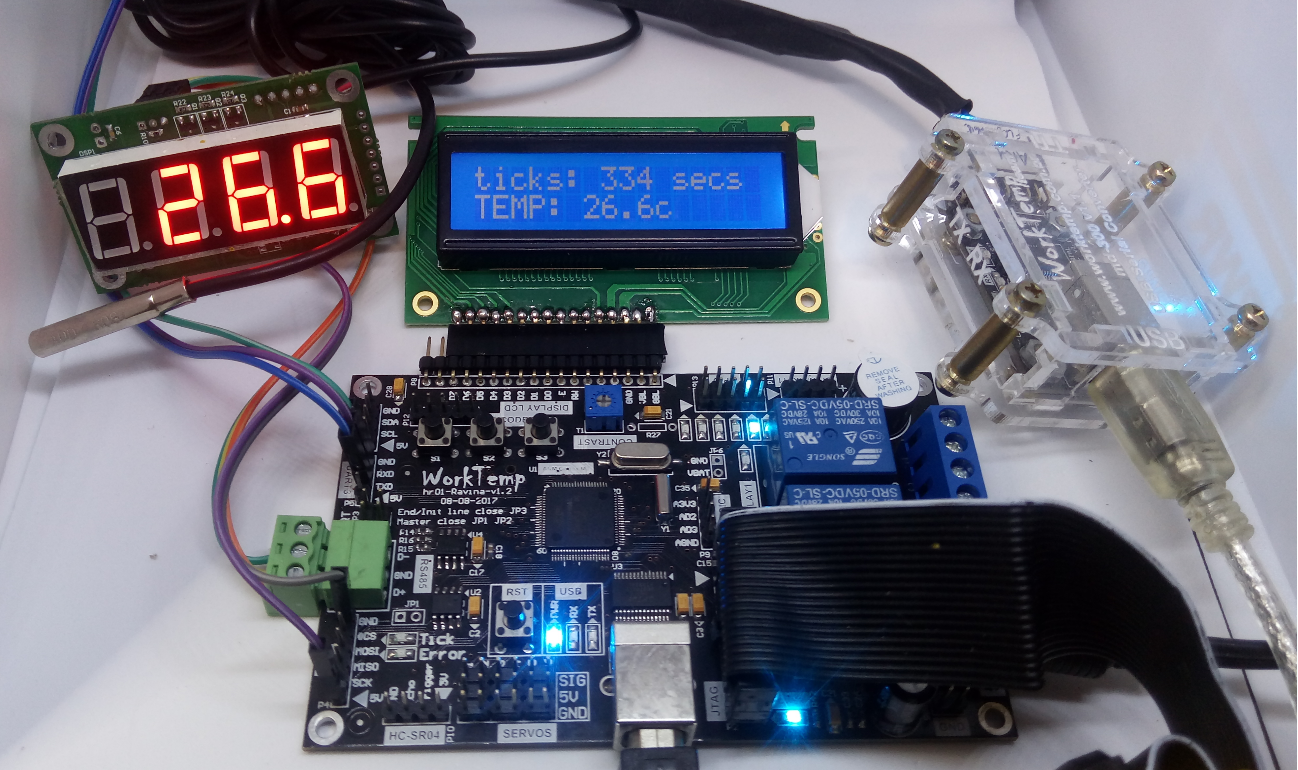
**Modbus Master e Slave**

Neste exemplo mostra como usar a comunicação modbus tanto no modo mestre com nodo escravo no kit ARM. O modo mestre o kit ARM faz acessos a um termômetro digital e mostra o valor da temperatura no display LCD. No modo escravo um terminal no PC vai acessar o kit ARM e coletar as informações e exibir em uma janela. Neste exemplo estamos acessando um só termômetro, mas a estrutura do sistema permite acessar muitos termômetros no mesmo barramento, na prática foi feito até nove termômetros. Ainda reutilizando a mesma estrutura do sistema podemos adaptar para acessar diferentes tipos de dispositivos no mesmo barramento.

