Notes:

# HARDWARE:

PHY Transceiver

* Physical layer of the OSI model…
* Connects a link layer device to a physical medium (Wire)

Ethernet Physical Transceiver

* Works on the physical layer
* Provides an analog signal
* Normally interfaced with a MII (Media independent interface) to a MAC chip in a microcontroller or similar.
* Hardware send and receive function of \*Ethernet frames\*

Power System:

* Handle 1-2 Lipos Operation
  + lithium-ion polymer battery (Just a battery)
* So, we need to have input of two batteries. (Most likely from the rest of the system)

MOSFETs

* rated at 4x current and voltage required to ignite the Ignitors and e-matches used

# SOFTWARE:

* Have to get battery voltage
  + How do we measure that?
* Also measure how many battery’s there are in the system (1-2)
* FSM for each of the ignition systems
  + Arming
  + Firing
  + Voltage
  + Lockouts

Coms layer handling messages

* + Output status messages
  + Two igniter states and voltage

Messages received are:

* + Ping – return pong
  + Arm igniter 1 – Wait for an 8-byte password
  + Fire igniter 2 – Wait for an 8-byte password
  + Arm igniter 1 – Wait for an 8-byte password
  + Fire igniter 3 – Wait for an 8-byte password

# Questions?

* What Gs do these need to work with?
* Do they need to always display data throughout the flight?
* Will the range of the PHY effect the chip I want to use and how far is the rocket going for?
* Can we use surface mount? – Assuming we can because needs to be small.

<https://www.microchip.com/en-us/product/KSZ8051#document-table> – Does this work?

* 3.3V
* parallel-to-serial conversion

# PHY CHIP

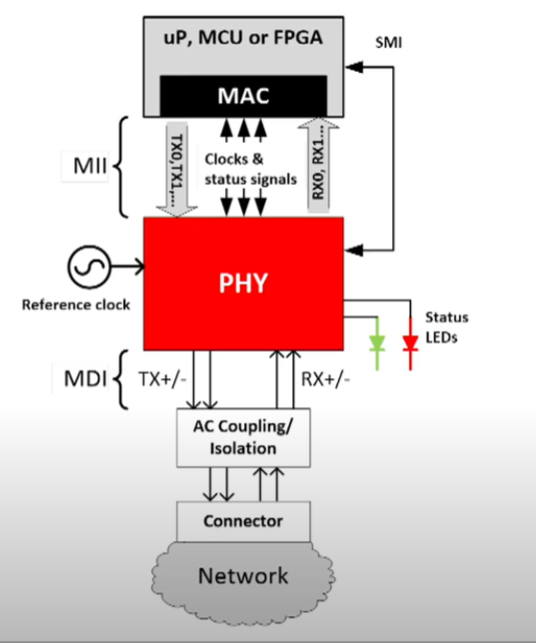
## MII

<https://en.wikipedia.org/wiki/Media-independent_interface>

Interface to connect a MAC to a PHY chip (This seems to be the reason how we can use different types of inputs?)

Because of these different types of PHY devices can connect without changing the MAC hardware

## MAC

* Controls the hardware responsible for interacting with the wired medium. flow control and multiplexing for the transmission medium.

https://www.youtube.com/watch?v=JH3cMYErmKI

General layout for a PHY

Mac device can be a microcontroller :O

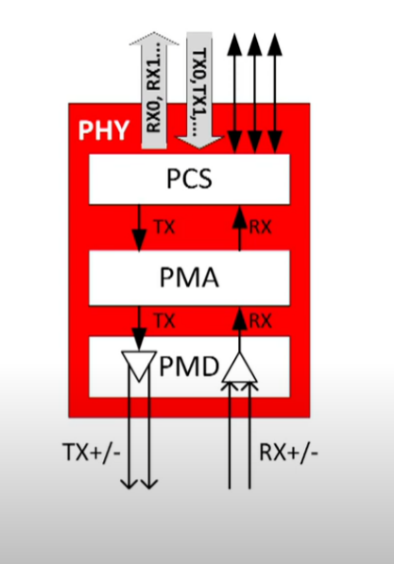
MII Connects PHY and the MCU

(10/100/1000MBS+) – data speeds supported

SMI access eternal registers

MDI the physical wire (optical copper)

External clock (25MHz +/-100ppm) -Error

INSIDE THE PHY

PCS – Physical coding Sublayer

* Encode and decode data (125mbs)

PMA – Physical medium attachment layer

* Bit to signal mapping
* Detection of error

PMD – Physical medium dependent layer (not always used)

* Functions to support the wires input

Multiple streams in parallel for faster speeds

## PCS

* Encoding and decoding
* Monitor activity of the channel as well as finding collisions within the network
* COL for collision
* Reverses the send encoded data to get back message
* RXDV – successful signal sent
* RXER – Detected an error in the channel
* Function to synchronise the clock with the transmitting device (CLOCK Recovery function)

## PMA

* Convert encoded to appropriate data
* Network into bits
* Separates the random junk through the channel into useful bits (Like analogue signals into useful IDK)
* Link Status? – Auto Negotiation?
* Can also receive falts

## PMD

* Converting TX to physical signals for network
* Not always used
* Twisted pairs = more PMD
* Debugging medium

PAM5? PAM3? <https://en.wikipedia.org/wiki/OSI_model>?

