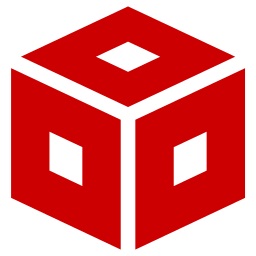
**Code Composer Studio e TI-RTOS**





**Elaborado por:**

Valentim Rodrigues

**Verificado por:**

**Orientado por:**

Professor Luís Bento

Professor Carlos Neves

Índice

[1 Code Composer Studio 5](#_Toc74600304)

[1.1 Instalação do Code Composer Studio 5](#_Toc74600305)

[1.2 Introdução ao Code Composer Studio 5](#_Toc74600306)

[1.3 Carregar um programa para uma placa 5](#_Toc74600307)

[2 TI-RTOS em Ambiente Windows 5](#_Toc74600308)

[2.1 Instalação do SDK TI-RTOS 5](#_Toc74600309)

[2.2 Real-Time Software Components (RTSC) 5](#_Toc74600310)

[2.3 Relação entre SYS/BIOS e o RTSC 5](#_Toc74600311)

[2.4 Criação de projetos exemplos 5](#_Toc74600312)

[2.5 Configuração de Drives e Periféricos 5](#_Toc74600313)

[2.6 Iniciação de um projeto exemplo: 5](#_Toc74600314)

[2.7 Configuração dos pinos como I/O 5](#_Toc74600315)

[2.8 Configuração do SYS/BIOS de forma visual 5](#_Toc74600316)

[2.9 Ferramentas de Depuração do TI-RTOS 5](#_Toc74600317)

**Índice de Figuras**

[Figura 1 - Vista geral do CCStudio 6](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690315)

[Figura 2 - Arquivo .ccxml a configurar 7](#_Toc74690316)

[Figura 3 - Teste Conexão entre Placa e CCStudio 7](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690317)

[Figura 4 - Configurações Avançadas CCStudio 8](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690318)

[Figura 5 - ESCREVER AINDA 8](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690319)

[Figura 6 - Barra de Tarefas Superior CCStudio 9](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690320)

[Figura 7 - Pasta C://ti 11](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690321)

[Figura 8 - Importação dos componentes e módulos da UART 13](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690322)

[Figura 9 - Pasta Padrão pdk\_am335x 15](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690323)

[Figura 10 - Variáveis a modificar no script CreateProject 15](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690324)

[Figura 11 - ESCREVER 16](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690325)

[Figura 12 - Exemplos de comandos para a criação de projetos 17](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690326)

[Figura 13 - PINMUXTOOLS Layout 18](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690327)

[Figura 14 - PINMUX configuração de periféricos 19](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690328)

[Figura 15 - ESCREVER 19](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690329)

[Figura 16 - Pinmux Uart 2 e Uart Pru 1 20](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690330)

[Figura 17 - Pinmux DCAN0 20](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690331)

[Figura 18 - Importação de um projeto para o Workspace do CCS 21](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690332)

[Figura 19 - Arquivos do projeto importado 22](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690333)

[Figura 20 - ESCREVER 22](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690334)

[Figura 21 - Arquivo .cfg do projeto criado 23](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690335)

[Figura 22 - ESCREVER 23](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690336)

[Figura 23 - Código a inserir na função taskFnx 24](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690337)

[Figura 24 - Console Terminal projeto exemplo 24](file:////Users/catarina/Desktop/OneDrive-2021-06-14/Capa_Exemplo_marco.docx#_Toc74690338)

**PARTE I – Tutorial Code Composer Studio**

# Code Composer Studio

O Code Composer Studio é descrito como um ambiente de desenvolvimento integrado (IDE), baseado no *framework* Eclipse, que oferece um suporte ao portfólio de microcontroladores e processadores incorporados da TI. Desta forma, o CCS compreende um conjunto de ferramentas usadas para o desenvolvimento e *debug* de aplicativos incorporados, como um compilador C/C++ (GCC) otimizado, um editor de código-fonte, um ambiente de construção de projetos, *debugs*, criadores de perfil e muitos outros recursos.

Face a estas características, o CCS mostra-se uma grande ferramenta de desenvolvimento para o que este tutorial pretende ensinar e explicitar aos leitores.

## Instalação do Code Composer Studio

A instalação do CCS em ambiente Windows é relativamente simples, uma vez que é apenas necessário realizar o respetivo download do CCS na página oficial da Texas, executar o instalador, escolher a família de processadores que se deseja instalar que, neste caso de estudo, é a família SITARA e, por fim basta selecionar o local de instalação desejado.

A instalação em ambiente Linux requere a instalação de bibliotecas adicionais, contudo no site da Texas disponibiliza um tutorial de como proceder à respetiva instalação e quais os módulos a instalar, uma vez que estes podem variar consoante a versão que o utilizador esteja a utilizar.

É recomendado instalar o CCS na raiz do sistema operativo, no caso do Windows em “C:/” ou em Linux em “/home/nome\_usuario/” pois, como será observado ao longo deste tutorial, existem diversos *scripts* que irão ser necessários utilizar e que não foram construídos de uma forma totalmente dinâmica. Desta forma, se o CCS não for instalado na raiz inicialmente, será necessário realizar-se mudanças profundas nestes mesmos *scripts*.

