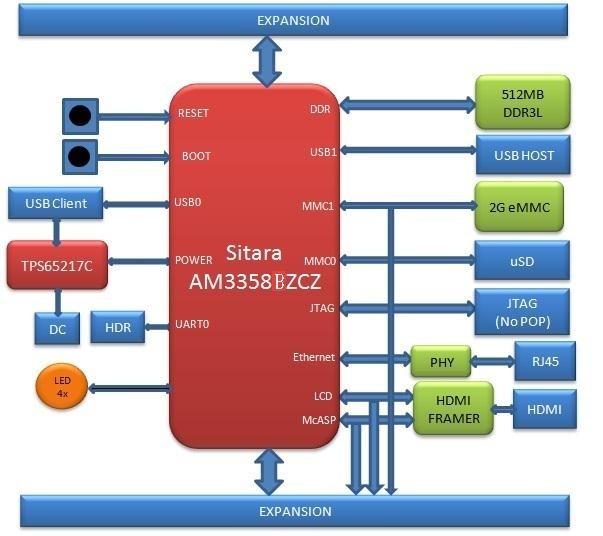
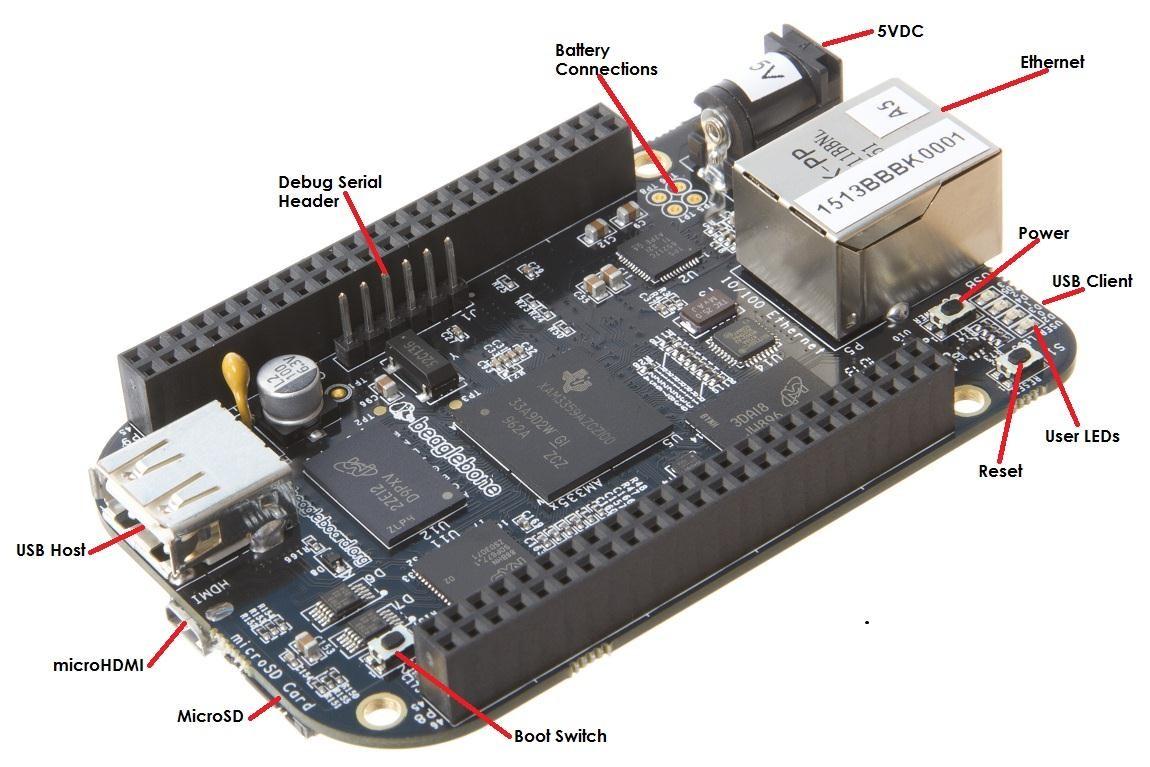
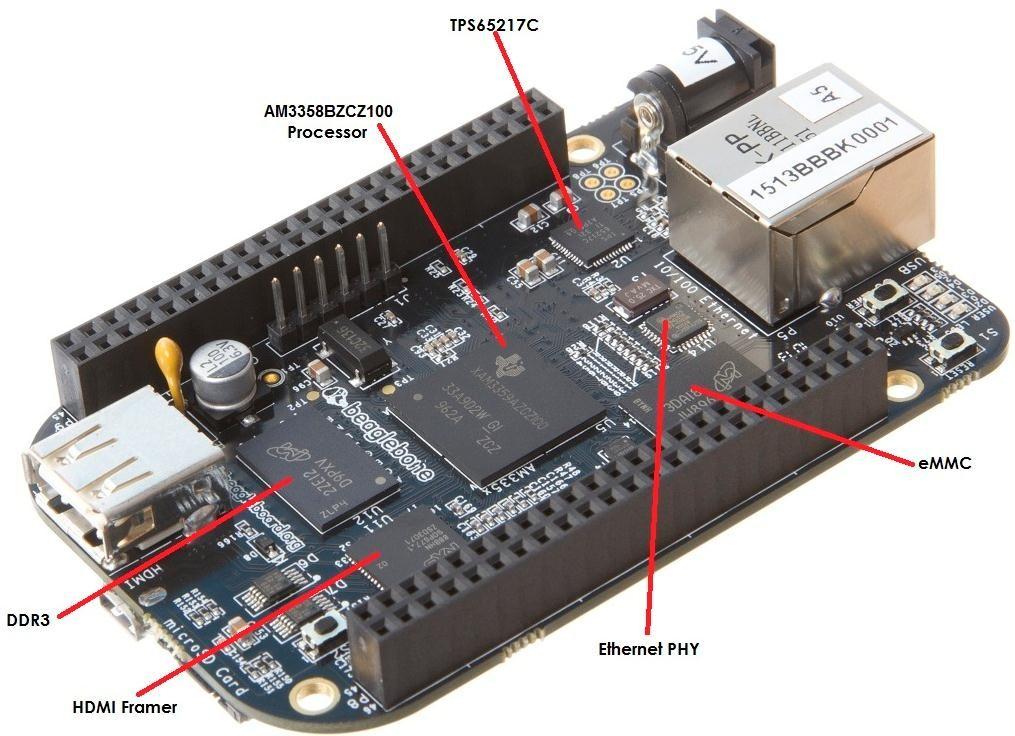
# BeagleBone Diario de Bordo

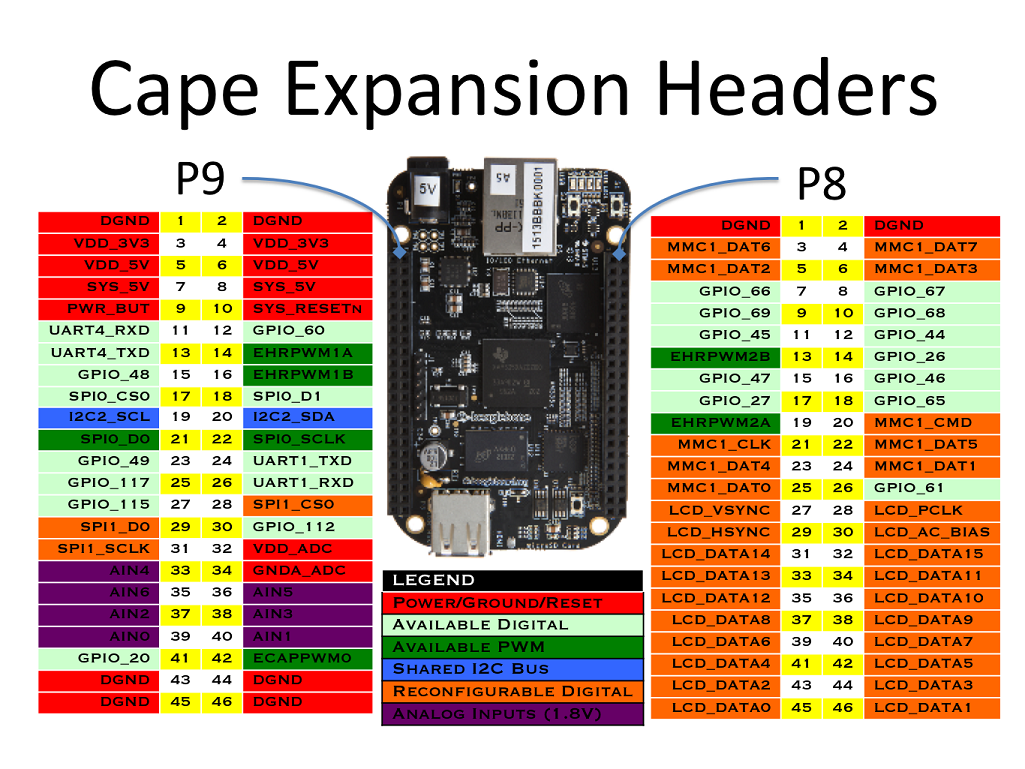
Objetivos: Primeiros passos na Beaglebone Black (BBB) e no código freemodbus.

