Configurações para uso do PCOM WiFi/Bluetooth em placas ESP32 na Plataforma Arduino

revisão-00, agosto de 2019

Elcids H. das Chagas

Obs.:

- ***este documento foi escrito considerando a versão 1.8.8 da IDE do Arduino, e a versão 1.0.2 do ESP32 Espressif Pack para Arduino.***

***Funcionalidades básicas do PCOM WiFi-Bluetooth*:**

O ***PCOM WiFi/Bluetooh*** é um mecanismo que gerencia as conexões WiFi e Bluetooth para um código do usuário executando em uma ***placa ESP32*** na ***Plataforma Arduino***. Este gerenciamento disponibiliza uma interface para o código do usuário, que é capaz das seguintes funcionalidades:

1) Gerenciar automaticamente a conexão do ESP32 à rede WiFi especificada, e mesmo que esta rede caia ou fique temporariamente indisponível, o PCOM será capaz de reconectar assim que a rede especificada estiver novamente ao alcance.

2) Instanciar um ***WiFi Server***, para que este sirva como elemento de comunicação entre o código do usuário e um ***Cliente WiFi***. As funcionalidades da comunicação são as mesmas existentes no WiFi convencional do ESP32 para a Plataforma Arduino.

3) Disponibilizar um ***Mecanismo de Gerenciamento da Comunicação*** com o Cliente WiFi, funcionando com uma Interface entre ambos, simplificando e padronizando a forma de acesso no código. Este mecanismo pode ser utilizado para gerenciar inúmeras ***Páginas HTML*** (normalmente em uma comunicação com um Navegador de Internet), ou então utilizando qualquer protocolo sobre o ***TCP/IP*** (seja um protocolo padrão ou dedicado). Um mecanismo de ***Timeout*** simples também está implementado, para o caso do Cliente WiFi parar de responder e não completar uma requisição em andamento.

4) Gerenciar automaticamente a conexão do ESP32 a um ***Cliente Bluetooth*** "***Classic***", quando o WiFi não estiver disponível, permitindo que este Cliente possa acessar as mesmas funcionalidades implementadas no Sistema da placa ESP32 que podem ser acessadas através do WiFi.

5) Instanciar uma ***Interface Bluetooth SPP*** (*Serial Port Profile*), para ser utilizada como meio de comunicação entre o Cliente Bluetooth e o código executando no Arduino/ESP32.

6) Gerenciar um ***Mecanismo de Autenticação*** para o Cliente Bluetooth, mantendo uma ***Lista de Clientes Autenticados***. A Lista é não-volátil, sendo armazenada na EEPROM do ESP32.

As funcionalidades listadas são as básicas, e poderão ser ampliadas em revisões da implementação do ***PCOM WiFi/Bluetooth***.

Não há obrigatoriedade em se usar ambos os PCOMs WiFi e Bluetooth. O código do Arduino pode sistematicamente habilitar/desabilitar um ou outro PCOM (ou ambos), a qualquer momento. O default é ambos habilitados, mas caso se use apenas um dos PCOMs, nenhuma ação é necessária, já que o Mecanismo de conexão é automático e utilizará a conexão que estiver disponível e ao alcance.

Este é um documento simples, cujo objetivo é mostrar e descrever algumas opções para uma placa ESP32 na IDE do Arduino, a fim de que resulte num código compilado sem problemas com espaço de Memória na placa, quando se utiliza o PCOM WiFi/Bluetooth. Também são apresentadas as vantagens e desvantagens para as opções abordadas, mostrando-se como isso pode afetar eventuais revisões no código do Sistema implementado.

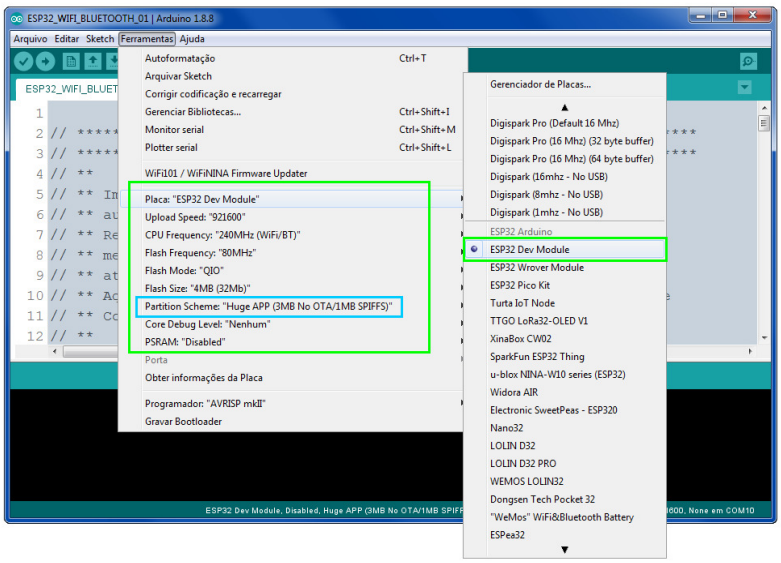
A questão do espaço de Memória, se refere essencialmente à ***Memória de Programa***, uma vez que a implementação do WiFi e Bluetooth atualmente disponibilizada pela ***Espressif*** (fabricante original do ESP32), tem um consumo bastante alto da área da Memória de Programa (provavelmente devido à utilização de estruturas/objetos grandes em C/C++ e à grande quantidade de perfis atualmente existentes no Bluetooth).

