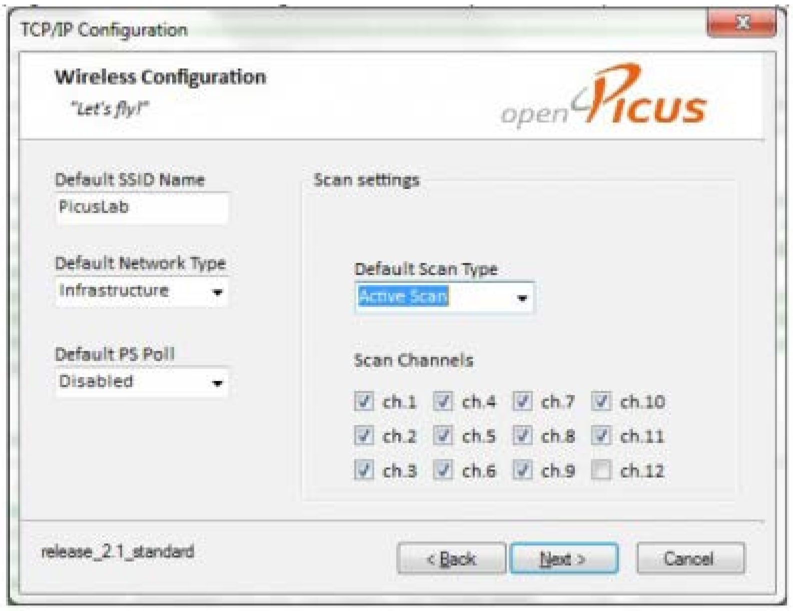
Primary DNS Server: 192 . 168 . 1 . 1

Secondary DNS Server: 0 . 0 . 0 . 0

release_2.1_standard

➢ **Wireless Network Configuration** (Only for Wi-Fi projects!)

Through this page you can configure wireless parameters such as SSID, Default Network type, Scan channels etc.



Wireless Configuration

"Let's fly!"

Default SSID Name: PicusLab

Default Network Type: Infrastructure

Default PS Poll: Disabled

Scan settings

Default Scan Type: Active Scan

Scan Channels

✓ ch.1 ✓ ch.4 ✓ ch.7 ✓ ch.10
✓ ch.2 ✓ ch.5 ✓ ch.8 ✓ ch.11
✓ ch.3 ✓ ch.6 ✓ ch.9 ☐ ch.12

release_2.1_standard

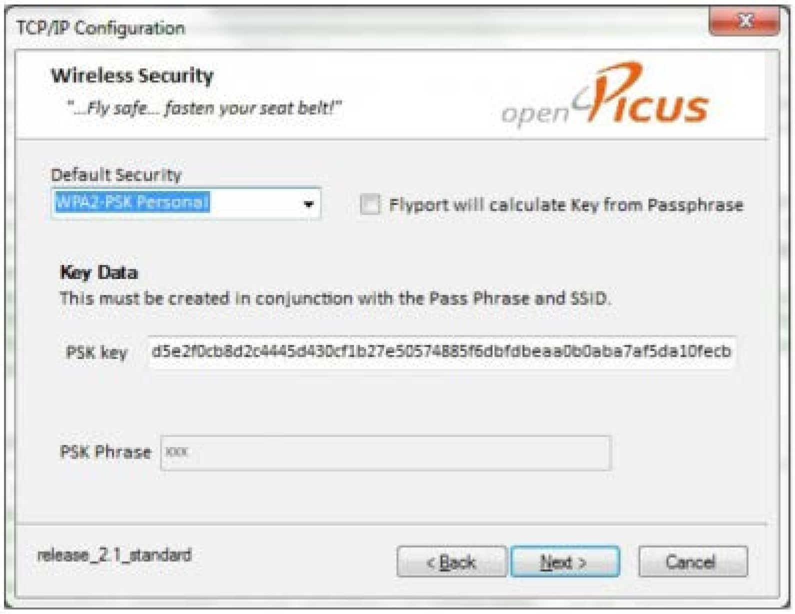
➢ **WIRELESS SECURITY CONFIGURATION**

In this page is possible to choose between the following security mode:

Open: no security.

WEP: WEP security, at 40 or 104 bits.

WPA or WPA2: WPA, WPA2, or automatic detection. NOT AVAILABLE FOR AD HOC NETWORKS.


TCP/IP Configuration

Wireless Security

"...Fly safe... fasten your seat belt!"

Default Security: WPA2-PSK Personal

Flyport will calculate Key from Passphrase

Key Data

This must be created in conjunction with the Pass Phrase and SSID.

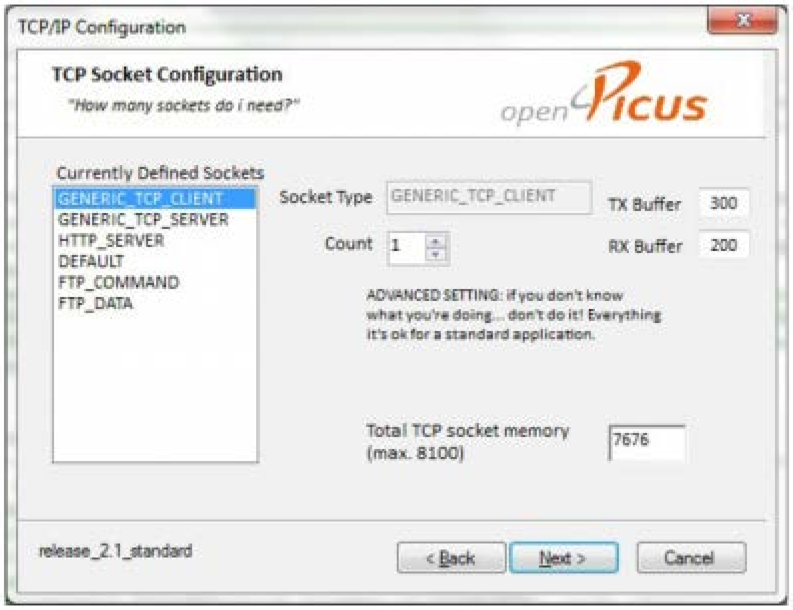
PSK key: d5e2f0cb8d2c4445d430cf1b27e50574885f6dbfdbeaa0b0aba7af5da10fecb

PSK Phrase: xxx

release_2.1_standard

➢ Socket Configuration

This page contains advanced socket configuration to select the number of available socket for each service and the memory allocated (TX and RX) for each socket.


TCP/IP Configuration

TCP Socket Configuration

"How many sockets do i need?"

Currently Defined Sockets:
- GENERIC_TCP_CLIENT
- GENERIC_TCP_SERVER
- HTTP_SERVER
- DEFAULT
- FTP_COMMAND
- FTP_DATA

Socket Type: GENERIC_TCP_CLIENT

Count: 1

TX Buffer: 300

RX Buffer: 200

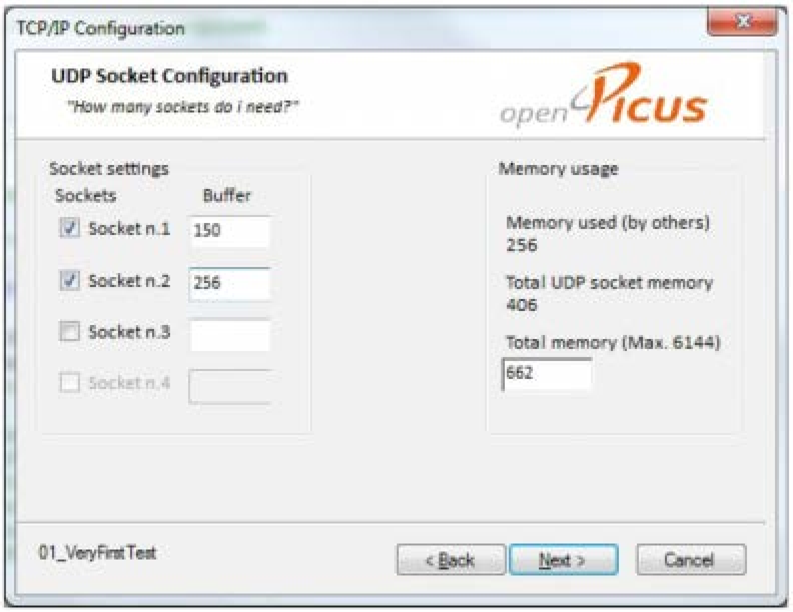
ADVANCED SETTING: if you don't know what you're doing... don't do it! Everything it's ok for a standard application.

Total TCP socket memory (max. 8100): 7676

release_2.1_standard

➢ UDP Socket Configuration

This screen allows users to configure the UDP sockets needed by user application and their related buffers.


TCP/IP Configuration

UDP Socket Configuration

"How many sockets do i need?"

Socket settings

Sockets	Buffer
☑ Socket n.1	150
☑ Socket n.2	256
☐ Socket n.3	
☐ Socket n.4	

Memory usage

Memory used (by others) 256

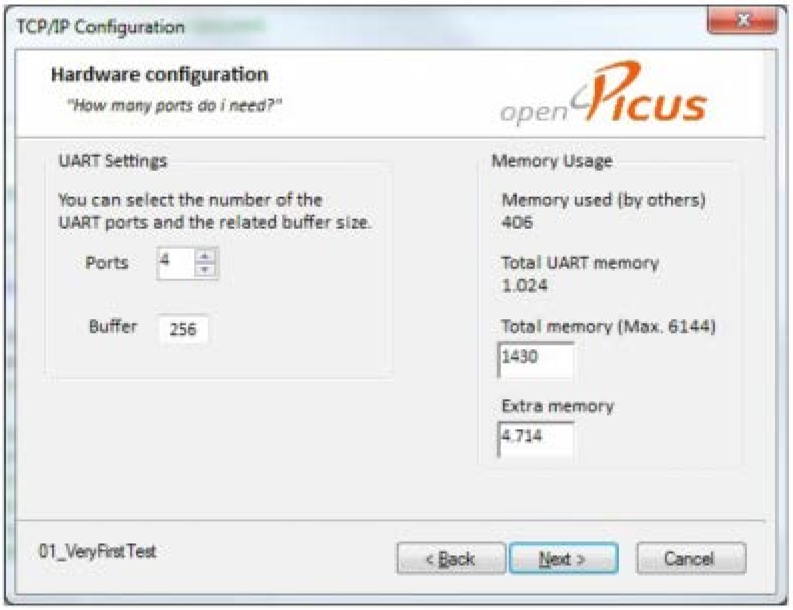
Total UDP socket memory 406

Total memory (Max. 6144) 662

01_VeryFirstTest

➢ Hardware Configuration

In this page it is possible to configure the UART configuration of Flyport's PIC microcontroller.


TCP/IP Configuration

Hardware configuration

"How many ports do i need?"