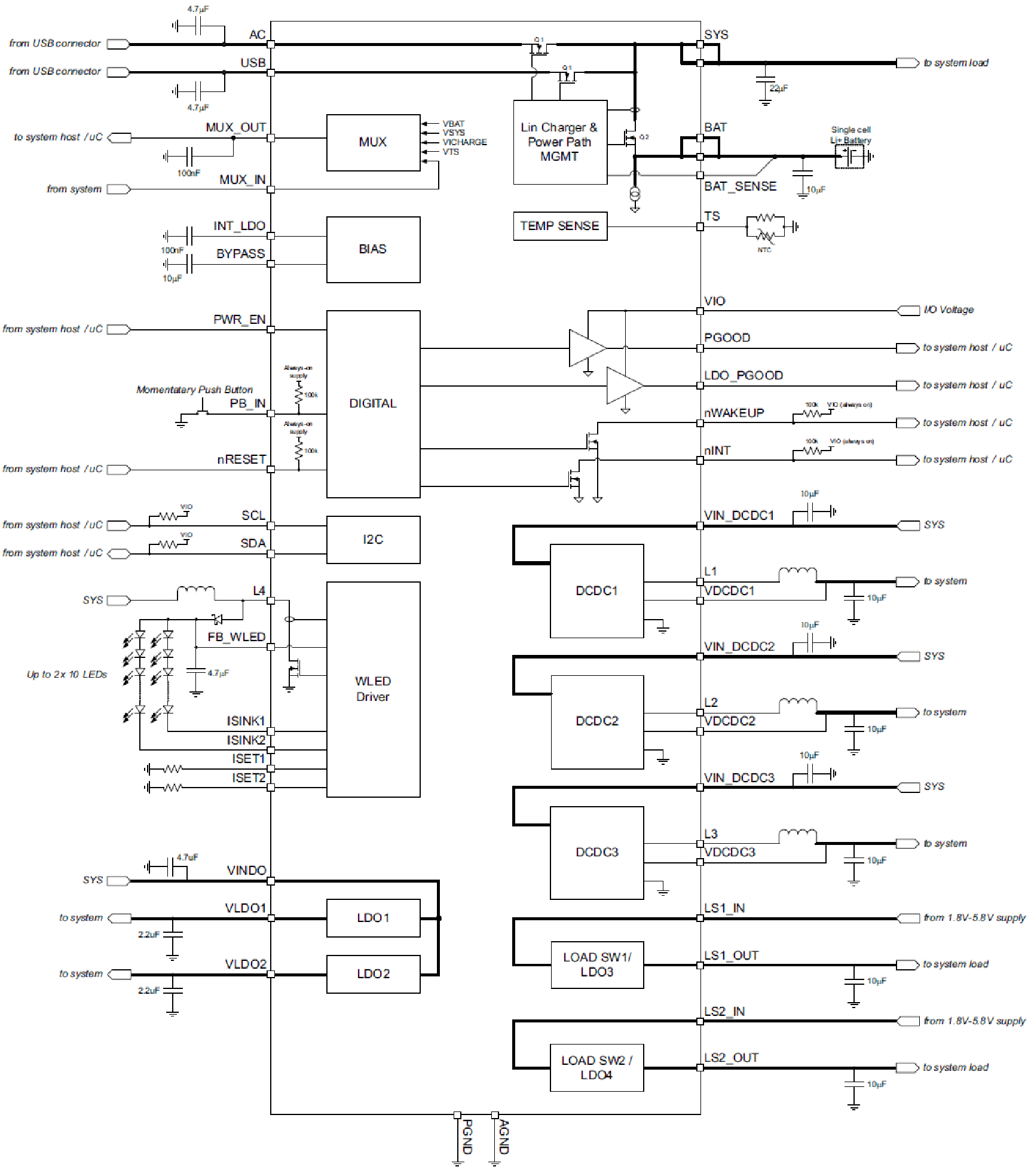
# Hardware da BBB











# Maquina Virtual Linux

O usuário e a senha da máquina virtual Ubuntu do fabio é :

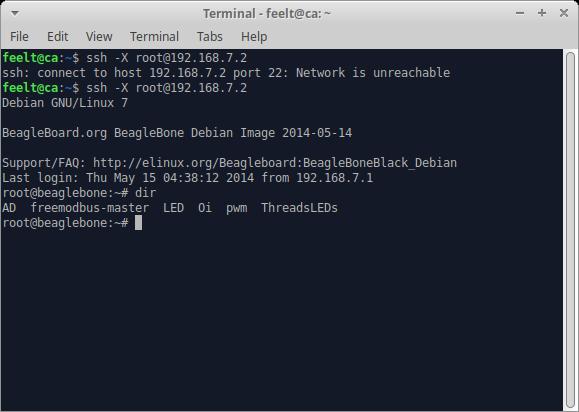
User: UFU

Senha : 1

## Conectando a BBB pelo cabo serial

quando configurado somente o cabo serial:

ufu@SE:~$> ssh -X [root@192.168.7.2](about:blank)



Aqui acabou que eu não precisei logar, mas tem vezes que precise logar. (descobrir a senha da BBB)

## Transferindo arquivos através do Filezilla

Ainda dentro da máquina virtual eu conectei também ao filezilla. Este programa servirá para descarregar o programa diretamente na BBB após ter compilado no PC (Linux, neste caso da máquina virtual). Ele serve para copiar toda a pasta ou conteúdo para dentro da beaglebone, ou fazer o contrário, ler algo da beaglebone para o computador.