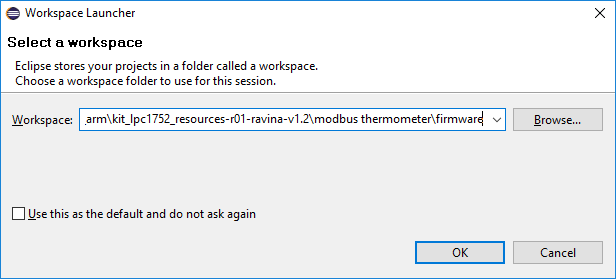


**Conexão com o Kit ARM**

O termômetro digital é conectado à porta rs485 do kit ARM, esta comunicação é feita via UART1 do ARM. A comunicação com o PC se faz via porta UART3 do kit, então deve conectar um cabo serial entre o PC e o conector P5 do kit, podemos usar um conversor USB para serial para esse propósito sem a interface rs232.

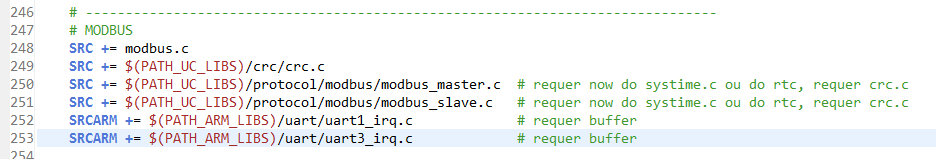
**Programa Exemplo**

Abra o exemplo da pasta “modbus thermometer\firmware” com o Eclipse:

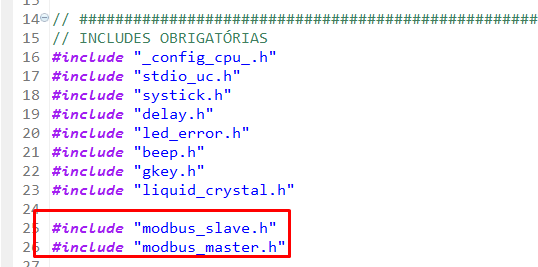


No makefile, na secção de inserção de fontes, foi incluindo as seguintes fontes:

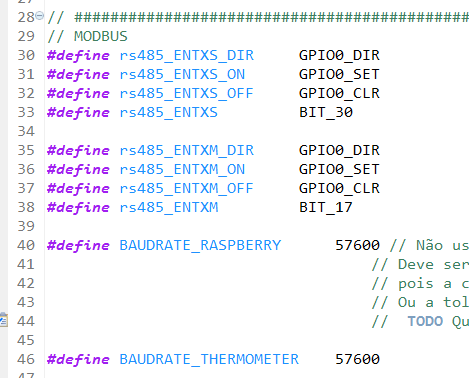
* modbus.c: Gerenciador de comunicação com o PC e com o termômetro;
* crc.c: O protocolo modbus requer CRC nos tráfegos de dados;
* modbus\_master.c: Controlador mestre modbus;
* modbus\_slave.c: Controlador escravo modbus;
* uart1\_irq.c: Porta de comunicação para mestre modbus;
* uart3\_irq.c: Porta de comunicação para escravo modbus.



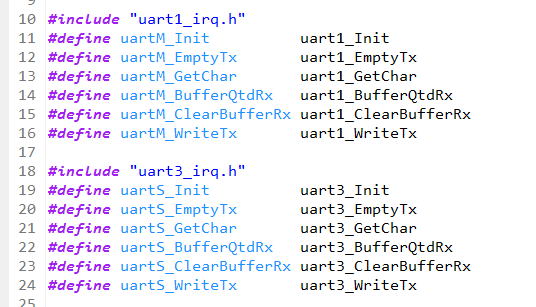
Em app.h foi adicionado os includes das gerenciadores modbus para uso na aplicação:



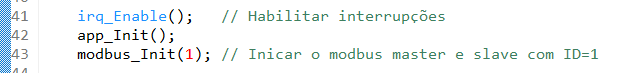
Em app.h é feito as configurações das velocidades de comunicação das portas UARTs tanto para o gerenciador mestre como escravo, e as definições dos pinos de inversão dos barramentos:



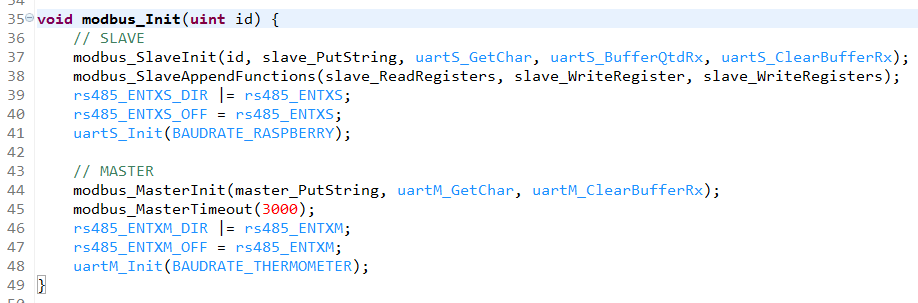
No início do arquivo modbuc.c especificamos quais as portas UARTs serão usadas para o modbus master e slave:



Em main.c dentro do int main(void), é inicializado os gerenciadores modbus com a função modbus\_Init, e o paramento determina o endereço do kit ARM no barramento modbus quando este trabalha em modo escravo, onde vai ser acessado pelo PC:



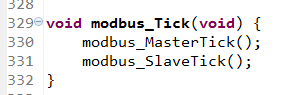
O modbus\_Init inicializa respectivamente os gerenciadores modbus escravo e mestre, configura os pinos de controle de inversão de barramento e inicializa as portas de comunicações UARTs:



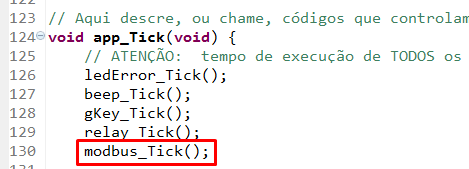
Na inicialização do barramento escravo anexamos as funções de veículo de dados, que neste caso são as funções da UART3, e anexamos as três funções de leituras e escritas em registradores modbus que vão tratar a comunicação.

Na inicialização do barramento mestre anexamos as funções de veículo de dados, que neste caso são as funções da UART1, e determinamos o tempo de espera por uma resposta do escravo, que neste caso é de 3 segundos.

Os gerenciadores do modbus requerem procedimentos que sejam executados em tempo reguladores e precisos, logo é necessário adicionar modbus\_MasterTick e modbus\_SlaveTick ao tick do sistema. Neste caso ambas funções foram colocadas em modbus\_Tick:



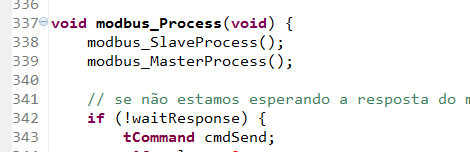
Por sua vez a função modbus\_Tick é chamada pelo app\_Tick, e esta chama todas as funções ticks do sistema:



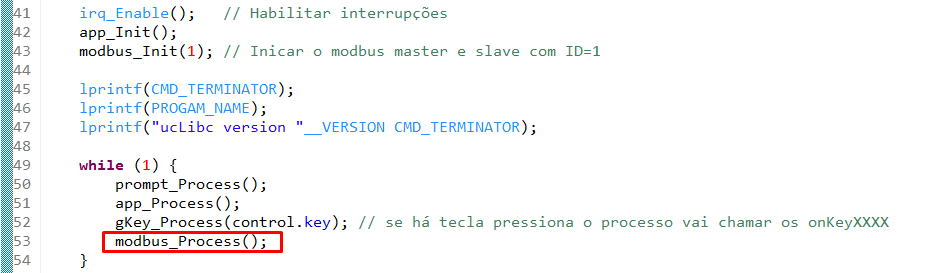
A função app\_Tick é chamada a cada 1ms, isto foi especificado no início do programa na função main:



Todos os acessos modbus sejam do kit ao termômetro, ou do PC para o kit, são feitos pelo modbus\_Processo no modbus.c.



O modbus\_Process é chamado em loop infinito na função main junto com outros processos:



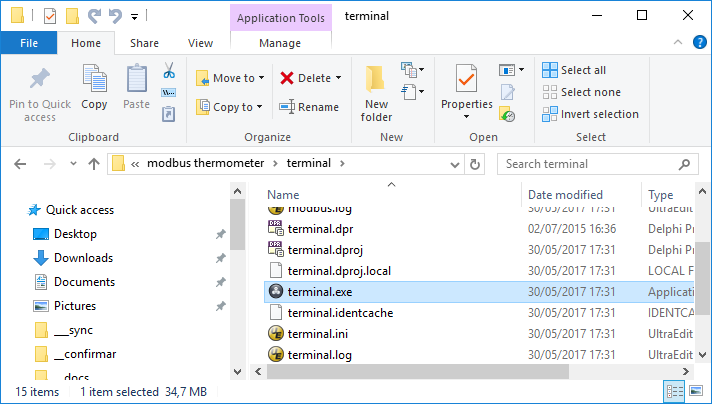
Em modbus\_Process além de chamar as funções processos dos gerenciadores de modbus, também faz a comunicação com o termômetro, por ser o mestre do barramento, sempre está iniciando a comunicação e esperando pela resposta. Caso receber uma resposta do escravo, o valor é gravado em um local correto, ou se houver silencio no barramento ou houver algum tipo de erro, um tratamento adequado será feito.

Dentro do modbus.c contém funções de acessos aos registradores modbus para serem acessados por um mestre quando solicitado. Ou seja, o kit ARM vai ser comportar como escravo para o PC, e dependo dos comandos recebidos, o kit vai responder com suas funções de acesso aos registradores. Essas funções são: id e modelo do kit ARM; condição de execução de comandos; e valores de temperatura dos termômetros.

**Compile e Execute**

Com tudo instalado e configurado, compile a aplicação, grave e execute no ARM. Já no início da execução do firmware o valor da temperatura será exibido no display LCD.

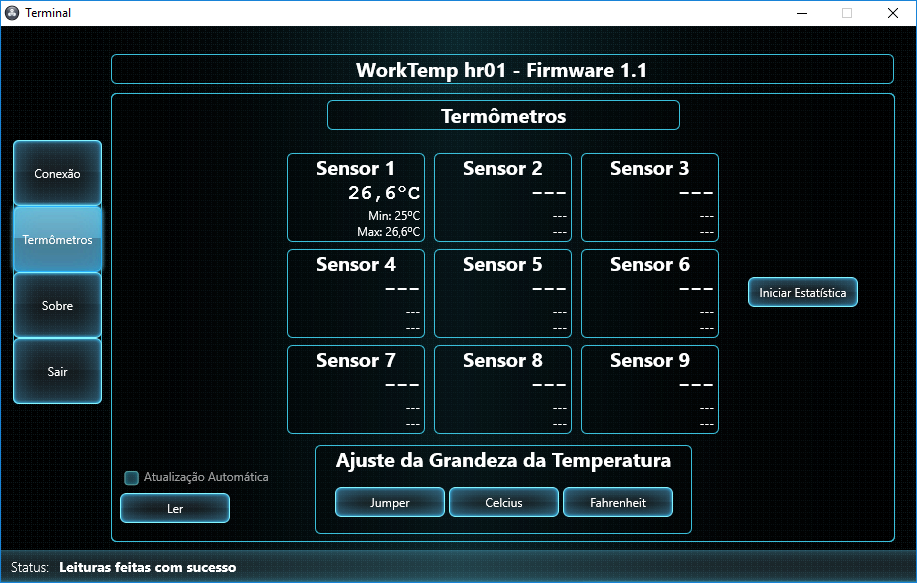
No PC há terminal exemplo que possa ser usado para acessar o kit ARM e coletar os valores dos termômetros.



Na primeira janela entramos com a porta COM que será feita a comunicação com o kit ARM, não confunda com a porta COM de acesso ao CLI do kit. Entramos com o ID do kit, este ID é o endereço de acesso do dispositivo no barramento, e por último determinamos o tempo de espera por respostas do kit ARM no barramento. Depois de entrar com esses parâmetros clique em conectar:



Se houver algum erro o mesmo vai ser impresso no log de eventos, se a comunicação com o kit foi feita com sucesso, o botão termômetros estará habilitado, clique nele para ir para a janela de leituras. Quando ir à janela de leitura o terminal envia comandos de leituras dos valores dos termômetros para o kit:



Se desejar que a leitura seja de forma continua, habilite a opção de atualização automática, senão clique no botão ler sempre que deseja atualizar os valores de temperaturas.

Em ajuste de grandezas cada botão envia um comando de mudanças das escalas de medição para os termômetros. O botão jumper sinaliza aos termômetros assumirem a grandeza de acordo com os seus jumpers de configurações.

O botão iniciar estatísticas limpa os registros de temperatura máxima e mínimas nos termômetros.