UART Settings

You can select the number of the UART ports and the related buffer size.

Ports 4

Buffer 256

Memory Usage

Memory used (by others) 406

Total UART memory 1.024

Total memory (Max. 6144) 1430

Extra memory 4.714

01_VeryFirstTest

The whole configuration is now ready to be compiled on Flyport's project! **Click "Finish" and Remember to compile again the application before to download the firmware.**

Flyport GPRS

IDE wizard for Flyport GPRS helps user to properly setup UART configuration:

How many ports enabled

UART Receive buffer size

Default baud rate for debug UART

For more details, please refer to using Flyport IDE

3.4 SOFTWARE ARCHITECTURE

FIRMWARE STRUCTURE:

The Firmware for IIIT-D WSN node has the following subfolders and files :

Project subfolders:

- **Libs –** contains the FLYPORT Framework libraries, the FreeRTOS files and the (optional) external libraries.

➢ **External libs :**

apds9300.c	for initialization of light sensor and determination of ambient light
clock.c	- for initialization of RTCC based on SNTP
ds1820.c	- for initialization of temperature sensor and determination of temp.
generalfx.c	- for byte to decimal and decimal to byte conversions
heap.s	- for correct compilation of the SNTP helper functions
pir.c	- for initialization of PIR motion sensor and determination of motion
profile.c	- for checking the network parameters to connect the FLYPORT
rtcc.c	- for alarm time configuration
upload.c	- for uploading the sensor data to the defined Server IP

➢ **Flyport Libs:**

ARPlib.c	**-** provides the commands to manage ARP-IP association.
FTP lib.c	- provides Internet communication abilities
HWlib.c	**-** Hardware library to manage the analog and digital IOs and UART
INT lib.c	- Hardware library to manage the Interrupts
ISRs.c	**-** for ISR (Interrupt Service routines)
NET lib.c	- to manage Wi-Fi and Ethernet modules and to change the network settings at runtime.
Regs.c	**-** for defining Registers
SMTPlib.c	**-** to set the parameters for the SMTP connection, and to send emails from the FLYPORT
TCPlib.c	- contains all the command to manage the TCP sockets
UDPlib.c	- provides the commands to manage UDP connections

➢ **Free RTOS :**

heap_2.c	-for memory management
list.c	-list implementation used by the scheduler
port.c -	-contains actual hardware-dependent code

portasm_PIC24.s	-contains actual hardware-dependent code
queue.c	-handle FreeRTOS communication
tasks.c	-for creating, scheduling, and maintaining tasks.

- **Obj** – contains the compiled files.
- **Web pages** – contains the webpages of the webserver (optional). The webpages and all the related files can be placed anywhere on the PC, but placing all the files inside this folder keep the complete project easier to port, store and update.
- **Doc** – contains the HTML documentation of the project. The first file that IDE will open should always be named "index.html"

<u>Project Files</u> :

➢ **Editable by user** : These files can be edited by user in IDE.

taskFLYPORT.c	: The custom application source code (with related taskFLYPORT.h).
HTTPApp.c	: The code to manage the dynamic components of the webpages for the webserver

➢ **Non-Editable by user :**

HTTPPrint.h, HTTPPrint.idx, MPFSImg2.s (pic memory usage) or MPFSImg2.bin (external flash usage)	These files contain the webpages, the images (and all media) scripts, and all the information about dynamic variables of the webserver. They are automatically generated during webpages import, user MUST NOT modify them manually.
TCPIPConfig.h and WF_Config.h	The configuration files. These files are automatically modified by TCP wizard. User should not modify them manually unless he doesn't know what he's doing. However, after manual changes (made outside IDE), to take effect, a **Recompile All** command must be issued

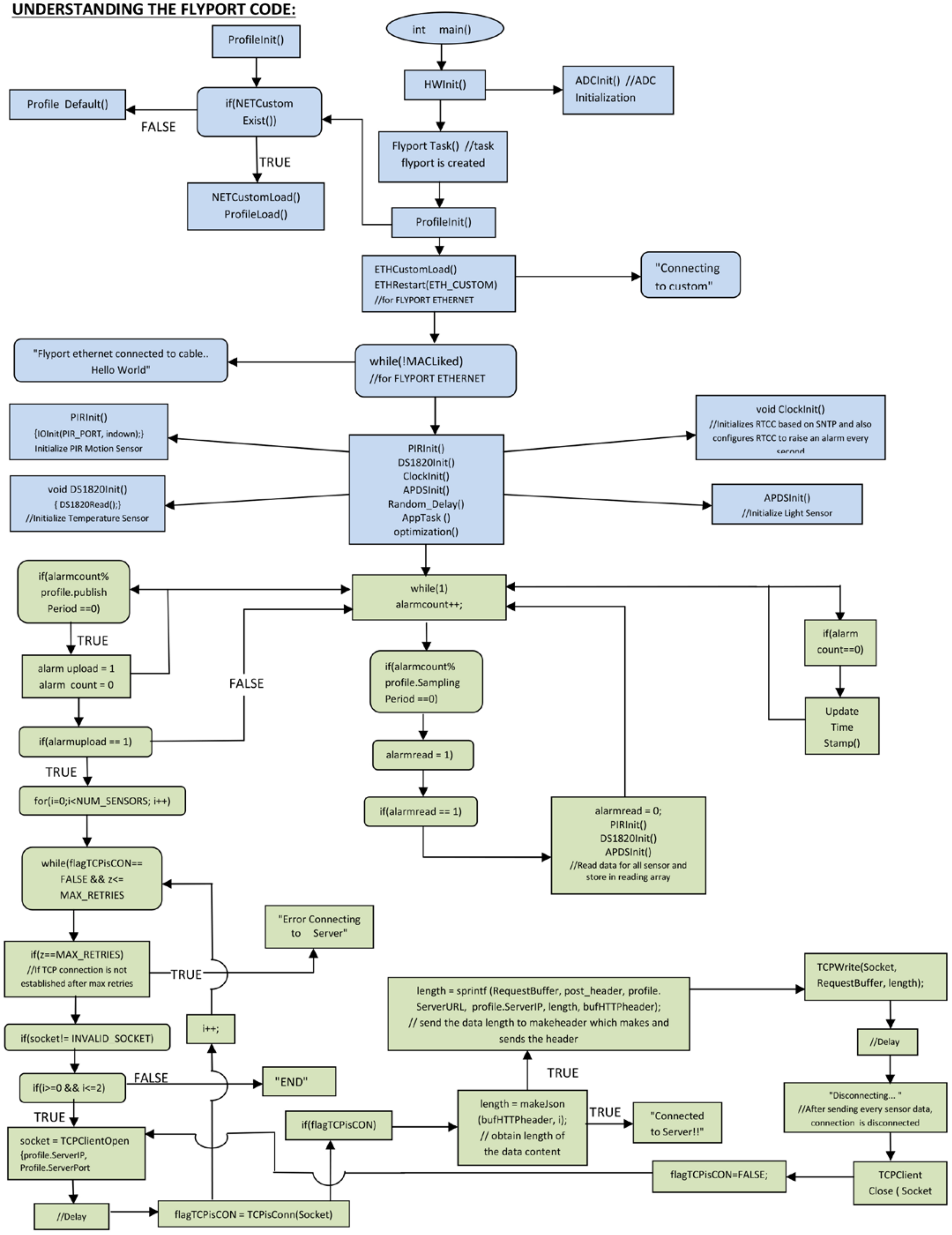
Note : These are the files which are common in both Ethernet Module and Wi-Fi module.
The next section describes the module specific files and Flow diagram of both modules for understanding the code.

<u>ETHERNET MODULE</u> :

Ethernet.conf : It's the file used by the IDE to load the project and retrieve its information for FLYPORT Ethernet

A schematic Diagram of the flow of the firmware for Ethernet Module in Flyport Development is shown below :

UNDERSTANDING THE FLYPORT CODE:


int main()
HWInit()
ADCInit() //ADC Initialization
Flyport Task() //task flyport is created
ProfileInit()
if(NETCustom Exist())
Profile Default()
FALSE
TRUE
NETCustomLoad() ProfileLoad()
ProfileInit()
ETHCustomLoad() ETHRestart(ETH_CUSTOM) //for FLYPORT ETHERNET
"Connecting to custom"
while(!MACLiked) //for FLYPORT ETHERNET
"Flyport ethernet connected to cable.. Hello World"
PIRInit() {IOInit(PIR_PORT, indown);} Initialize PIR Motion Sensor
void DS1820Init() { DS1820Read();} //Initialize Temperature Sensor
PIRInit() DS1820Init() ClockInit() APDSInit() Random_Delay() AppTask () optimization()
void ClockInit() //Initializes RTCC based on SNTP and also configures RTCC to raise an alarm every second
APDSInit() //Initialize Light Sensor
while(1) alarmcount++;
if(alarmcount% profile.publish Period ==0)
TRUE
alarm upload = 1 alarm count = 0
FALSE
if(alarmupload == 1)
TRUE
for(i=0;i<NUM_SENSORS; i++)
while(flagTCPisCON== FALSE && z<= MAX_RETRIES
if(z==MAX_RETRIES) //If TCP connection is not established after max retries
TRUE
"Error Connecting to Server"
i++;
if(socket!= INVALID_SOCKET)
if(i>=0 && i<=2)
FALSE
"END"
TRUE
socket = TCPClientOpen {profile.ServerIP, Profile.ServerPort
//Delay
flagTCPisCON = TCPisConn(Socket)
if(alarmcount% profile.Sampling Period ==0)
alarmread = 1)
if(alarmread == 1)
alarmread = 0; PIRInit() DS1820Init() APDSInit() //Read data for all sensor and store in reading array
if(alarm count==0)
Update Time Stamp()
if(flagTCPisCON)
length = makeJson (bufHTTPheader, i); // obtain length of the data content
TRUE
"Connected to Server!!"
TRUE
length = sprintf (RequestBuffer, post_header, profile. ServerURL, profile.ServerIP, length, bufHTTPheader); // send the data length to makeheader which makes and sends the header
TCPWrite(Socket, RequestBuffer, length);
//Delay
"Disconnecting... " //After sending every sensor data, connection is disconnected
TCPClient Close (Socket
flagTCPisCON=FALSE;