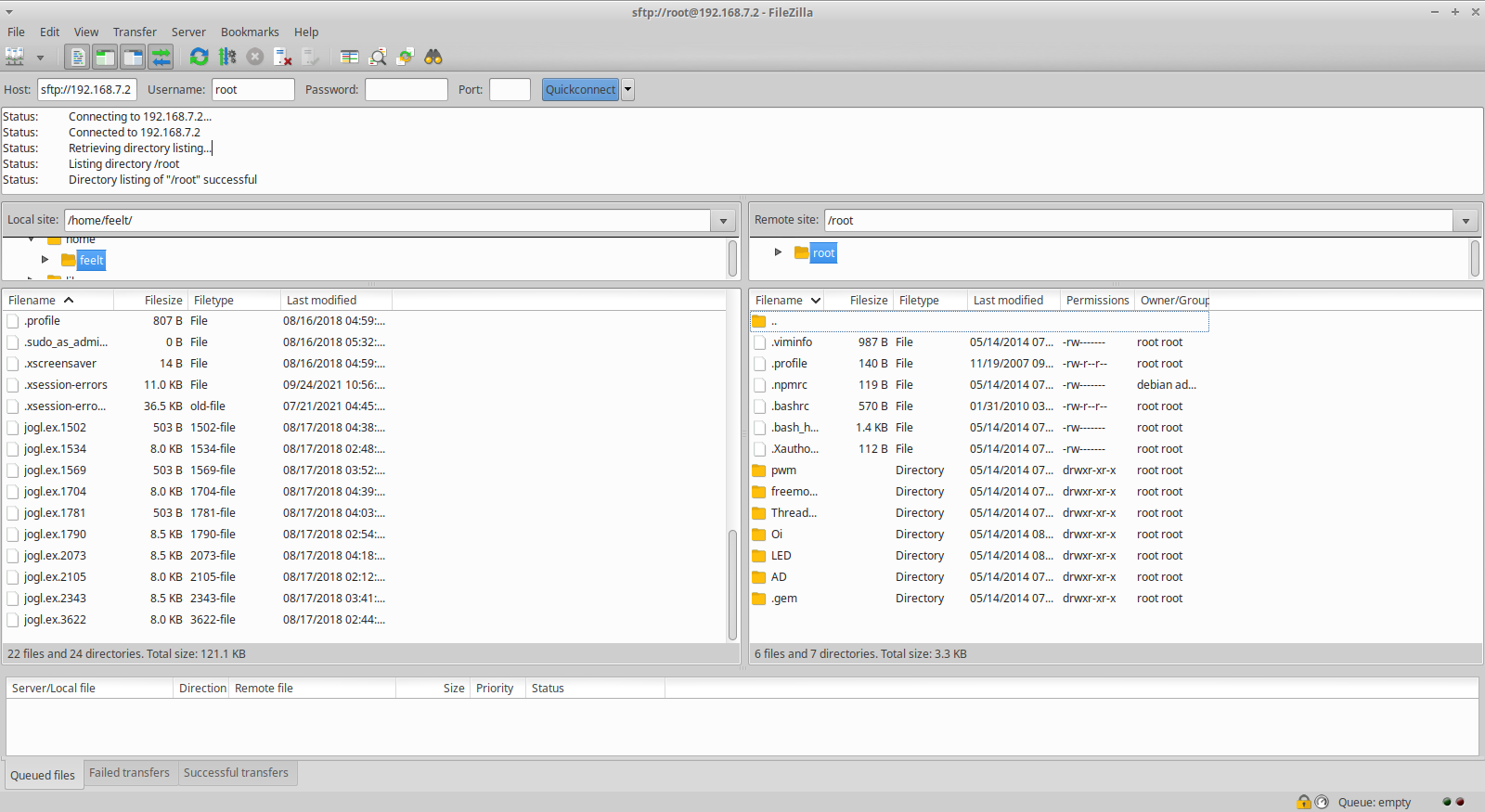
Primeiramente, coloquei o cabo USB e loguei na bbb conforme mostrado no item anterior (porta USB\SSH). Para conectar o Filezilla eu fiz a seguinte configuração:

Host : 192.168.7.2 (conexão pelo cabo serial USB)

User : root

Senha: nenhuma

Porta : 22 (que é a porta SSH).



Na Figura acima, do lado direito está o conteúdo da BBB. Do lado esquerdo o conteúdo do PC. O que eu quiser trocar entre os dois é possível, facilmente ou arrastando ou escolhendo o arquivo e fazendo upload ou download.

Dicas: quando eu vou substituir um arquivo já existente na BBB parece que as vezes da problema e até ele desconecta do Filezilla. Ideal é vc primeiro apagar na BBB (e deixar na pasta certa do lado direito), então ir no lado esquerdo (PC) e então selecionar o arquivo e fazer upload. Ele vai copiar para onde está já apontando previamente.

Outra dica eh que as vezes o filezilla, após um determinado tempo com a janela aberta fecha, e meio que trava, ficando sempre tentando reconectar. Ele não vai conseguir reconectar. Deve desligar a conexão filezilla, desconectar a conexão do terminal e tentar novamente. Para evitar isso parece que eh melhor sempre desconectar do filezilla.

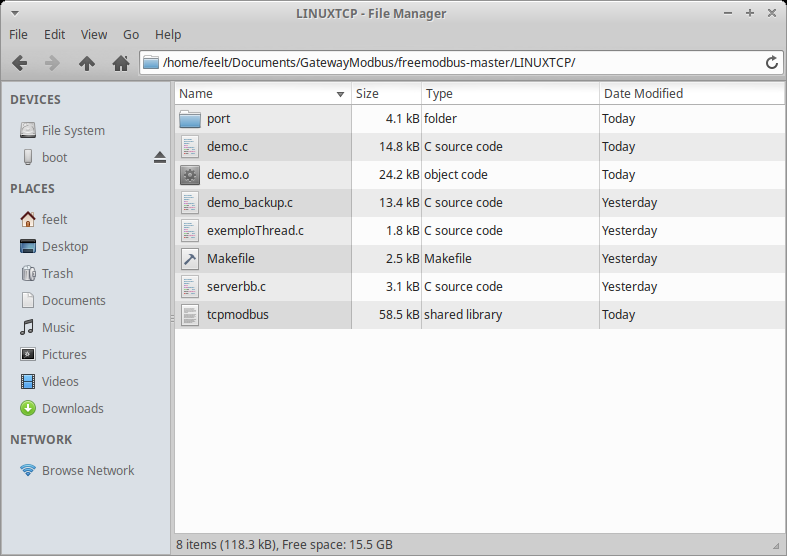
# Compilando o programa no PC

Uma opção que eu fiz foi colocar o mesmo programa na Máquina Virtual (PC) e compilar ele no PC. Para isso precisei instalar o arm-gcc no pcc.

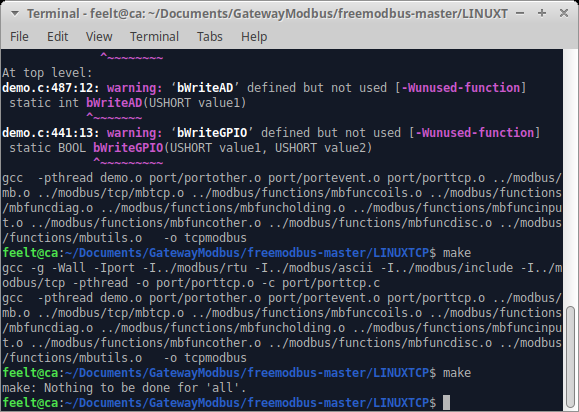
A instalação usei os passos deste link:

<https://www.ti-enxame.com/pt/linux/comando-gcc-arm-linux-gnueabi-nao-encontrado/1069884245/>

Agora o programa free-modbus compila no Linux. Para isso, basta ir na pasta do freemodbus (arquivo makefile) e então pedir para fazer um ***make***.



A janela abaixo mostra um exemplo de compilação.

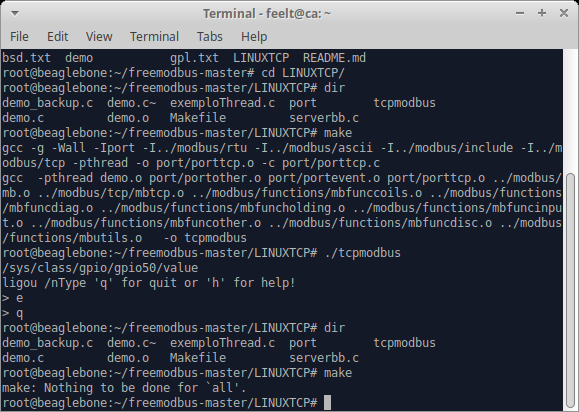


Obs: Ainda não fiz o teste de rodar o programa no PC e copiar o executável somente para a BBB.

O Marcio Cunha falou que eles utilizam para a raspberry o programa VisualCode que é mais fácil de utilizar pois ele compila e atualiza diretamente a placa.

## Compilando o programa na BBB

Após transferir os arquivos para a beaglebone, para compilar o código na própria BBB basta selecionar a pasta e executar o comando make (como foi feito no PC).