Então dada esta problemática atual da Memória de Programa disponível nas placas do ESP32, tornam-se necessários alguns cuidados na seleção de opções na configuração das placas, tendo-se também ciência do impacto de cada uma das opções selecionadas.

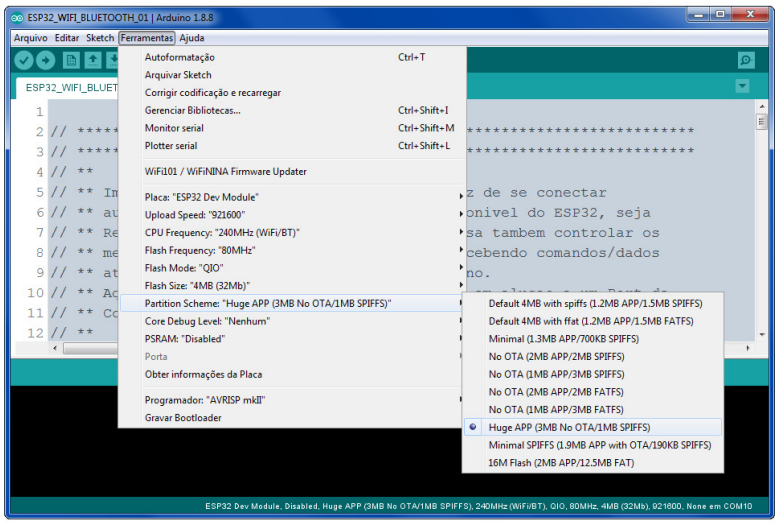
Como o ESP32 utiliza chips de Memória com interface serial (estes chips sempre possuem a mesma Interface de Hardware independente da capacidade da memória), para o armazenamento de código permanente, então eventualmente a questão do espaço de Memória de Programa disponível para o código do usuário, não será mais um problema. Isto porque já existem chips de Memória serial com capacidades muito além do que é hoje utilizado na maioria das placas do ESP32, e devido às características da Interface de Hardware destes chips, o Hardware dessas placas não precisaria ser revisado para que tais chips de Memória fossem utilizados (provavelmente isto ainda não é amplamente feito devido à relação entre o custo de produção e a popularidade das placas no mercado).

Seleção/análise das opções para uma placa ESP32:

Embora se possa selecionar especificamente a placa ESP32 que se está utilizando, é aconselhável selecionar a placa "genérica", que corresponde à opção "***ESP32 Dev Module***", pois esta opção permite especificar algumas características interessantes para a compilação. Para isto, basta ir no Menu "***Ferramentas***" e selecionar a placa, conforme mostrado na figura a seguir:



Observar que além do modelo da placa, também está selecionada a opção ***Partition Scheme"Huge APP..."***, referente ao uso (ou esquema) da ***Memória de Programa*** do ESP32. É possível escolher diversas opções para o esquema de Memória. Estas opções são mostradas na figura a seguir:

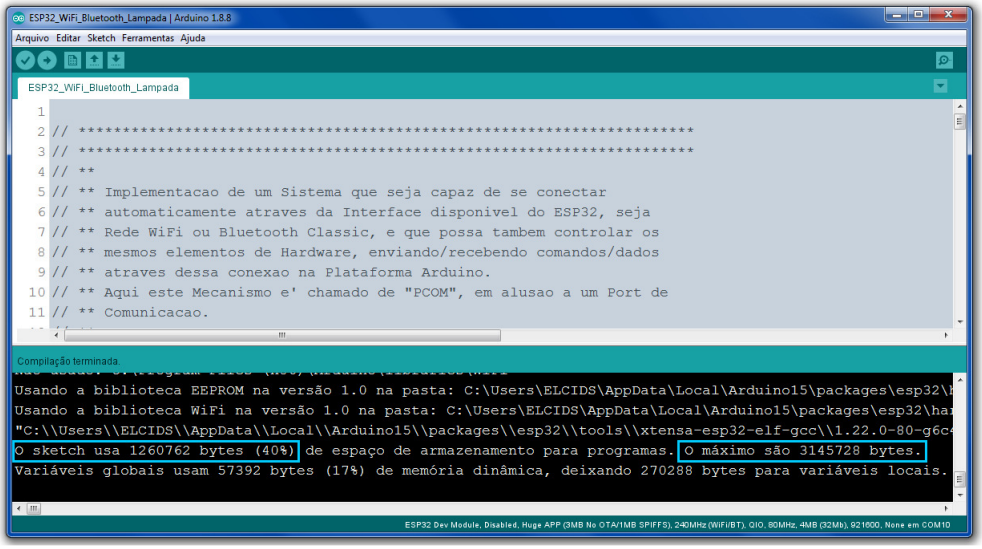


O esquema de Memória permite que se escolha como a ***Memória de Programa*** será utilizada. Por exemplo, se o Sistema utiliza o ***OTA*** (atualização de código via WiFi), será necessário escolher um esquema de Memória que inclua o OTA. Vamos entender melhor o porquê disto: ocorre que o Mecanismo básico do OTA, ocupa uma certa área na Memória de Programa, e não há como reduzir essa área, pois é o código básico da ***Biblioteca OTA*** para a ***Plataforma Arduino***. Logo, se o OTA ocupa uma certa área da Memória, o que "sobra" para se utilizar no código do Arduino, será o total da Memória de Programa do ESP32 menos a área ocupada pelo OTA.

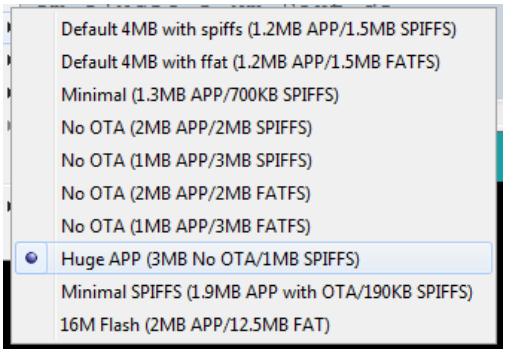
A questão é que diversos recursos do ESP32 que se utiliza comumente (além do OTA), também ocupam área na Memória de Programa. Por exemplo o ***Sistema de Arquivos FATFS*** (onde além do próprio código do FATFS, também precisa de uma área para armazenar os próprios Arquivos).

Assim, é preciso escolher o esquema de Memória que melhor se encaixa para cada aplicação, considerando-se todos os recursos do ESP32 que se está utilizando.

Obviamente nem sempre é fácil se escolher o esquema de Memória mais adequado para cada caso. Mas também, para cada modelo de placa, não existe uma enorme quantidade de opções pra se escolher. Então procura-se usar o bom censo ao se escolher o esquema de Memória, e compila-se para este esquema. Ao final da compilação, será mostrado quanto o código está utilizando da Memória de Programa, como pode ser visto no exemplo da figura a seguir para o esquema de Memória ***Partition Scheme"Huge APP..."***:



Analisando estes números: o resultado diz que o código ocupa ***1260762 Bytes*** da ***Memória de Programa***, e que isto corresponde a ***40%*** do máximo disponível de ***3145728 Bytes***. Observe que neste esquema de memória, ***3MBytes*** são disponíveis (os 3145728 Bytes), e que se pode também usar ***1MByte*** para o ***Sistema de Arquivos SPIFFS***, mas *sem* o ***OTA***, conforme na figura a seguir:



Este esquema de Memória, totaliza ***4Mbytes***, justamente os **3MB**+**1MB**. A maioria das placas do ESP32 disponíveis no mercado, possuem um total de 4MBytes de Memória de Programa, por isso a opção de placa "***ESP32 Dev Module***", considera que existem 4MB disponíveis. E conforme o esquema de Memória que se escolha, irá "sobrar" uma certa quantidade para o código do usuário. O importante aqui, é o *percentual correspondente ao código atual compilado*. No exemplo mostrado isto é ***40%***. Logo, neste esquema de Memória, do total reservado para o código, ainda se tem ***60% disponíveis*** para se usar, o que dá uma imensa flexibilidade caso posteriormente seja necessária alguma alteração no código (revisões, as quais são muitas vezes inevitáveis, ou então o acréscimo de novas características/funcionalidades no Sistema).