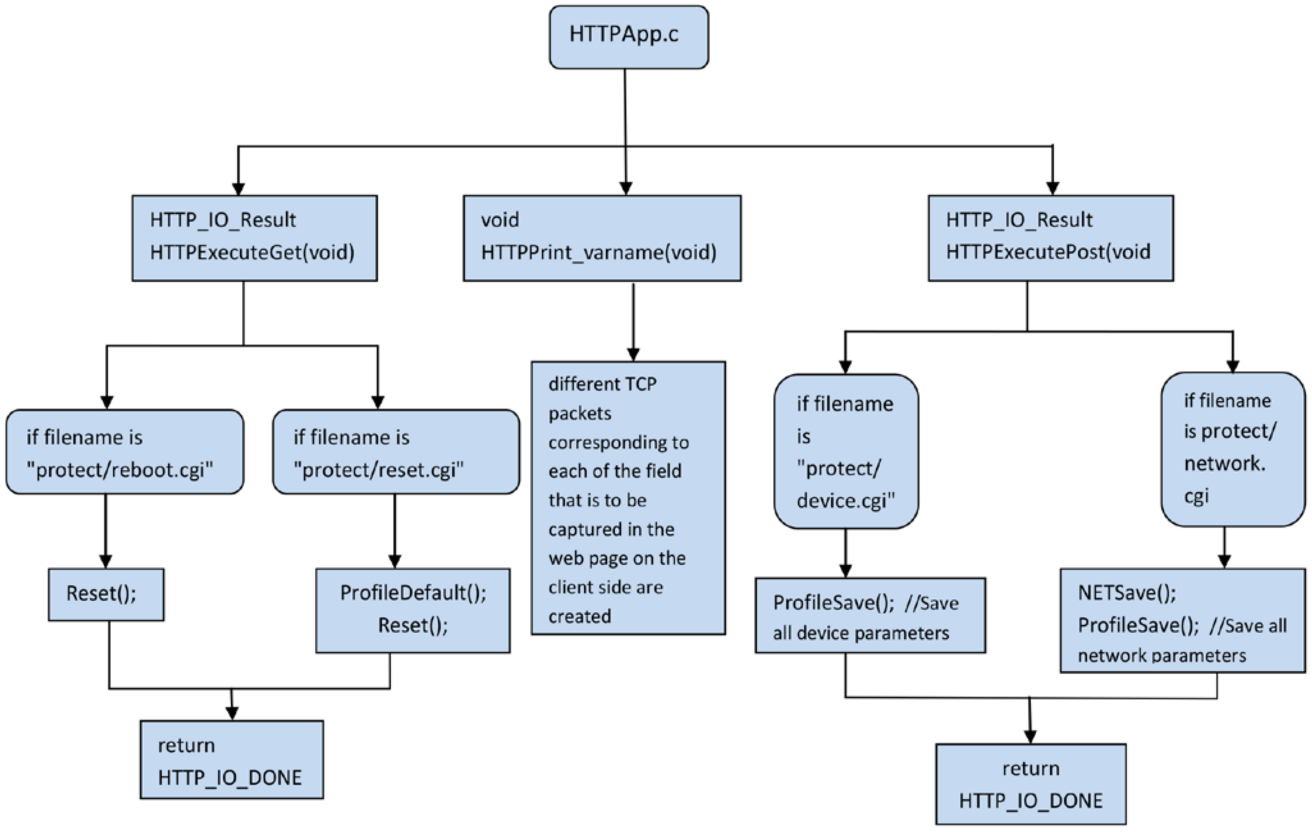
DESCRIPTION OF THE FLOW DIAGRAM :

The first step in the execution of the code is the initialization of application specific hardware. Following this, the FLYPORT Task is created. This task is dedicated to user code and is defined in the taskFLYPORT.c file.
In this task, the basic Wi-Fi information, necessary for FLYPORT to know how to connect to the network is initialized by ProfileInit( ) function. This function checks the network parameters to connect with.

If the User Defined Network configuration does exist, it means that a custom configuration was previously saved inside the flash memory of the External Memory of FLYPORT Ethernet.
In this case, the firmware connects the module to the *customized* network parameters.

In case ETH_CUSTOM does not exist it could mean that it is the first start of FLYPORT Ethernet, or that ETH_CUSTOM configuration was never saved. In this case, the firmware connects the module to the default network parameters.
NOTE: The network parameters are changed by the HTTPApp.c file if the user provide a new configuration inside the configuration webpage of FLYPORT Ethernet's web server. The function responsible of the handle of the update of parameters is the HTTPExecutePost that changes all the ETH_CUSTOM parameters, and the saving of the new configuration is done by the taskFLYPORT.c.

Please read **Programming the FLYPORT-Core Concepts** for a better understanding of the FLYPORT Web server and **Set WSN Node parameters for Node Server connection using P2P Client** to understand the configuration webpage of IIIT-D WSN node web server.
The schematic below shows the operation of HTTPApp.c :


HTTPApp.c
HTTP_IO_Result HTTPExecuteGet(void)
void HTTPPrint_varname(void)
HTTP_IO_Result HTTPExecutePost(void
if filename is "protect/reboot.cgi"
if filename is "protect/reset.cgi"
different TCP packets corresponding to each of the field that is to be captured in the web page on the client side are created
if filename is "protect/ device.cgi"
if filename is protect/ network. cgi
Reset();
ProfileDefault(); Reset();
ProfileSave(); //Save all device parameters
NETSave(); ProfileSave(); //Save all network parameters
return HTTP_IO_DONE
return HTTP_IO_DONE

The HTTPApp.c firstly provides for access protection to the FLYPORT Web page. In addition it supports for both GET and POST requests made by the client and saves the customized network parameters.

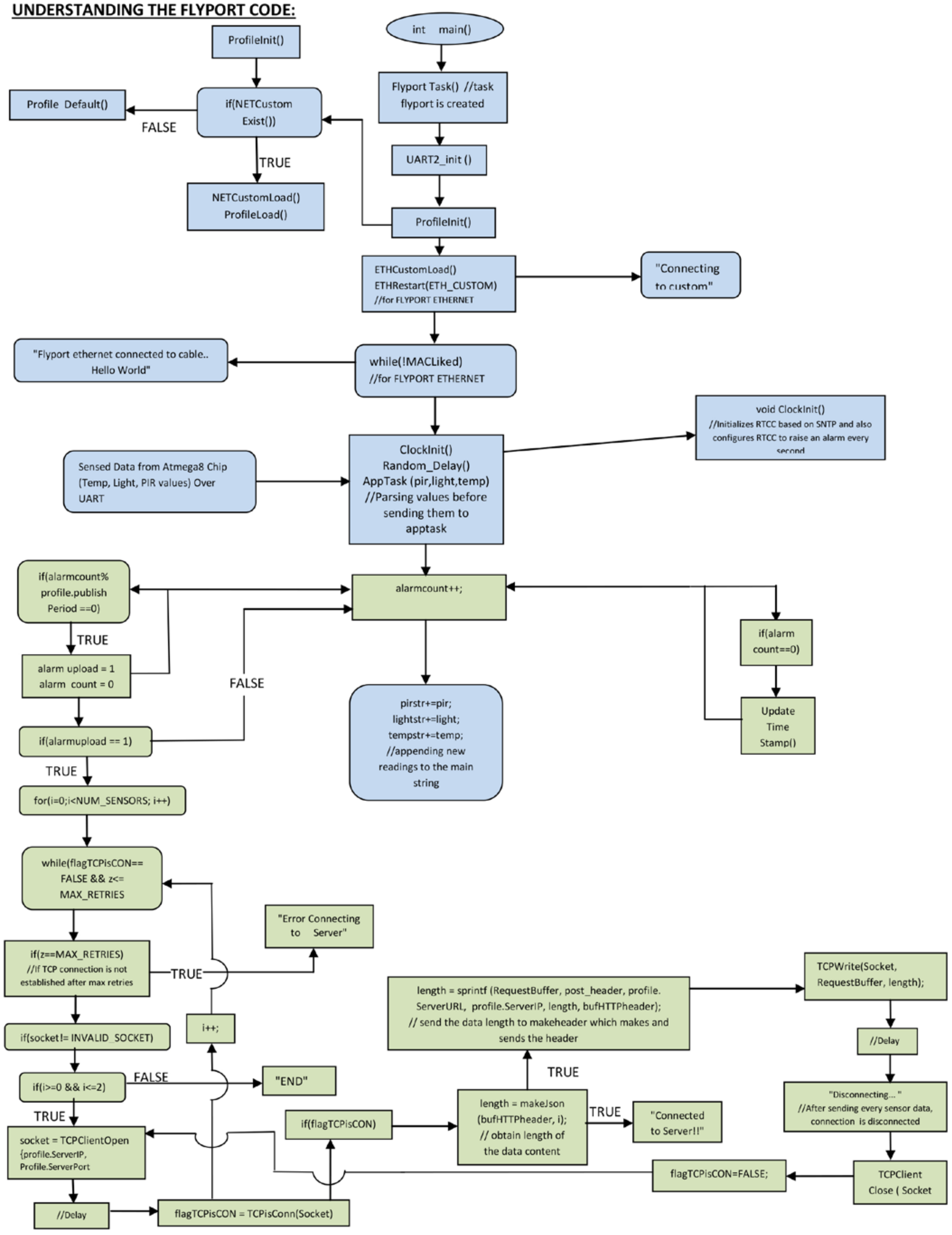
The FLYPORT displays ***"Flyport ethernet connected to the cable... hello world!"*** to indicate successful Ethernet connection on the UART.
Following a successful connection, all the sensors on-board i.e. PIR motion sensor, Temperature sensor (DS18B20 /DS18S20) and the Light sensor (APDS 9300), connected to the FLYPORT are initialized in the same order.
The **pir.c** file in the external library of the Firmware consist of the **PIRInit( )** for the initialization of the motion sensor. The **ds1820.c** file in the external library has the **DS1820Init( )** for the initialization of temperature sensor that is based upon one-wire protocol.
The **apds9300.c** file in the external library of the Firmware consist of the **APDSInit( )** for the initialization of the light sensor and it also uses the I2C library.
Also initialization of RTCC based on SNTP and configuration of RTCC to raise an alarm every second is done in **ClockInit( )** that is in the External library file, **clock.c**.
In addition, a random delay using **Random_Delay( )** is also added, which occur while initialization of the sensors.
If at the FLYPORT Web server page, client has enabled upload then the AppTask( ) function will also be executed. This task is defined in the **upload.c** file include in the External library and enables the upload of sensor data into the server in the form of JSON packets.
The function uses a variable alarmcount that is initially 0. At every initial value of alarmcount the *timestamp* accompanying the data in the JSON packet is updated.
For alarmcount divisible by the Sampling Period i.e. by default 1 and leaving no remainder, another variable alarmread is initialized to 1 and that commands the FLYPORT to read all the sensor's data and store them in the respective arrays.
The DS1820Read( ), APDSRead( ) and PIRRead( ) functions are called to obtain temperature reading, light reading and motion results respectively. All these functions are put in **Critical Sections** in order to avoid intermixing of sensor values. This is done by using **taskENTER_CRITICAL** to enter the critical section and **taskEXIT_CRITICAL** to exit the same.
For alarmcount divisible by Publish Period i.e. by default 10 and leaving no remainder, the alarmupload is set to 1, indicating that for each sensor the connection will now be established and data will be posted to the server port.
On alarmupload set to 1, the FLYPORT enters a loop to establish TCP connection and if connection is not established even after maximum tries (10 in case of firmware for IIIT-D WSN node) a message **" *Error Connecting to Server*"** is displayed on the UART.
If the FLYPORT is able to establish a TCP connection before reaching the maximum number of tries, for each sensor the Server Port at the defined Server IP is opened and a message ***"Connected to Server!!"*** is displayed on the UART.
After this the JSON packet for the respective sensor is made and pushed to the Server port. The port is then disconnected and another message ***" Disconnecting…"*** is displayed on the UART. The Server port is then closed and connection is reestablished for the next sensor's data.
On completion of data upload of all the three sensors i.e. PIR motion sensor, Temperature sensor and Light Sensor, the message ***"END."*** is displayed on the UART indicating successful posting of sensor data to the respective Server port of the defined Server IP.