<https://electronics.stackexchange.com/questions/35033/how-to-measure-battery-voltage-from-a-microcontroller> or <https://www.instructables.com/Measure-Current-and-Voltage-Using-a-Microcontrolle/>

* To measure the voltage of the battery.
* Is that complicated? IDK
* Seem to have to read from the ADC and get an input then change that by the given formula
* Use a coulomb counter? – But we don’t know reference battery amount? But we know how much to take away? Maybe not sure

AVOID BROWNOUTS – Should have a better look into this

* They are a drop in electrical power supply system
* For example, when light dim! They go browner.
* A “brown out” of a microcontroller is a partial and temporary reduction in the power supply voltage below the level required for reliable operation.

For speed and reliability, we are going to use an STM32 board (For job)

# STM32:

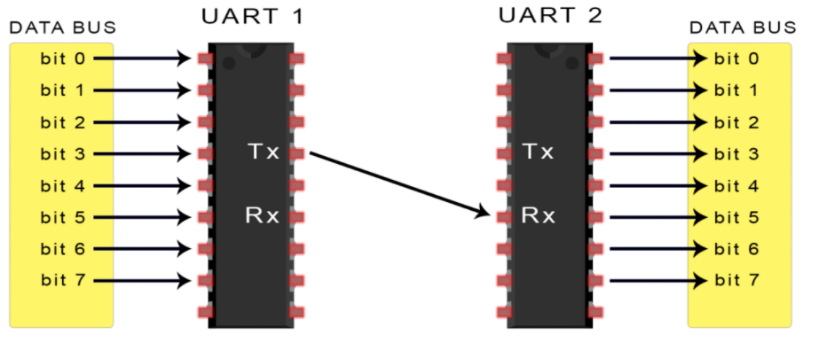
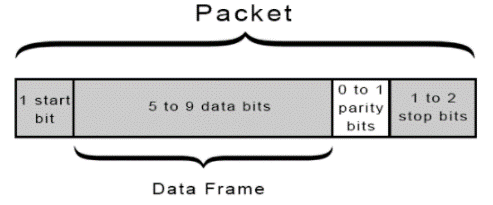
* Arm
* Mpu – (operating system)
* F0 – mainstream controller
* Architecture (ARM Cortex – M3/4) – This is important
* Use development boards for testing
* Program with C/C++
* The pin layouts are very similar so you can swap between them nicely (Speed and memory)
* Stm32CubeIDE – windows
* Like eclipse file has a workspace ;( LEGIT ECLIPSE.
* Pick specific part to work on (board selector and close the L0 one)
* Turn on all peripherals (better for the Nucleo not so much otherwise.)
* Graphical tool within Cube IDE
* Categories is where to find all peripherals. Or just click it and pin the type of pin we want – very cool
* Clock view? – might have to set these
* Current draw calculator in tools which is nice
* File Save prompts generating code
* Have main file like the Eclipse stuff
* Super loop – The graphical interface generates all the code for us nicely.
* Has guard comments to stop your code from being deleted (USER CODE END WHILE)
* MUST HAVE CODE BETWEEN THESE OR IT WILL GET DELETED
* Gnu tools library
* What libraries do we need? – website
* Stm32XX Hal – Peripherals used in order (toggle pin)
* All the functions on the documentation
* Has step through debugging and a whole bunch of stuff – break points and etc
* Very similar to the CCS

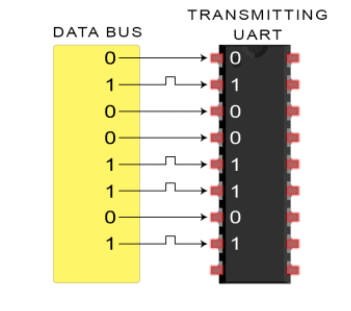
Do the practise projects from the website when get a cord for it to work!!! So we can have some knowledge of how to get it working

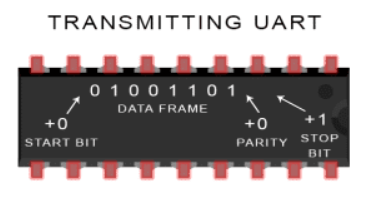
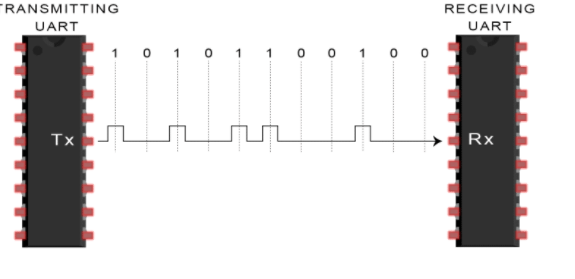
<https://www.youtube.com/watch?v=hyZS2p1tW-g> – Help

# Comm Systems

## UART - https://www.circuitbasics.com/basics-uart-communication/

* Communicates all through TX and RX - Universal Asynchronous Receiver/Transmitter
* Converts parallel data from a controlling device like MCU to serial form (serial = 1 bit at a time) then is converted back by the other controller to be used
* Only 2 wires needed
* Asynchronously so no clock is needed. It adds a stop and start bit to say when the packets stop and end.
* Reads at the Baud rate (speed of data transfer in bits per second) both must operate at the same baud rate or else you will becorrupt data (10% off or will be wrong approximately)
* Gets the data adds stop start and parity bit then sends it serially through the channel as a packet





Pretty fool proof.

Advantages:

* 2 Wires, No clock, Parity but for errors?

Disadvantages:

* Limited to 9bits, no slave or master systems, baud rates must be close.

## CANBUS