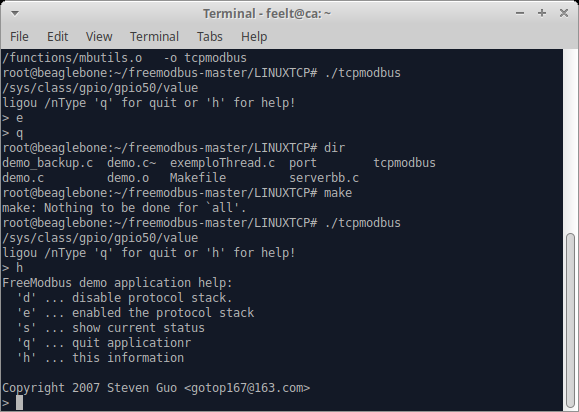
Para compilar > **make**

Para deletar a compilação anterior > **make clean**

**Make all**

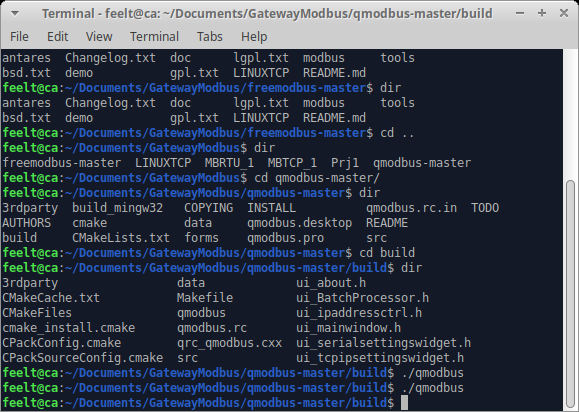
# Rodando o programa Modbus na BBB

Para rodar o programa na BBB basta executar o arquivo executável através do comando “./tcpmodbus”. Depois para rodar a tarefa modbus basta digitar “e” para executar. Existem outras opções que pode ser consultada no comando “h” para help.

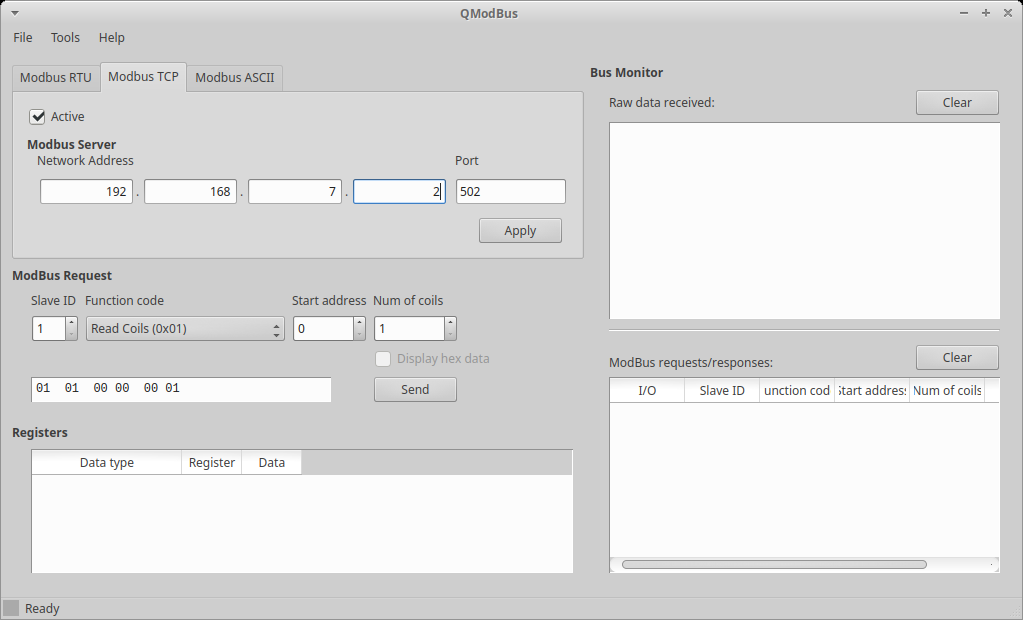


## Usando o programa qmodbus no Linux

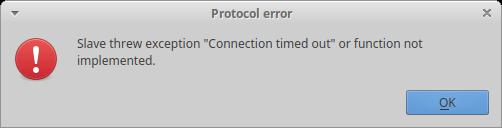
O programa qmodbus é similar ao ModScan. Ele funciona como um mestre modbus TCP ou RTU. Neste caso, para rodar o qmodbus basta entrar na pasta build e executar “./qmodbus”.



O programa eh bem intuitivo onde vc deve ativar “Active” a conexão TCP, digitar o IP selecionar os comandos e dar Apply. Caso os registros aparecerem na parte inferior da tela, significa que a leitura foi OK. Caso contrário ele mostrara uma mensagem de erro do protocolo.

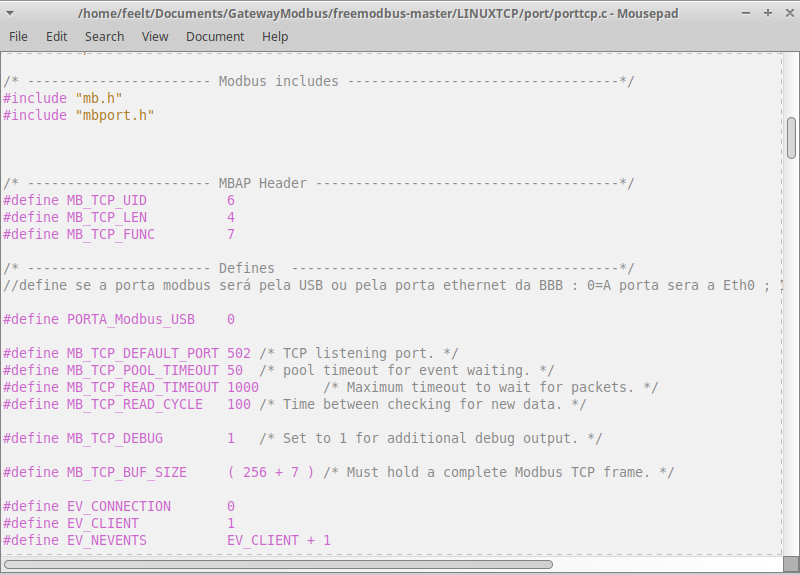


No caso, a conexão USB não estava disponível então ele vai enviar uma mensagem de erro conforme mostrado na figura abaixo



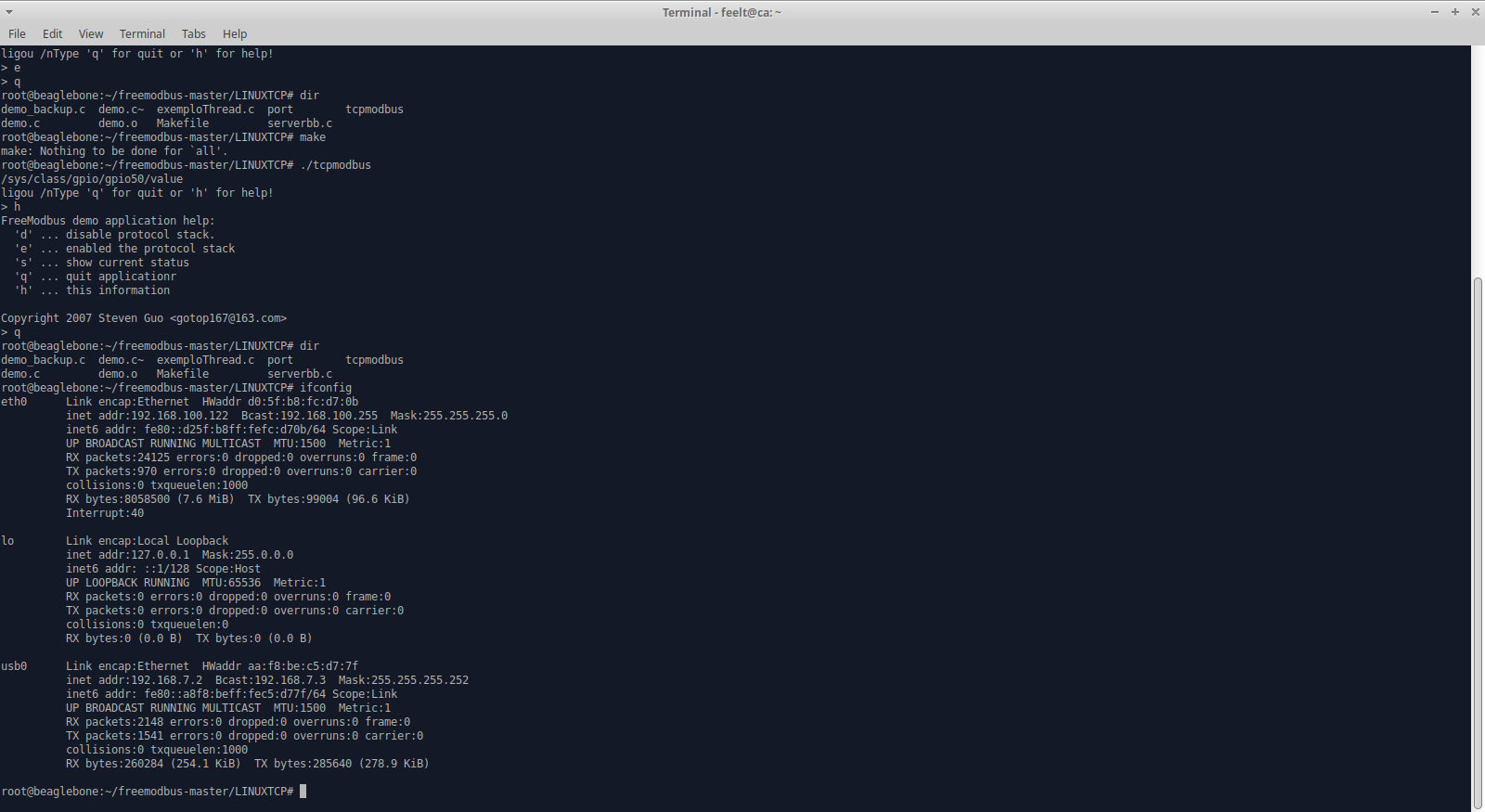
# Chaveando entre cabo ethernet ou porta USB