ETHERNET MODULE WITH ATMEGA8:

This is a mode for Ethernet based Flyport module to take the sensors data from one another Atmega8 chip and then process it and send the data over the Ethernet to the Sensoract server. So the Atmega8 Chip actually does the sensing and then send that data over a Ethernet cable to the Flyport. Flyport takes that data from the second UART and then parse it and extracts all the sensor vaues from it. And then calls the AppTask function and pass those values to it. AppTask then appends that data so that we can use it later for making json packets.

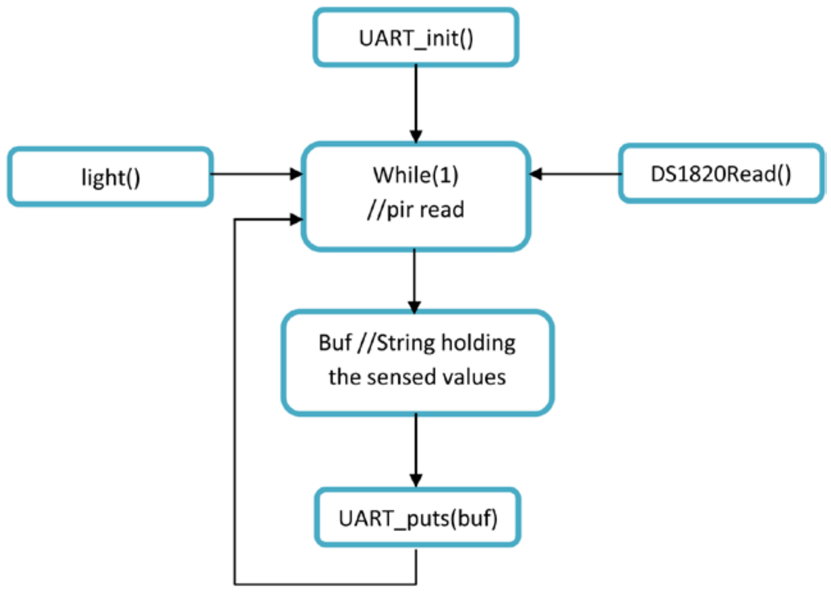
The Flow chart of this process is shown below:

UNDERSTANDING THE FLYPORT CODE:


int main()
Flyport Task() //task flyport is created
UART2_init ()
ProfileInit()
ProfileInit()
if(NETCustom Exist())
FALSE
Profile Default()
TRUE
NETCustomLoad() ProfileLoad()
ETHCustomLoad() ETHRestart(ETH_CUSTOM) //for FLYPORT ETHERNET
"Connecting to custom"
while(!MACLiked) //for FLYPORT ETHERNET
"Flyport ethernet connected to cable.. Hello World"
void ClockInit() //Initializes RTCC based on SNTP and also configures RTCC to raise an alarm every second
Sensed Data from Atmega8 Chip (Temp, Light, PIR values) Over UART
ClockInit() Random_Delay() AppTask (pir,light,temp) //Parsing values before sending them to apptask
alarmcount++;
if(alarmcount% profile.publish Period ==0)
TRUE
alarm upload = 1 alarm count = 0
FALSE
if(alarmupload == 1)
TRUE
for(i=0;i<NUM_SENSORS; i++)
pirstr+=pir; lightstr+=light; tempstr+=temp; //appending new readings to the main string
if(alarm count==0)
Update Time Stamp()
while(flagTCPisCON== FALSE && z<= MAX_RETRIES)
"Error Connecting to Server"
if(z==MAX_RETRIES) //If TCP connection is not established after max retries
TRUE
i++;
if(socket!= INVALID_SOCKET)
if(i>=0 && i<=2)
FALSE
"END"
TRUE
socket = TCPClientOpen (profile.ServerIP, Profile.ServerPort
//Delay
flagTCPisCON = TCPisConn(Socket)
if(flagTCPisCON)
length = makeJson (bufHTTPheader, i); // obtain length of the data content
TRUE
"Connected to Server!!"
TRUE
length = sprintf (RequestBuffer, post_header, profile. ServerURL, profile.ServerIP, length, bufHTTPheader); // send the data length to makeheader which makes and sends the header
TCPWrite(Socket, RequestBuffer, length);
//Delay
"Disconnecting..." //After sending every sensor data, connection is disconnected
TCPClient Close (Socket
flagTCPisCON=FALSE;

Flow Diagram for Atmega8 side is shown below:

Flow Diagram for Atmega8 Chip:



On the Atmega8 side:
After booting up Atmega8 goes in an infinite loop where after each second it sense the environment using the ambient sensor board. Then it creates a data packet and sends the packet over the UART.

Some Statistics:

Data sensing rate at Atmega8 side: **1 Hz**
Baud rate for UART used between Atmega8 and Flyport: **9600**
Format Used to send values by Atmega8:

"Temp_val:" + temp + ",Lux_calc:"+ light +"|Pir:"+ pir +"\n"

Where temp is the temperature value, light is the actually light value and pir is the pir value for any instance. And + operator is string concatenate operator.

On the Flyport side:
Main changes on the Flyport side are that now in the upload.c file flyport will not be doing the sensing instead will be taking the values sent by the TaskFlyport.c file as the function parameters to the AppTask function. Rest functionally for the flyport will be the same.

uart.h: For switching between Ethernet Flyport UART based mode and Normal Ethernet Flyport mode. You can do this by just commenting or uncommenting the #define uart.

Wi-Fi MODULE :

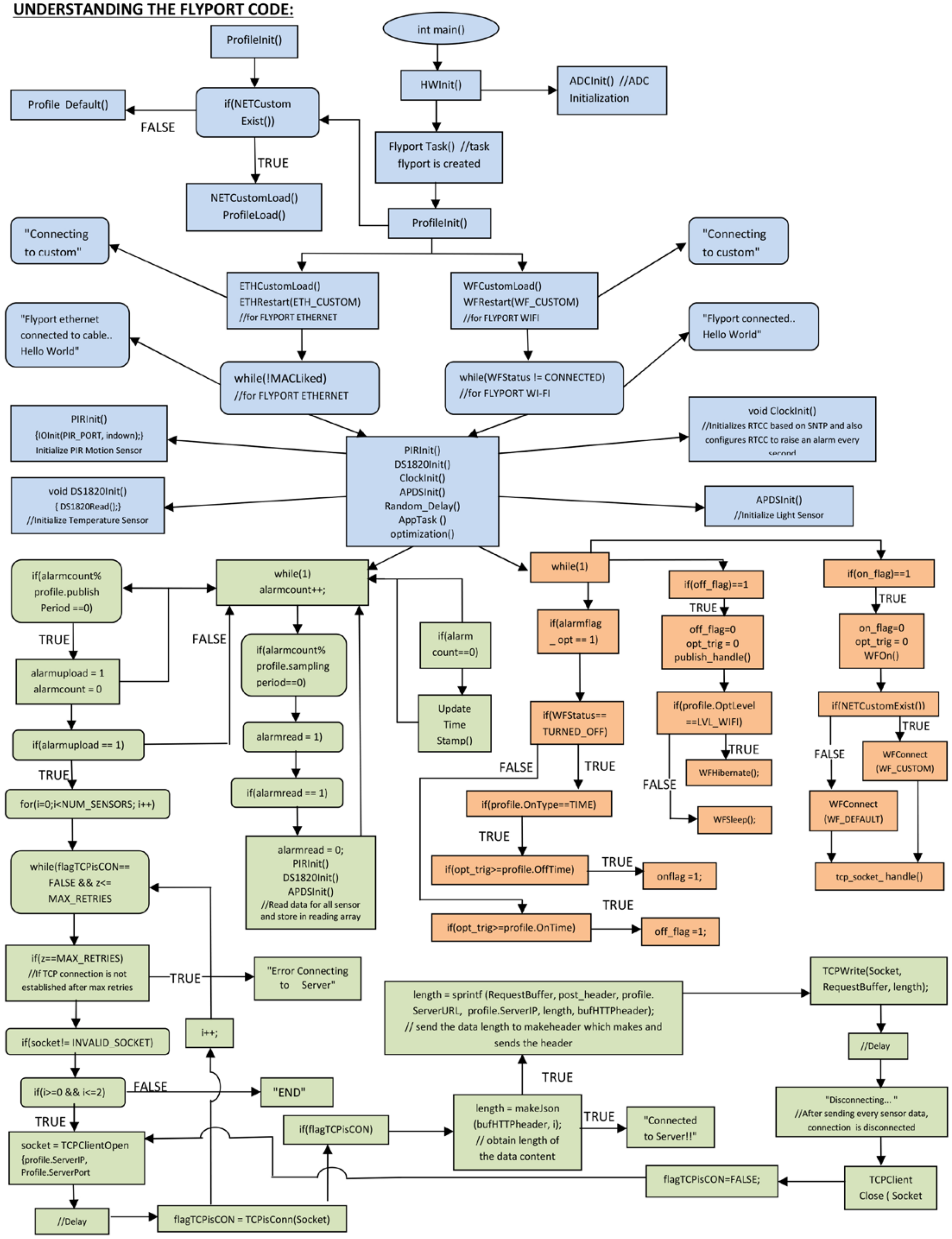
Sensorwifi.conf : It's the file used by the IDE to load the project and retrieve its information for FLYPORT Wi-Fi.

WF_events.c : The file that manages all the WiFi events (and related WF_events.h). This file can also be edited by user in IDE

Optimization.c : For utilising the flyport effectively by consuming less power. This file is present in the folder External libs in the Wi-Fi firmware.