Consideremos por exemplo, que um determinado esquema escolhido resulte no uso de ***95%*** do total disponível. Neste caso, se tem apenas ***5%*** "sobrando" para ser usado caso se faça uma posterior alteração no código. Ou seja, é uma margem bem estreita para alterações futuras. Então deve-se tomar a decisão de se usar este esquema que "sobra" apenas 5%, ou então alterar o esquema de Memória para outro, que resulte em um percentual de "manobra" mais confortável. Eventualmente, é preciso abrir mão de recursos para se conseguir isso. Por exemplo: é fácil "amar" o OTA, mas no Sistema sendo implementado, o OTA é realmente necessário? Ou será que existe uma possibilidade de se usar o OTA, mas é mínima e improvável na aplicação implementada? Então, como dito anteriormente, o bom censo deve ser colocado em prática aqui. Ao final, tudo se resume a "custo/benefício".

Ainda sobre as opções do esquema de Memória, algumas tem "partições" que ao serem somadas resultam em menos que os ***4MBytes***, como por exemplo o esquema ***"Minimal SPIFFS..."***. E no caso deste esquema, ao se fazer as contas conclui-se que o "faltante" de Memória é cerca de **1.9MB**. Esta memória "faltante" está sendo reservada para o ***OTA***.

Este tamanho reservado ao OTA, tem uma explicação: a forma como o OTA funciona. Essencialmente, o OTA *exige* que se reserve o mesmo tamanho reservado para o código do usuário. Então se alguém quer ter 1.3MB de código, e quer ter também o OTA, isto ocupará 2.6MB da Memória de Programa, e numa placa onde existem 4MB no total, sobrará no máximo 1.4MB para outros recursos do ESP32.

No caso do esquema ***"Minimal SPIFFS..."*** onde o código pode chegar a 1.9MB, e ainda se tem o OTA, pode-se constatar facilmente a distribuição da

Memória de Programa: como o código do usuário (código do Arduino) pode chegar a 1.9MB, para o OTA também serão necessários outros 1.9MB. Isto implica que 1.9+1.9 = 3.8MB serão necessários para código e OTA. Por este motivo sobra menos de 0.2MB para o SPIFFS (na opção descrito como 190kB).

Para um código que utiliza o ***PCOM WiFi/Bluetooth***, recomenda-se usar um desses dois esquemas de Memória para a placa do ESP32:

1) esquema ***Partition Scheme"Huge APP..."*** que não inclui o OTA mas permite um ***Sistema de Arquivos SPIFFS*** de ***1MB***. Este esquema permite um tamanho de código de até **3MB**, o que dá a liberdade para se fazer implementações que necessitem muita lógica e ramificações complexas de código. Um exemplo é um Sistema com exibição de grande quantidade de Menus em um Display TFT Gráfico, incluindo-se diversas tabelas de caracteres (as "fontes", que costumam ocupar muita memória de Programa). E possibilita também o uso simultâneo de uma enormidade de bibliotecas para os mais diversos dispositivos de Hardware existentes atualmente no mercado (*Sensores*, *Motores*, *etc*), além de ser possível incluir-se uma grande quantidade de páginas HTML para a Inteface com um Cliente WiFi. Isso tudo com a capacidade de se armazenar dados em arquivos de tamanho pequeno ou mediano.

2) esquema ***"Minimal SPIFFS..."*** que inclui o OTA mas com um mínimo de SPIFFS. Isto é ideal para implementações onde não há grande complexidade no Sistema, mas existe grande necessidade de atualização do código através do OTA por motivos diversos (por exemplo, devido à placa estar localizada em um local de difícil acesso). Escrevendo-se um código de forma racional e utilizando-se de técnicas de programação adequadas (a Linguagem C/C++ permite isso), é possível se ter um Sistema relativamente sofisticado. Neste esquema, o espaço para o Sistema de Arquivos SPIFS é mínimo, mas pode ser muito bem aproveitado, ao se usar por exemplo pequenos arquivos de Log e/ou arquivos de dados binários.

Nestes dois esquemas de Memória, o percentual de código que "sobra" para alterações posteriores será entre ***30%*** e ***60%***, o que dá uma margem bastante confortável para revisões futuras no código.

E lembrando que é possível se usar também no Sistema, um ***Cartão de Memória*** tipo ***SD Card***. Atualmente existem a baixo custo, Cartões com grandes capacidades, permitindo-se ter um Sistema de Arquivos convencional na Plataforma Arduino (normalmente ***FAT16*** ou ***FAT32***), ampliando-se muito as possibilidades do que se pode fazer usando o ESP32 com o PCOM WiFi/Bluetooth na Plataforma Arduino.

AFA

(Arduino for all)