Dentro da BBB eh possível estabelecer conexão “Modbus TCP” tanto com o cabo serial USB quanto com a porta ethernet da BBB. Isto porque a BBB cria uma conexão TCP na porta USB. Desta forma, dentro do código eh possível escolher entre portas Ethernet ou porta USB. Isto eh feito através do define PORTA\_MODBUS\_USB no arquivo port\porttcp.c conforme mostrado na figura abaixo.



Caso for escolhido porta ethernet 0 para a BBB (PORTA\_Modbus\_USB =0), o código modbus vai responder somente na porta ethernet0. Caso o define for 1, a porta modbus vai ser a mesma da SSH (192.168.7.2 IP fixo).

No caso da ethernet0 é necessário setar o IP para o correspondente subrede do PC, como mostrado na figura abaixo:



O comando ***ifconfig*** no Linux mostra a configuração de todas as portas do device. Neste caso, é mostrado que a ***eth0*** tem ip : 192.168.100.122.

# Instalando o Eclipse Cross Compiler no Linux

O eclipse pode ser uma ferramenta mais fácil para trabalhar com o Linux e a beaglebone black.

(eu testei o Microsoft Visual Code no Windows) e ele não é compatível com a nossa beaglebone. Ele precisa ter um sistema SSH server e suporta a java que é a partir da versão 8 do Linux ou para a versão da raspberry. Desta forma parti para tentar fazer com o eclipse.

Os dados que estão aqui eu peguei do seguinte vídeo no youtube:

<https://www.youtube.com/watch?v=_SBCES64gYM> (install eclipse no linux)

<https://www.youtube.com/watch?v=qVNow-C2UZ8&t=622s> (cross compile beaglebone)

verificar java no linux :

> javac -version

Instalar java jdk no Linux (caso o comando acima der errado):

> sudo apt-get install default-jdk

Instalar o eclipse:

# Comunicando BBB com a internet

Para conseguir instalar “coisas” na BBB é necessário que ela tenha acesso na internet.

Uma forma de fazer isso eh “pingando” a internet dentro da BBB.

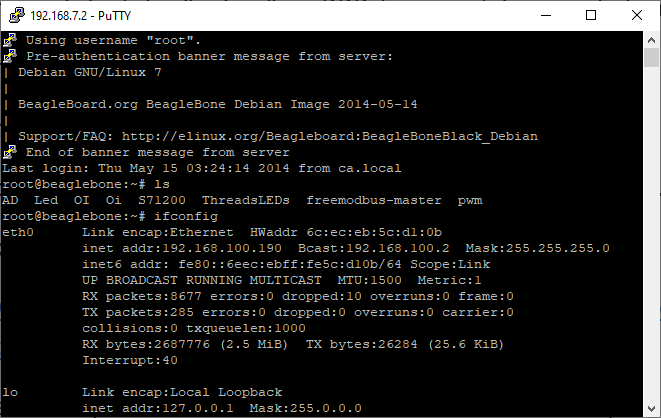
Por exemplo, ping 8.8.8.8 é pingar o servidor do google.

Dentro da maquina virtual eu não consegui fazer isso funcionar.

Este link da uma dica do que fazer

<https://www.digikey.com/en/maker/blogs/how-to-connect-a-beaglebone-black-to-the-internet-using-usb>

O que resolveu para mim foi no Windows conectar na BBB pelo Putty através da porta USB.



Dentro do putty é possível conectar a BBB e então checar um IP externo valido. Por exemplo o IP da porta eth0 era 192.168.100.190 e ela estava na mesma subnet do computador 192.168.100.99 e do roteador 192.168.100.1. Desta forma foi possível associar todos os comandos ethernet para vir para a porta do gateway com o seguinte comando:

**> sudo /sbin/route add default gw 192.168.100.1**

Após isso foi possível fazer o acesso a internet.

Bom, fiz uma conexão pela porta ethernet usando o putty (porta telnet). Ele não conseguia pingar o dns do google. Então, fiz o comando acima e consegui estabelecer a conexão com a BBB.

# Atualizando o relógio da BBB

Para verificar o relógio no Linux basta digitar o comando “date”.

Para atualizar o relógio da BBB tentei fazer varias coisas e não sei se tudo isso teve dependência do comando.

> sudo apt-get update (atualizar o Linux)

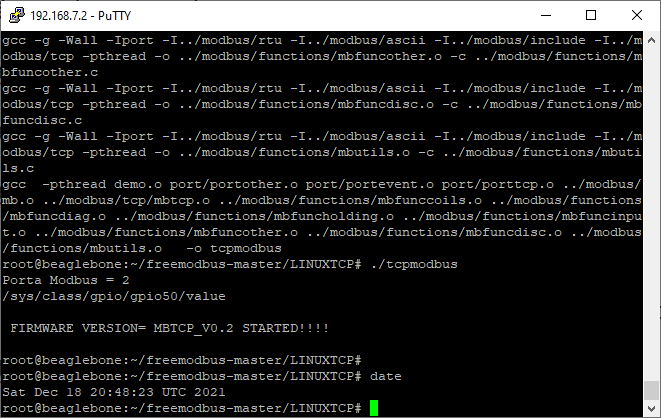
> sudo apt-get install locate (instalar o commando locate)

> sudo apt-get install ntp (instalar o servidor ntp)

Após isso foi feito o seguinte comando para atualizar o relógio do BBB:

**> ntpdate -b -s -u** [**pool.ntp.org**](http://pool.ntp.org/)

Depois pode utilizar o comando >**date**



# Configurando IP e NetMask

Antigo

ifconfig eth0 up 192.168.0.1/24

Novo

ip link set dev eth0 up && ip addr change dev eth0 192.168.0.1/24.