A schematic diagram of the flow of the firmware for Wi-Fi Module in IIIT-D WSN node is shown below:

UNDERSTANDING THE FLYPORT CODE:


int main()
HWInit()
ADCInit() //ADC Initialization
Flyport Task() //task flyport is created
ProfileInit()
ProfileInit()
if(NETCustom Exist())
FALSE
Profile Default()
TRUE
NETCustomLoad() ProfileLoad()
ETHCustomLoad() ETHRestart(ETH_CUSTOM) //for FLYPORT ETHERNET
WFCustomLoad() WFRestart(WF_CUSTOM) //for FLYPORT WIFI
"Connecting to custom"
"Connecting to custom"
while(!MACLiked) //for FLYPORT ETHERNET
while(WFStatus != CONNECTED) //for FLYPORT WI-FI
"Flyport ethernet connected to cable.. Hello World"
"Flyport connected.. Hello World"
PIRInit() DS1820Init() ClockInit() APDSInit() Random_Delay() AppTask () optimization()
PIRInit() {IOInit(PIR_PORT, indown);} Initialize PIR Motion Sensor
void DS1820Init() { DS1820Read();} //Initialize Temperature Sensor
void ClockInit() //Initializes RTCC based on SNTP and also configures RTCC to raise an alarm every second
APDSInit() //Initialize Light Sensor
while(1) alarmcount++;
if(alarmcount% profile.publish Period ==0)
TRUE
alarmupload = 1 alarmcount = 0
FALSE
if(alarmupload == 1)
TRUE
for(i=0;i<NUM_SENSORS; i++)
while(flagTCPisCON== FALSE && z<= MAX_RETRIES
if(z==MAX_RETRIES) //If TCP connection is not established after max retries
TRUE
"Error Connecting to Server"
if(socket!= INVALID_SOCKET)
i++;
if(i>=0 && i<=2)
FALSE
"END"
TRUE
socket = TCPClientOpen {profile.ServerIP, Profile.ServerPort
//Delay
flagTCPisCON = TCPisConn(Socket)
if(flagTCPisCON)
if(alarmcount% profile.sampling period==0)
alarmread = 1)
if(alarmread == 1)
alarmread = 0; PIRInit() DS1820Init() APDSInit() //Read data for all sensor and store in reading array
if(alarm count==0)
Update Time Stamp()
while(1)
if(alarmflag _opt == 1)
if(WFStatus== TURNED_OFF)
FALSE
TRUE
if(profile.OnType==TIME)
TRUE
if(opt_trig>=profile.OffTime)
TRUE
onflag =1;
TRUE
if(opt_trig>=profile.OnTime)
off_flag =1;
if(off_flag)==1
TRUE
off_flag=0 opt_trig = 0 publish_handle()
if(profile.OptLevel ==LVL_WIFI)
TRUE
WFHibernate();
FALSE
WFSleep();
if(on_flag)==1
TRUE
on_flag=0 opt_trig = 0 WFOn()
if(NETCustomExist())
TRUE
WFConnect (WF_CUSTOM)
FALSE
WFConnect (WF_DEFAULT)
tcp_socket_handle()
length = sprintf (RequestBuffer, post_header, profile. ServerURL, profile.ServerIP, length, bufHTTPheader); // send the data length to makeheader which makes and sends the header
TRUE
length = makeJson (bufHTTPheader, i); // obtain length of the data content
TRUE
"Connected to Server!!"
TCPWrite(Socket, RequestBuffer, length);
//Delay
"Disconnecting... " //After sending every sensor data, connection is disconnected
TCPClient Close (Socket
flagTCPisCON=FALSE;

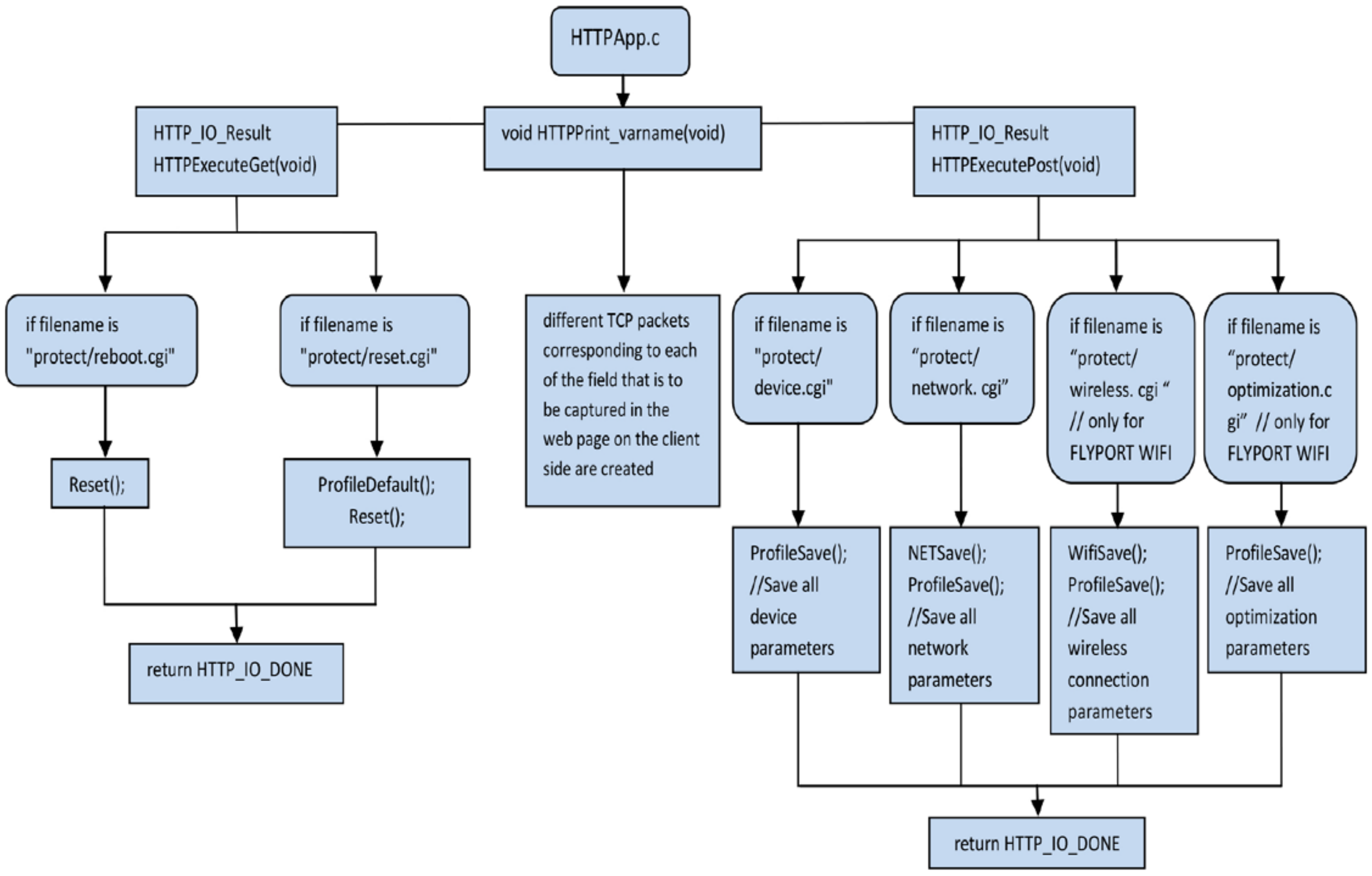
Description of the Flow Diagram :
The first step in the execution of the code is the initialization of application specific hardware. Following this, the FLYPORT Task is created. This task is dedicated to user code and is defined in the taskFLYPORT.c file.

In this task, the basic Wi-Fi information, necessary for FLYPORT to know how to connect to the network is initialized by ProfileInit( ) function. This function checks the network parameters to connect with. If the User Defined Network configuration does exist, it means that a custom configuration was previously saved inside the flash memory of the FLYPORT Wi-Fi microcontroller. In this case, the firmware connects the module to the *customized* network parameters.

In case WF_CUSTOM does not exist it could mean that it is the first start of FLYPORT Wi- Fi, or that WF_CUSTOM configuration was never saved. In this case, the firmware connects the module to the *default* network parameters.

NOTE: The network parameters are changed by the HTTPApp.c file if the user provide a new configuration inside the configuration webpage of FLYPORT Wi-Fi's web server. The function responsible of the handle of the update of parameters is the HTTPExecutePost that changes all the WF_CUSTOM parameters, and the saving of the new configuration is done by the taskFLYPORT.c. Please read **Programming the FLYPORT-Core Concepts** for a better understanding of the FLYPORT Web server and **Set WSN Node parameters for Node Server connection using P2P Client** to understand the configuration webpage of IIIT-D WSN node web server.

The schematic below shows the operation of HTTPApp.c :


HTTPApp.c
HTTP_IO_Result HTTPExecuteGet(void)
void HTTPPrint_varname(void)
HTTP_IO_Result HTTPExecutePost(void)
if filename is "protect/reboot.cgi"
if filename is "protect/reset.cgi"
different TCP packets corresponding to each of the field that is to be captured in the web page on the client side are created
if filename is "protect/ device.cgi"
if filename is "protect/ network. cgi"
if filename is "protect/ wireless. cgi " // only for FLYPORT WIFI
if filename is "protect/ optimization.c gi" // only for FLYPORT WIFI
Reset();
ProfileDefault(); Reset();
ProfileSave(); //Save all device parameters
NETSave(); ProfileSave(); //Save all network parameters
WifiSave(); ProfileSave(); //Save all wireless connection parameters
ProfileSave(); //Save all optimization parameters
return HTTP_IO_DONE
return HTTP_IO_DONE

The HTTPApp.c firstly provides for access protection to the FLYPORT Web page. In addition it supports for both GET and POST requests made by the client and saves the customized network parameters.
The SSID name, the Network Type (Infrastructure or adhoc) and Security Configuration Parameters (type and password) in the HTTPExecutePost are only for FLYPORT Wi-Fi projects.
After the customized Wi-Fi connection parameters are saved, the FLYPORT checks for Wi-Fi connection and displays a message ***"FLYPORT connected… Hello World !!"*** for successful Wi-Fi connection on the UART.
Following a successful Wi-Fi connection, all the sensors on-board i.e. PIR motion sensor, Temperature sensor (DS18B20 /DS18S20) and the Light sensor (APDS 9300), connected to the FLYPORT are initialized in the same order.

The **pir.c** file in the external library of the Firmware consist of the **PIRInit( )** for the initialization of the motion sensor. The **ds1820.c** file in the external library has the **DS1820Init( )** for the initialization of temperature sensor that is based upon one-wire protocol.
The **apds9300.c** file in the external library of the Firmware consist of the **APDSInit( )** for the initialization of the light sensor and it also uses the I2C library.
Also initialization of RTCC based on SNTP and configuration of RTCC to raise an alarm every second is done in **ClockInit( )** that is in the External library file, **clock.c**.
In addition, a random delay using **Random_Delay( )** is also added, which occur while initialization of the sensors.

If at the FLYPORT Web server page, client has enabled upload then the AppTask( ) function will also be executed. This task is defined in the **upload.c** file include in the External library and enables the upload of sensor data into the server in the form of JSON packets.
The function uses a variable alarmcount that is initially 0. At every initial value of alarmcount the *timestamp* accompanying the data in the JSON packet is updated.
For alarmcount divisible by the Sampling Period i.e. by default 1 and leaving no remainder, another variable alarmread is initialized to 1 and that commands the FLYPORT to read all the sensor's data and store them in the respective arrays.
The DS1820Read( ), APDSRead( ) and PIRRead( ) functions are called to obtain temperature reading, light reading and motion results respectively.
All these functions are put in **Critical Sections** in order to avoid intermixing of sensor values. This is done by using **taskENTER_CRITICAL** to enter the critical section and **taskEXIT_CRITICAL** to exit the same.
For alarmcount divisible by Publish Period i.e. by default 10 and leaving no remainder, the alarmupload is set to 1, indicating that for each sensor the connection will now be established and data will be posted to the server port.
On alarmupload set to 1, the FLYPORT enters a loop to establish TCP connection and if connection is not established even after maximum tries (10 in case of firmware for IIIT-D WSN node) a message ***" Error Connecting to Server"*** is displayed on the UART.
If the FLYPORT is able to establish a TCP connection before reaching the maximum number of tries, for each sensor the Server Port at the defined Server IP is opened and a message ***"Connected to Server!!"*** is displayed on the UART.
After this the JSON packet for the respective sensor is made and pushed to the Server port. The port is then disconnected and another message ***" Disconnecting…"*** is displayed on the UART. The Server port is then closed and connection is reestablished for the next sensor's data.
On completion of data upload of all the three sensors i.e. PIR motion sensor, Temperature sensor and Light Sensor, the message ***"END."*** is displayed on the UART indicating successful posting of sensor data to the respective Server port of the defined Server IP.

If at the FLYPORT Web server page, client has enabled optimization then the optimization( ) function will also be executed . This task is defined in the **optimization.c** file included in the External library and enables the Flyport or Wi-Fi as selected by the user on the FLYPORT webpage to go into sleep and then wake up to help Flyport to work more efficiently and also lowering the power consumption.
From the Flyport Webpage user can select between **Wi-FI & Microcontroller + Wi-Fi** where he wants to do Wi-Fi On/Off or he wants Wi-Fi plus Microcontroller On/Off.

There are Two ways in which optimization is implemented:
1.) **Time Triggered:** In this mode Microcontroller/Wi-Fi will be put to sleep after the ON time (as selected by the user on the webpage) and then will remain off for OFF time period (also selected by the user on the webpage) and this loop will repeat infinite times.
2) **Motion Triggered:** In this mode Microcontroller/Wi-Fi will be put to sleep after the ON time from the last motion detected and then will be ON whenever we sense any motion. (OFF time is of no use in this method)
As mentioned earlier **ON Time** (From the webpage) is the time period for which user wants to keep Flyport ON and similarly **OFF Time** is the time for which Flyport will go to sleep.

Note: Minimum limit for On/Off time is 30 seconds. If user give input below than that then the optimization will be disabled automatically as then it will be not so much efficient shifting Wi-Fi / Microcontroller states so fast as reconnecting and other things have their own overheads.
Also infrastructure mode with WPA security is not allowed with optimization because in WPA security it takes 30 sec to just make a connection.

THE JSON PACKET :

- **What is JSON ??**

• JSON stands for **J**ava**S**cript **O**bject **N**otation
• JSON is lightweight text-data interchange format
• JSON is language independent *****
• JSON is "self-describing" and easy to understand
• JSON is smaller than XML, and faster and easier to parse.

- **What is JSON object ??**

The JSON object contains methods for parsing JSON and converting values to JSON. \

- **JSON object in the IIIT-DWSN node Firmware:**

All the sensor data from the IIIT-D WSN node is uploaded to the Server Port of the defined Server IP in the form of JSON packets. There is a predefined JSON format in which the data has to be sent.
The JSON object used in the IIIT-D WSN node Firmware is as shown below:

```
char *json_object[] = { "{\"secretkey\":\"", profile.ApiKey, "\",\"data\":",
"{\"loc\":\"", profile.Location, "\",\"dname\":\"", profile.DeviceName,
"\",\"sname\":\"", sensorname, "\",\"sid\":", sensorid,
",\"timestamp\":", timestamp, ",\"speriod\":", samplingperiod, ",\"channels\":",
"[{\"cname\":\"", channelname, "\",\"unit\":\"", unit,
"\",\"readings\":[", sreadings, "]}]}}\n"
};
```

An example of data received at the Server port for the three sensors viz. Motion, Light and Temperature, in the above format is shown below:

{ "secretkey" : "45dc8ad8ecb7453a8989131d634637c8", "data" : { "loc" : "New Lab", "dname" : "dev20", "sname" : "PIRSensor", "sid" : "1", "timestamp" : 1361698211, "channels" : [{ "cname" : "channel1", "unit" : "none", "readings" : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] }] } }

{ "secretkey" : "45dc8ad8ecb7453a8989131d634637c8", "data" : { "loc" : "New Lab", "dname" : "dev20", "sname" : "LightSensor", "sid" : "1", "timestamp" : 1361698211, "channels" : [{ "cname" : "channel1", "unit" : "none", "readings" : [8.9, 8.9, 8.9, 8.9, 8.9, 8.9, 8.9, 8.9, 8.9, 8.9, 8.9] }] } }
{ "secretkey" : "45dc8ad8ecb7453a8989131d634637c8", "data" : { "loc" : "New Lab", "dname" : "dev20", "sname" : "TemperatureSensor", "sid" : "1", "timestamp" : 1361698211, "channels" : [{ "cname" : "channel1", "unit" :
"celsius", "readings" : [23.3, 23.3, 23.3, 23.3, 23.4, 23.3, 23.3, 23.3, 23.3, 23.3] }] } }

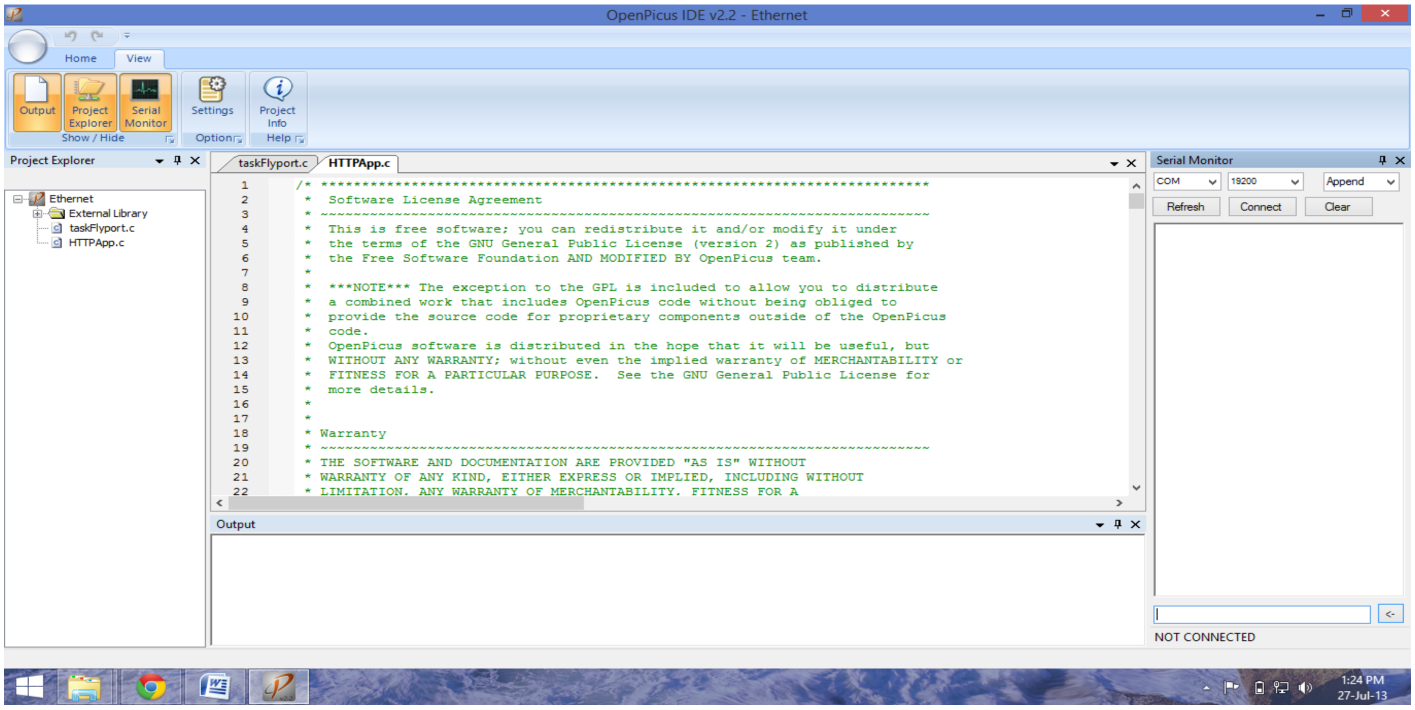
The data comprises of :
• ***Secret Key*** or the API key that is given as input by the client at the configuration webpage of the FLYPORT web server and saved by ProfileSave( ) in HTTPApp.c.
• ***loc*** that is the location of the node given as input by the client at the configuration webpage of the FLYPORT web server and saved by ProfileSave( ) in HTTPApp.c
• ***dname*** is the device name given by the client to the node, at the webpage of FLYPORT webserver and saved by
ProfileSave( ) in HTTPApp.c
• ***sname*** is the sensor name that is defined for each sensor in the switch case in makeJSON( ) in upload.c file and executed in the PostTask( ) when TCP port is open.
• ***sid*** is the sensor id that is defined in the upload.c file.
• ***timestamp*** is obtained from the SNTP.
• ***cname*** is the channel name defined in the upload.c file.
• ***unit*** is the measuring unit of sensor data specified in the switch case in makeJSON( ) in upload.c file that is
Celsius for temperature and none for light and motion data.
• ***readings*** are the sensor readings obtained by the DS1820read( ), APDSread( ) and PIRread( ) for the light, temperature and motion data respectively in the AppTask( ) in upload.c file.
In case you wish to create your own JSON object, use the REST client available for different browsers to validate it.