Ou de forma simplificada

ip l s eth0 up && ip a c dev eth0 192.168.0.1/24 netmask 255.255.255.0

**root@beaglebone:/etc# ip**

Usage: ip [ OPTIONS ] OBJECT { COMMAND | help }

ip [ -force ] -batch filename

where OBJECT := { link | addr | addrlabel | route | rule | neigh | ntable |

tunnel | tuntap | maddr | mroute | mrule | monitor | xfrm |

netns | l2tp }

OPTIONS := { -V[ersion] | -s[tatistics] | -d[etails] | -r[esolve] |

-f[amily] { inet | inet6 | ipx | dnet | link } |

-l[oops] { maximum-addr-flush-attempts } |

-o[neline] | -t[imestamp] | -b[atch] [filename] |

-rc[vbuf] [size]}

**root@beaglebone:/etc# ip route**

*default via 192.168.100.1 dev eth0*

*default via 192.168.7.1 dev usb0*

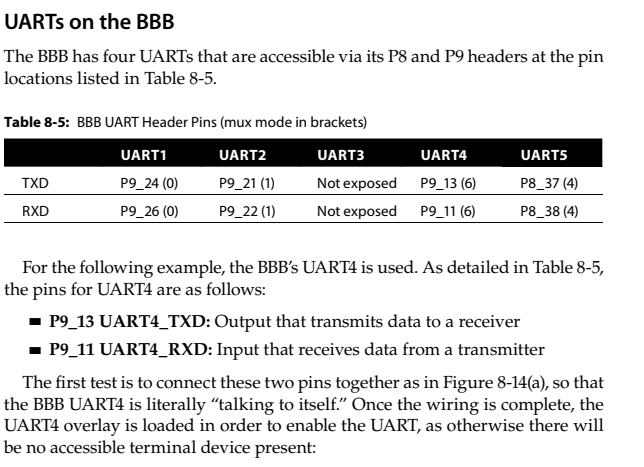
*192.168.7.0/30 dev usb0 proto kernel scope link src 192.168.7.2*

*192.168.100.0/24 dev eth0 proto kernel scope link src 192.168.100.190*

# Explorando a serial da BBB

Objetivos: este documento foi feito para tentar comunicar em um programa em linguagem C/C++ com uma das UARTs da BeagleBone Black. Este material foi montado baseado nos seguintes documentos: [1], [2], [3], [4], [5].

# Sobre a Serial UART da BBB [1]



* A BBB utiliza a biblioteca termios do Linux. Ver descrição [3] e [4].

Vou utilizar a UART4 conforme exemplo [1] e [2]. Para habilitar a UART4 fiz o seguinte comando [5] :

To enable UART1 and UART2 on Beaglebone Black Running Debian 8

* Step 1: sudo nano /etc/default/capemgr
* Step 2: Change #CAPE= to: CAPE=BB-UART1,BB-UART2
* Step 3: reboot
* Step 4: ls /dev/ttyO\*
* result: /dev/ttyO0 /dev/ttyO1 /dev/ttyO2

Entao instalei o programa picocom e então fiz o teste do loopback

<https://www.digi.com/resources/documentation/Digidocs/90001541/tasks/t_use_terminals.htm?TocPath=Get%20started%20with%20MicroPython%7C_____8>

Por fim, instalei um programa hyperterminal no Windows

<https://www.hilgraeve.com/hyperterminal-trial/>

E com a porta serial aberta consegui comunicar com ela

Na BBB eu enviei o seguinte comando

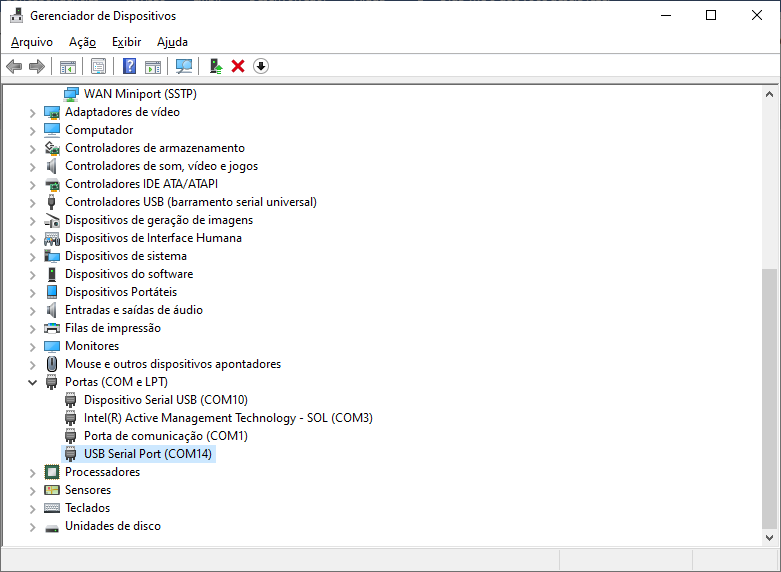
**picocom -b 115200 /dev/ttyUSB0**

**e ele me respondeu que a porta estava pronta para receber comandos. E todo comando que eu enviava aparecia no terminal do hyperterminal.**

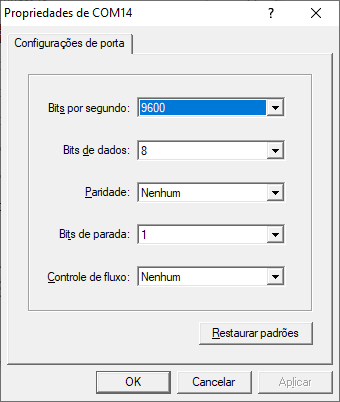
# Para fazer funcionar o programa Hyperterminal no Windows

O programa Hyperterminal é uma forma de ter um programa no lado do Windows que receba qualquer dado vindo da serial. Acho que poderia ser feito pela própria interface serial do Arduino.

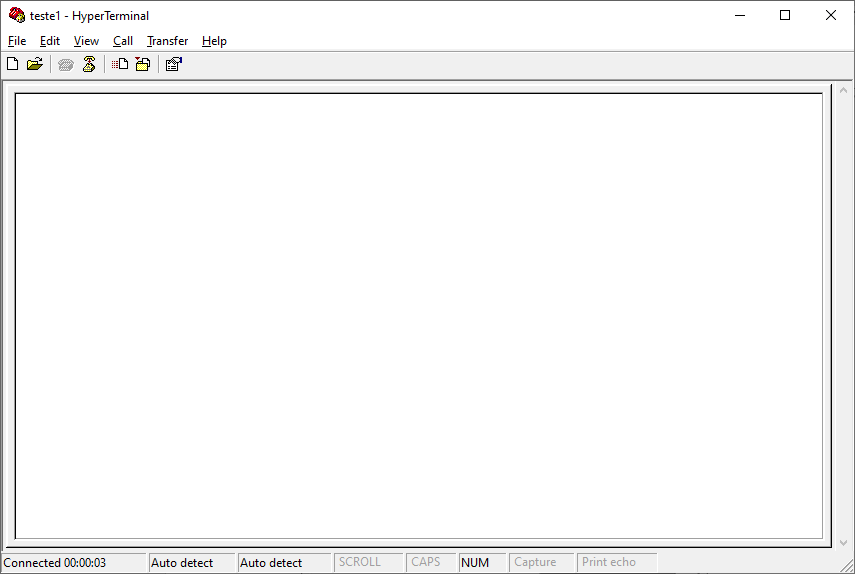
Primeiramente deveria ligar a interface FTDI no computador e descobrir qual a porta serial que ela está alocada. No meu caso foi na porta COM14 como mostrado na figura abaixo. (Paga isso vc pode ir na control panel do Windows e procurar por gerenciador de dispositivos.



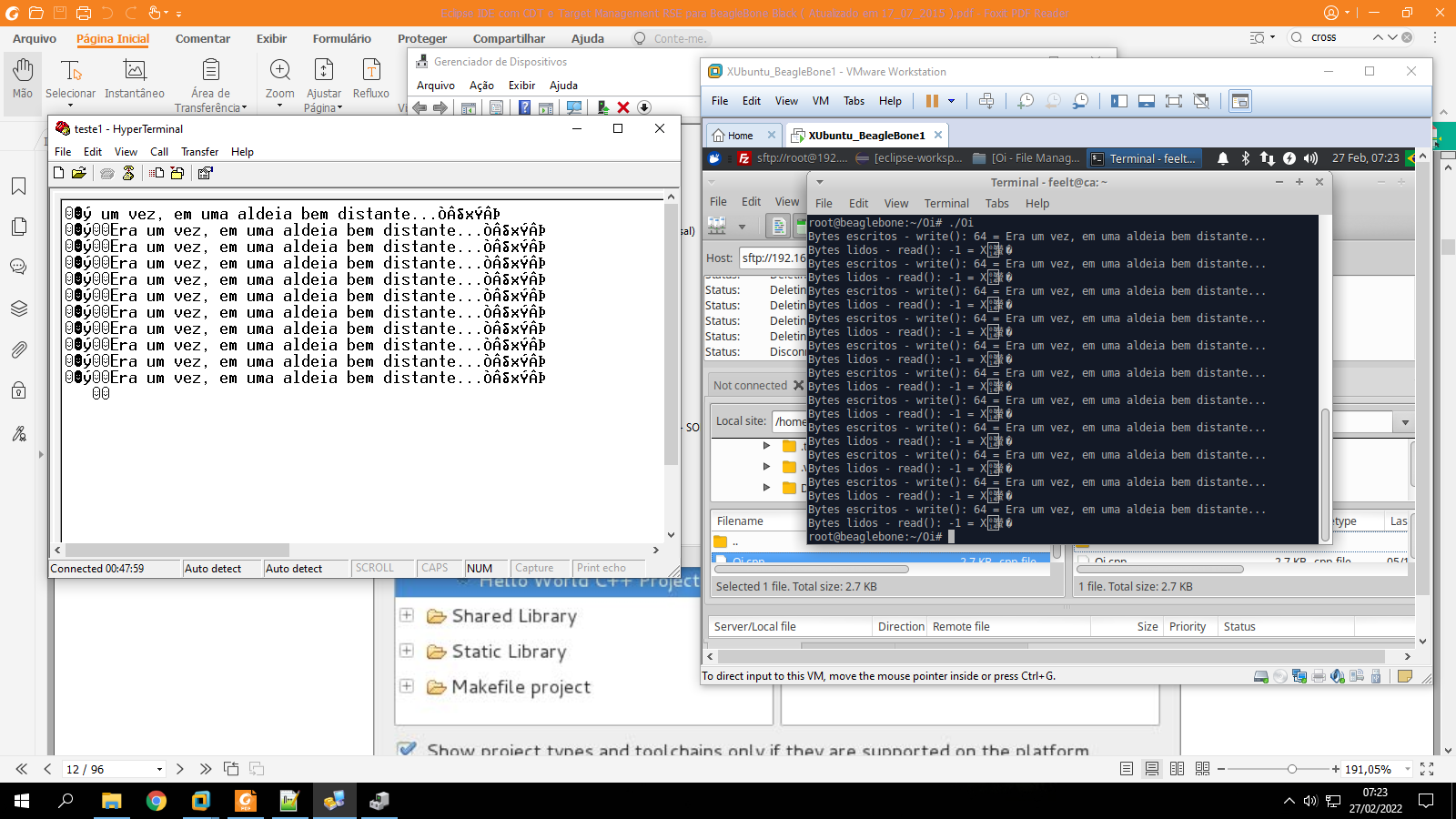
Então você deve configurar o programa Hyperterminal com as mesmas configurações de Hardware que vc configurou na BBB. No meu caso foi :baudrate 9600/8bitsdados/Paridade None/Sem RTS/CTS.



Então o programa vai ficar escutando a serial da COM14 esperando por um dado ser enviado.



Após rodar o programa na BBB ele ficou assim. A tela da direita em preto é o envio das mensagens e na tela da esquerda é a recepção das mensagens. O programa é exatamente aquele mostrado na referencia [2].



Fiz o mesmo teste com a interface serial do Arduino e funcionou perfeitamente.

# Trabalhando com Timers na BBB

A **Real-time Programable Unit** e o Subsistema de Comunicação Industrial (PRU-ICSS) no processador AM335x da BBB contém dois RISC de 32 bits e 200 MHz núcleos, chamados PRUs. Essas PRUs têm sua própria alocação de memória local, mas eles também podem usar os pinos de cabeçalho BBB P8/P9 e acionar interrupções e compartilhar memória com o dispositivo host Linux. O PRU-ICSS é uma plataforma Linux embarcada geral, uma vez que pode fornecer suporte para aplicativos de interface que têm dificuldade em tempo real restrições. Ele pode ser usado para manipular entradas, saídas e estruturas de dados mapeadas em memória para implementar interfaces de comunicação personalizadas (por exemplo, manipulação de E/S simples, troca de bits, SPI, UARTs). A RPU é mais complexa e precisa de uma compilação especial para habilitá-la.

Artigo muito bom sobre RPU :

<http://www.righto.com/2016/09/how-to-run-c-programs-on-beaglebones.html>

ou o capitulo 13 pag 503 do livro [1].

Para programar um programa em PRU (PRU code generation tools – compiler)

<https://www.ti.com/tool/PRU-CGT>

<https://www.ti.com/tool/PROCESSOR-SDK-AM335X>

Delay\_cycles Somente cria um delay com base em um timer de 5ns.

\_\_delay\_cycles(60000000); // Intrinsic method delay

Uma outra biblioteca que vi que trabalharia com timers seria a DL\_timer que precisa de alguns módulos do Linux. Segue um código exemplo usando esta biblioteca. Repare que ela precisa de uma biblioteca do próprio Linux onde não consegui fazer ela funcionar ou instalar na BBB minha atual.

I have been able to get the DM Timer module to generate a signal change for an exact time period. I've included my code below and changes to my device tree.

C Code

#include <linux/module.h> /\* Needed by all Linux Kernel Modules \*/

#include <linux/platform\_device.h> /\* Needed to query the plaform configuration for addresses, interrupts,etc \*/

#include <linux/kernel.h> /\* Needed for KERN\_INFO \*/

#include <linux/init.h> /\* Needed for the module\_ macros \*/

/\* ARM DM Timer Header \*/

#include <plat/dmtimer.h>

/\* Needed for clk functions \*/

#include <linux/clk.h>

/\* Interrupt Header \*/

#include <linux/interrupt.h>

#define DRIVER\_AUTHOR "Sean Moffatt"

#define DRIVER\_DESC "Test DM Timer"

#define DRIVER\_NAME "test\_module\_timer"

static struct omap\_dm\_timer \*timer\_ptr = NULL;

static int32\_t timer\_irq;

static uint32\_t timer\_rate;

static irqreturn\_t timer\_irq\_handler( int irq, void \* dev\_id)

{

int status = 0;

/\* Read the current Status \*/

status = omap\_dm\_timer\_read\_status(timer\_ptr);

/\* Clear the timer interrupt \*/

if (status == OMAP\_TIMER\_INT\_MATCH)

{

omap\_dm\_timer\_write\_status(timer\_ptr, OMAP\_TIMER\_INT\_MATCH);

}

/\* Stop the Timer \*/

omap\_dm\_timer\_stop(timer\_ptr);

/\* Indicate the Interrupt was handled \*/

return IRQ\_HANDLED;

}

static int test\_module\_init(struct platform\_device \*pdev)

{

int ret = 1;

struct clk \*timer\_clk;

/\* Need to request OMAP\_TIMER\_HAS\_PWM to ensure we have an I/O Pin to toggle \*/

/\* Still need to determine how to request a specific Timer\*/

timer\_ptr = omap\_dm\_timer\_request\_by\_cap(OMAP\_TIMER\_HAS\_PWM);

if(timer\_ptr == NULL){

/\* no timers available \*/

printk(KERN\_INFO "No more DM timers available!!\n");

return -1;

}

/\* Set the Clock source to the System Clock \*/

ret = omap\_dm\_timer\_set\_source(timer\_ptr, OMAP\_TIMER\_SRC\_SYS\_CLK);

/\* Determine what IRQ the timer triggers \*/

timer\_irq = omap\_dm\_timer\_get\_irq(timer\_ptr);

/\* Setup the IRQ handler \*/

ret = request\_irq(timer\_irq, timer\_irq\_handler, IRQF\_TIMER, DRIVER\_NAME, NULL);

/\* Setup the timer to trigger the IRQ on the match event \*/

omap\_dm\_timer\_set\_int\_enable(timer\_ptr, OMAP\_TIMER\_INT\_MATCH);

/\* Get the Clock rate in Hz \*/

timer\_clk = omap\_dm\_timer\_get\_fclk(timer\_ptr);

timer\_rate = clk\_get\_rate(timer\_clk);

/\* Setup the Output for the Timer pin \*/

/\* Setup to toggle on the overflow and the compare match event \*/

omap\_dm\_timer\_set\_pwm(timer\_ptr, 0, 1, OMAP\_TIMER\_TRIGGER\_OVERFLOW\_AND\_COMPARE);

/\* Enable the Timer \*/

/\* Needs to be done before we can write to the counter \*/

omap\_dm\_timer\_enable(timer\_ptr);