3.5 SOFTWARE CONFIGURATION

Set WSN Node parameters for Node Server connection using P2P Client

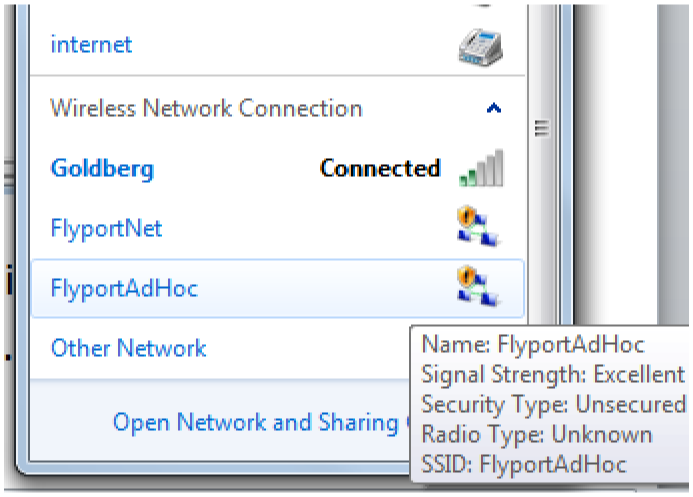
To set wireless, network and device parameters, follow the steps in a chronological manner

1. Load the latest firmware meant for IIIT-D WSN node from SVN into the program memory using Openpicus IDE.
2. After loading the firmware, connect to the serial port to emulate the behavior of node using serial response based on the current state of the node**.**



```
/* ***************************************************************************
 *  Software License Agreement
 * ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *  This is free software; you can redistribute it and/or modify it under
 *  the terms of the GNU General Public License (version 2) as published by
 *  the Free Software Foundation AND MODIFIED BY OpenPicus team.
 *
 *  ***NOTE*** The exception to the GPL is included to allow you to distribute
 *  a combined work that includes OpenPicus code without being obliged to
 *  provide the source code for proprietary components outside of the OpenPicus
 *  code.
 *  OpenPicus software is distributed in the hope that it will be useful, but
 *  WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 *  FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
 *  more details.
 *
 *
 * Warranty
 * ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT
 * WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT
 * LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A
```

3. The default configuration profile of wsn node is shown below. As the last event shown is a successful wi-fi connection which means our flyport module is broadcasting ssid (flyport adhoc) to connect to any wireless network.

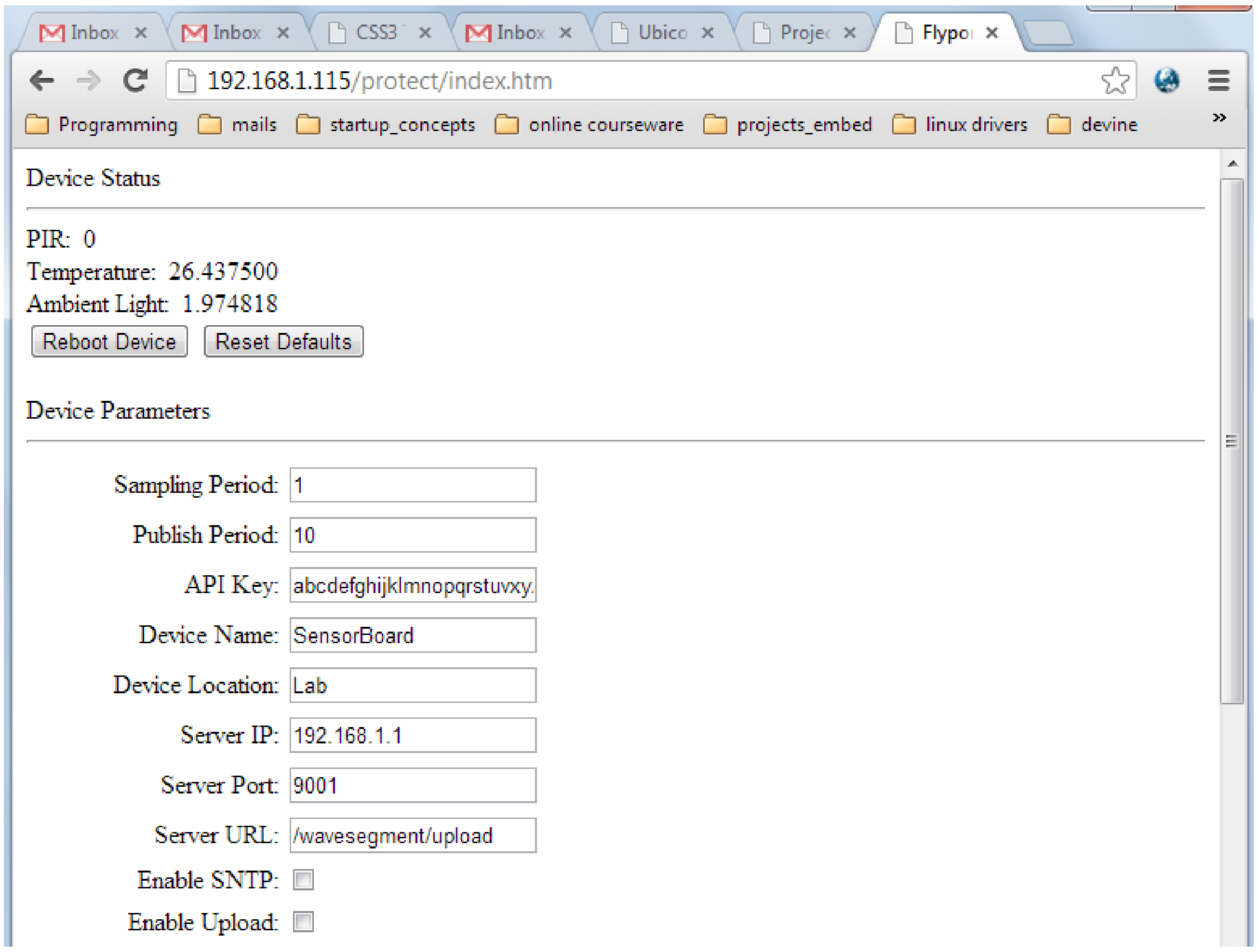


Name: FlyportAdHoc
Signal Strength: Excellent
Security Type: Unsecured
Radio Type: Unknown
SSID: FlyportAdHoc

4. Connect to this unsecured Wi-Fi network from network connections window.

5. Once you are connected, open the P2P client in a web browser having default IP: 192.168.1.115 with username: admin password: flyport.

6. This shows the P2P client with real-time light, temperature and motion sensor data along with default set of parameters from the default profile in various text fields.



Device Status

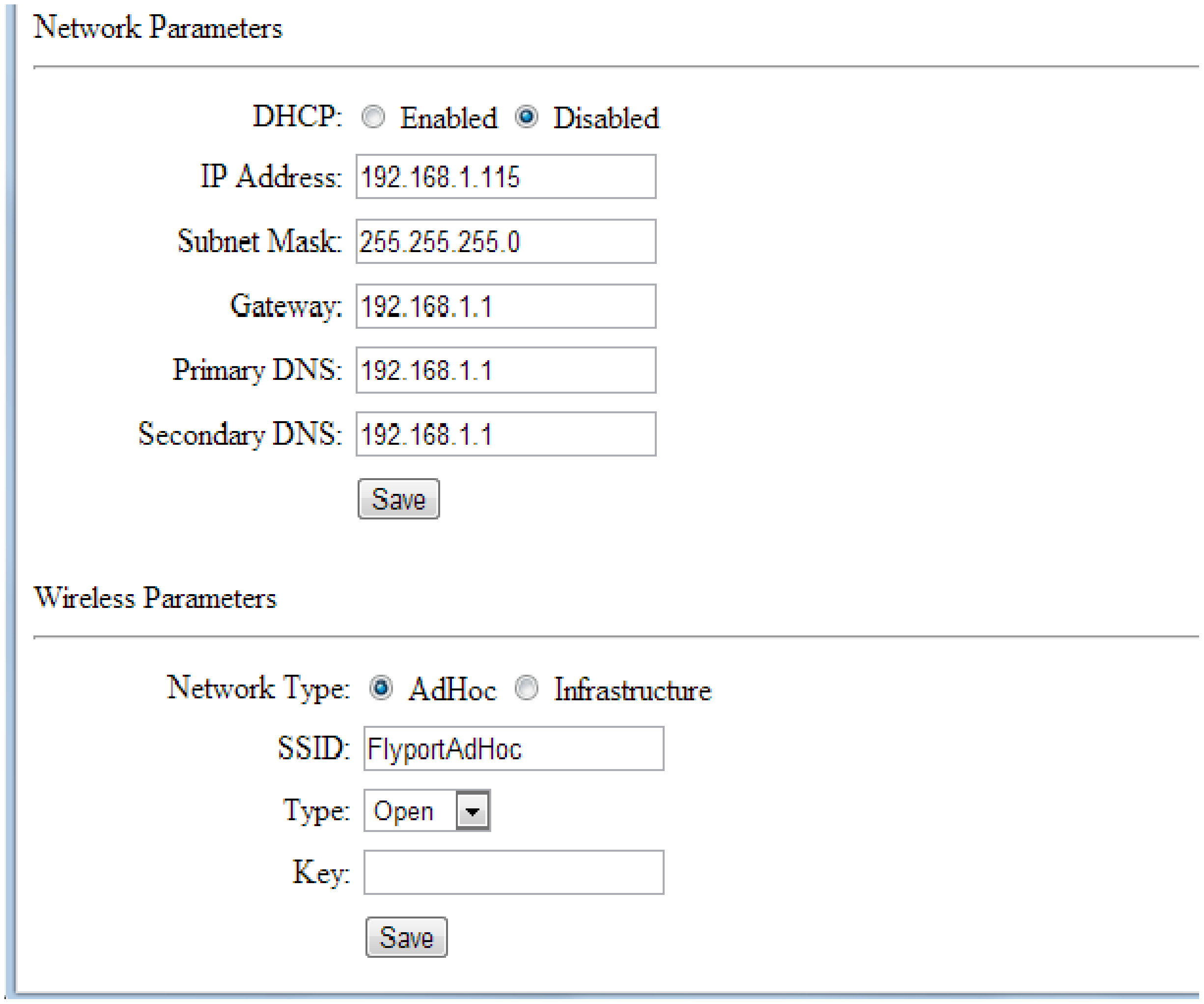
PIR: 0
Temperature: 26.437500
Ambient Light: 1.974818

Device Parameters

Sampling Period:	1
Publish Period:	10
API Key:	abcdefghijklmnopqrstuvxy.
Device Name:	SensorBoard
Device Location:	Lab
Server IP:	192.168.1.1
Server Port:	9001
Server URL:	/wavesegment/upload
Enable SNTP:	☐
Enable Upload:	☐

7. To set the access point details (for Flyport Wifi Project only) to which your node is going to connect in Infrastructure mode,

   (a) Scroll down the page as shown below



Network Parameters

DHCP:	Enabled [✓ Disabled]
IP Address:	192.168.1.115
Subnet Mask:	255.255.255.0
Gateway:	192.168.1.1
Primary DNS:	192.168.1.1
Secondary DNS:	192.168.1.1

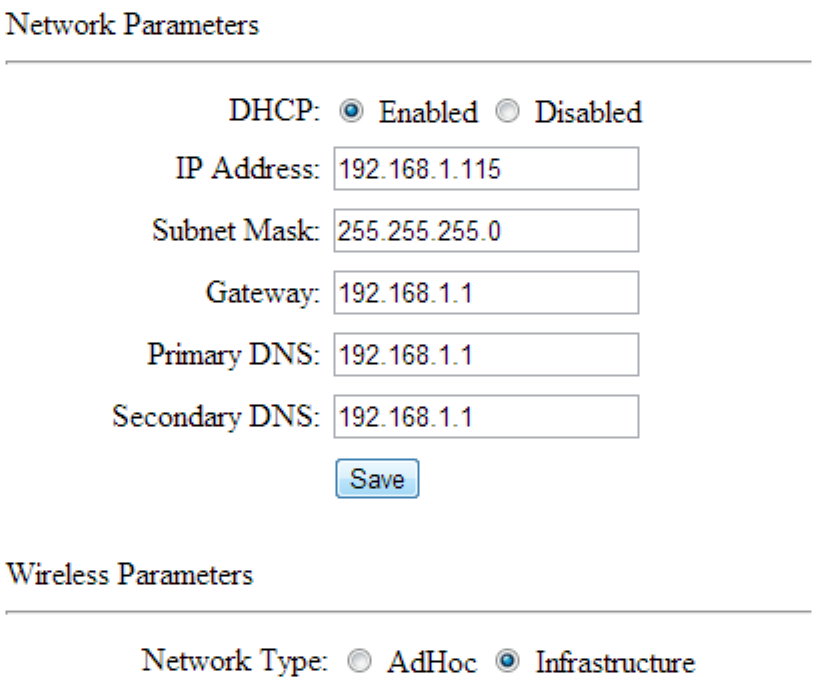
Wireless Parameters

Network Type:	[✓ AdHoc] Infrastructure
SSID:	FlyportAdHoc
Type:	Open
Key:	

(b) Select the infrastructure mode and enter the access point details. Following is a sample:
SSID: AP1
Type: WPA
Key: ap1passkey

(c) Now press Save. "Saving profile……" will be shown on the Terminal.

8. Select DHCP Enabled option in network parameters in P2P client and press Save.The node will work like a DHCP client to your access point and IP will be allocated from the access point (DHCP Server).



Network Parameters

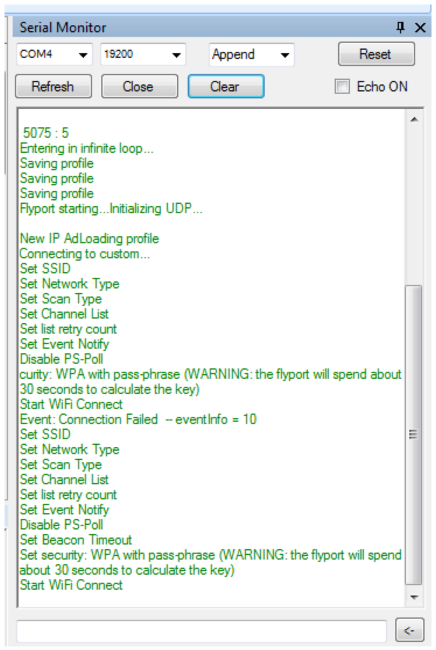
DHCP:	[✓ Enabled] Disabled
IP Address:	192.168.1.115
Subnet Mask:	255.255.255.0
Gateway:	192.168.1.1
Primary DNS:	192.168.1.1
Secondary DNS:	192.168.1.1

Wireless Parameters

Network Type: AdHoc [✓ Infrastructure]

9. Once again on the terminal, "Saving profile…" will be displayed which means you have created a new custom profile. You can now switch from Default profile to Custom profile by rebooting the node. To do that, go to top of the P2P page and click Reboot Device. This will call software reset.

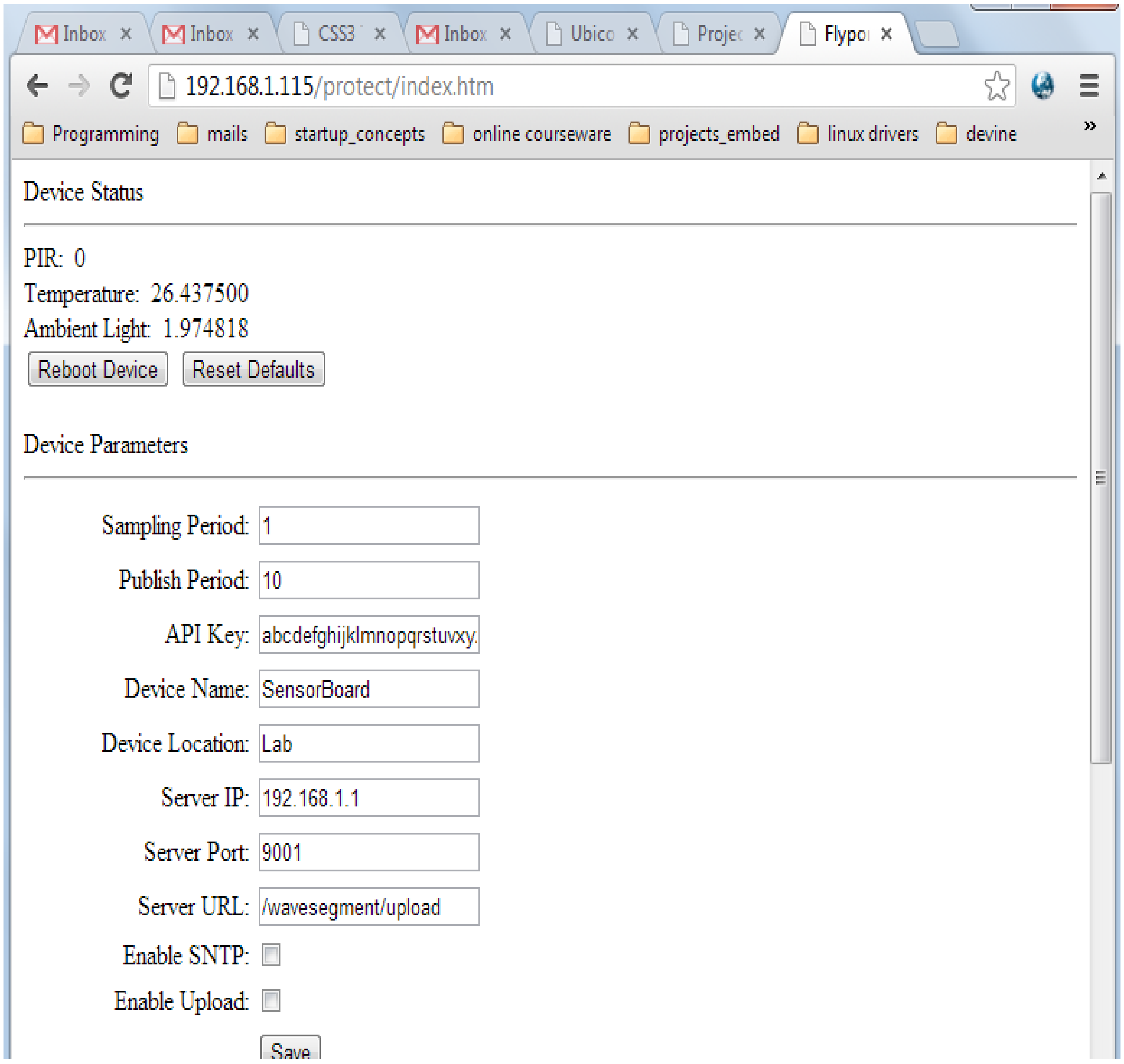
   This means the WSN node is trying to connect using Custom profile which was recently configured. After 30 seconds, as pass-phrase calculation will be over, a new IP address will be allocated by the access point (e.g. 10.0.0.5). This will also be shown in the Terminal.



```
 5075 : 5
Entering in infinite loop...
Saving profile
Saving profile
Saving profile
Flyport starting...Initializing UDP...

New IP AdLoading profile
Connecting to custom...
Set SSID
Set Network Type
Set Scan Type
Set Channel List
Set list retry count
Set Event Notify
Disable PS-Poll
curity: WPA with pass-phrase (WARNING: the flyport will spend about
30 seconds to calculate the key)
Start WiFi Connect
Event: Connection Failed  -- eventInfo = 10
Set SSID
Set Network Type
Set Scan Type
Set Channel List
Set list retry count
Set Event Notify
Disable PS-Poll
Set Beacon Timeout
Set security: WPA with pass-phrase (WARNING: the flyport will spend
about 30 seconds to calculate the key)
Start WiFi Connect
```

10. You can now connect to the same Wi-Fi access point (SSID: AP1) to access the same P2P page with the new IP address (say 10.0.0.5).



Device Status

PIR: 0
Temperature: 26.437500
Ambient Light: 1.974818

Device Parameters

Sampling Period:	1
Publish Period:	10
API Key:	abcdefghijklmnopqrstuvxy.
Device Name:	SensorBoard
Device Location:	Lab
Server IP:	192.168.1.1
Server Port:	9001
Server URL:	/wavesegment/upload
Enable SNTP:	☐
Enable Upload:	☐

11. If you want to enable Upload Calls to the server, scroll to the top of the P2P client in the browser window. Locate the Device parameters section. Specify parameters like sampling period of sensors, publish period, API key (SensorAct Data Upload API key - see instructions below), Device name, Location, Server IP, Server Port, URL, Enable SNTP and Enable Upload. After specifying these parameters press Save button to save current configuration into the custom profile and Reboot Device from this P2P client.

The reboot sequence will be shown in the Terminal and after connecting with the access point, the node will try to make a TCP connection with the Server every 10 seconds to upload wave segment containing sensor readings using HTTP POST request.

Appendix A: REFERENCES

The following table summarizes the documents referenced in this document.

Name	Description	Location
Open Picus Wiki	Contains technical details about Flyport modules, carrier boards and IDE	http://wiki.openpicus.com/index.php/Main_Page
Getting Started	Contains compiler and IDE download link	http://wiki.openpicus.com/index.php/Getting_Started
Node Server connection using P2P client	Software configuration	https://github.com/iiitd-ucla-pc3/SensorAct/wiki/Set-WSN-Node-parameters-for-Node-Server-connection-using-P2P-Client
Flyport Deployment	Download Firmware	https://iiitd-flyport-development.googlecode.com/svn/trunk/
Flyport IDE	Hardware Configuration	http://wiki.openpicus.com/index.php/FLYPORT_IDE#Using_FlyPort_IDE
Hardware Schema	Schemas Link	https://iiitd-flyport-development.googlecode.com/svn/Files/