/\* Set the initial Count \*/

/\* According to section 20.1.3.5 Pulse-Width Modulation, an overflow or match be used to toggle when a compare condition occurs\*/

/\* Therefore it we will trigger the overflow event almost immediately to ensure our toggle will be generated quickly \*/

omap\_dm\_timer\_write\_counter(timer\_ptr, (0xFFFFFFFF - 5));

/\* Setup the Load Register \*/

/\* Setup as autoload to set the the counter back to 0 on an overflow \*/

omap\_dm\_timer\_set\_load(timer\_ptr, 1, 0);

/\* Set the the compare register to 1 second \*/

/\* This will ensure the signal toggle of 1 second after the overflow event\*/

omap\_dm\_timer\_set\_match(timer\_ptr, 1, timer\_rate);

/\* Start the timer \*/

ret = omap\_dm\_timer\_start(timer\_ptr);

printk(KERN\_INFO "Timer Start ret %d\n", ret);

/\*

\* A non 0 returns means the call to init\_modules failed; module can't be loaded

\*/

return 0;

}

static int test\_module\_exit(struct platform\_device \*pdev)

{

int ret = 1;

/\* stop the timer \*/

ret = omap\_dm\_timer\_stop(timer\_ptr);

printk(KERN\_INFO "Timer Stop ret = %d\n", ret);

/\* Release the IRQ handler \*/

free\_irq(timer\_irq, NULL);

printk(KERN\_INFO "Free IRQ Done\n");

/\* Release the timer \*/

ret = omap\_dm\_timer\_free(timer\_ptr);

printk(KERN\_INFO "Timer Free ret = %d\n", ret);

return 0;

}

static const struct of\_device\_id test\_module\_timer\_of\_match[] = {

{ .compatible = "test,am33xx-test\_module\_timer" },

{},

};

static struct platform\_driver test\_module\_timer\_driver = {

.driver = {

.name = DRIVER\_NAME,

.owner = THIS\_MODULE,

.of\_match\_table = test\_module\_timer\_of\_match,

},

.probe = test\_module\_init,

.remove = test\_module\_exit,

};

module\_platform\_driver(test\_module\_timer\_driver);

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR(DRIVER\_AUTHOR); /\* Who wrote this module \*/

MODULE\_DESCRIPTION(DRIVER\_DESC); /\* What is this module \*/

# Código Gateway Modbus TCP to RTU em Python

Este código [6] é ja uma implementação de um Gateway modbus TCP para RTU em python. A descrição do funcionamento segue abaixo.

Modbus TCP to Modbus RTU gateway

[![License: CC BY-NC 4.0](https://img.shields.io/badge/License-CC%20BY--NC%204.0-lightgrey.svg)](http://creativecommons.org/licenses/by-nc/4.0/)

This is my Modbus TCP to Modbus RTU gateway written in Python.

I run it on a [BeagleBone Green](http://beagleboard.org/Green) equiped with a [Waveshare CAN RS485 Cape] (www.waveshare.com/wiki/RS485\_CAN\_CAPE) to communicate with my centralized ventilation system.

I've only tested Modbus functions [01,02,03,04,06] so far because I don't need any other functions for my purpose. Maybe the gateway need to be exteded to support other functions.

The gateway receives a ModbusTCP frame, translates it into a ModbusRTU frame, takes the ModbusRTU response and converts that into the ModbusTCP response for the initial request.

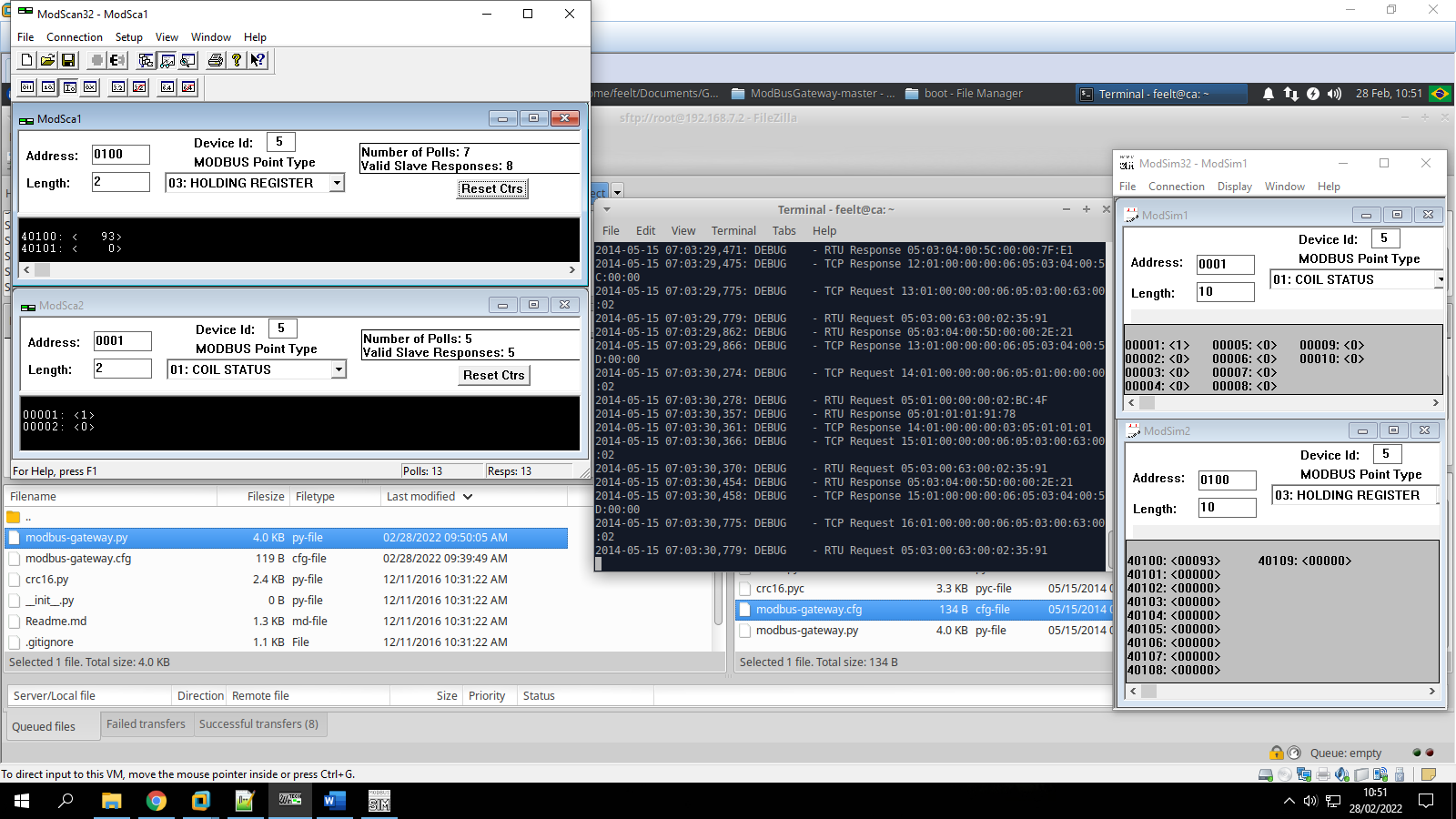
The configuration can be changed by editing the modbus-gateway.cfg file.

Procedimento:dir

* Aqui basta somente downlodar a pasta para a BBB
* Entrar dentro da pasta com os arquivos py
* Editar a configuração junto ao arquivo modbus-gateway.cfg (obs.: o HOST eh o IP da beaglebone black)
* E rodar o código em python :
* **python modbus-gateway.py**

**Resultado:**

Fiz o seguinte teste, coloquei um Modscan lendo da BBB e o Modsim deveria ser configurado do outro lado.



# Referências

[1] - Livro do Fabio Exploring Beaglebone (pagina 310)

[2] - Apostila do Fabio xxx ( pag 73)

[3] - <https://en.wikibooks.org/wiki/Serial_Programming/termios>

[4] <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/tty/serial?id=4a0ac0f55b18dc297a87a85417fcf068658bf103>

[5] <https://stackoverflow.com/questions/35881761/enabling-uart-on-beaglebone-black>

[6] Gateway Python - <https://github.com/Bouni/ModBusGateway>