## Introdução ao Code Composer Studio

A IDE do CCStudio, como mencionado inicialmente, é baseada em eclipse, pelo que o conceito desta se torna muito idêntico. Ao iniciar-se a aplicação aparece, normalmente, uma mensagem para criar um *workspace*, que pode ser descrito como uma pasta, onde são guardados os projetos e configurações da própria IDE. Na figura abaixo é apresentado um exemplo de uma interface padrão baseada em eclipse, contudo esta pode ser personalizada.

Uma imagem com texto

Descrição gerada automaticamente

Figura 1 - Vista geral do CCStudio

## Carregar um programa para uma placa

Como próximo passo, pretende-se realizar um carregamento de um *debug* de um programa para a placa. Em primeiro lugar, é necessário criar um documento com a extensão “.ccxml”, como por exemplo, “TargetConfiguration.ccxml”, que vai conter a informação da placa e alguns ficheiros que serão necessários para efetuar o carregamento.

1. Na barra de ferramentas clique na opção **View**, selecione “**TargetConfiguration**”, ou o nome do documento criado anteriormente e **clique OK**.
2. Na barra lateral esquerda auxiliar irá aparecer uma pasta com o nome “**User Defined**”, clique com o **botão do lado direito do mouse** e escolha a opção “**New target configuration”.** Irá surgir logo abaixo um ficheiro chamado “**NewTargetConfiguration.ccxml**”
3. Uma imagem com texto

   Descrição gerada automaticamenteÉ necessário **abrir o arquivo** para proceder às alterações necessárias.

Figura 2 - Arquivo .ccxml a configurar

1. Uma imagem com texto

   Descrição gerada automaticamenteEscolha o tipo de “**connection**” que a placa possui bem como a placa a utilizar e de seguida teste a comunicação entre ambas. Para este teste, clique no botão “Test Connection”, de seguida irá aparecer uma caixa com informações e, no final é possível visualizar se a conexão ocorreu com sucesso.

Figura 3 - Teste Conexão entre Placa e CCStudio

1. Após o teste de conexão, para o este passo, clique na opção “Advanced”. É de notar que, para o exemplo apresentado não será necessário proceder a nenhuma alteração no separador de configurações avançadas, contudo é de extrema importância ressalvar que existem configurações avançadas que podem ser alteradas consoante a necessidade do utilizador.

Nas configurações avançadas é mostrado as cores do processador bem como outros módulos, comos os PRU´s. Existe um arquivo (em formato.gel) que é, basicamente, um *script* que contêm algumas informações do processador, como por exemplo, a memória, que ao fazer o carregamento de um arquivo binário, é necessário saber a quantidade e quais as regiões de memoria que poderão ser escritas (entre outras funções que os arquivos gel possuem).

Uma imagem com texto

Descrição gerada automaticamente

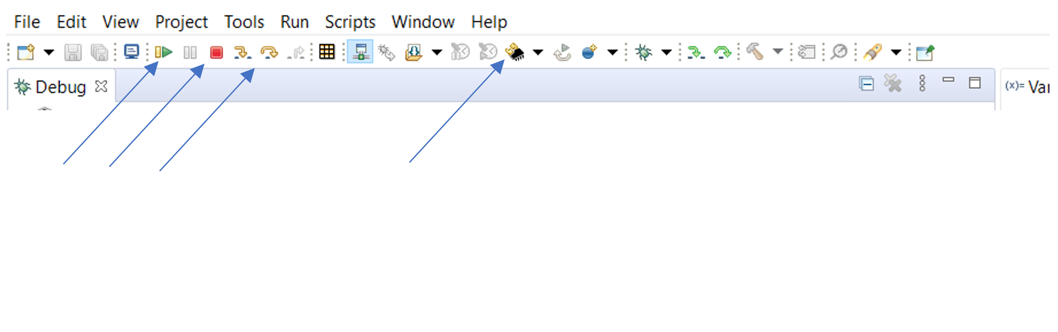
Figura 4 - Configurações Avançadas CCStudio

1. Para fazer o carregamento de um programa para a placa, basta clicar com o botão do lado direito do rato no arquivo “.ccxml” criado anteriormente e selecionar a opção “**Launch Selected Configuration**”.
2. Uma imagem com texto

   Descrição gerada automaticamenteDe seguida selecione a segunda opção (CortexA8) e clique ***crl+alt+c*** para iniciar o arquivo.gel referido anteriormente. Na consola do CCStudio irá aparecer mensagens de *debug* que o próprio arquivo.gel possui, sendo possível visualizar o processo de inicialização da placa.

Figura 5 - ESCREVER AINDA

1. Por fim, faça ***crl+alt+L*** e escolha o arquivo binário do projeto a carregar [[1]](#footnote-1). Clique no *play*, situado na barra de ferramentas superior e, junto a este símbolo existe também outros atalhos interessantes, que podem ser posteriormente explorados, que permitem executar um *debug*, como o “*pause*”, “*step into*”, entre outros.



# 

Figura 6 - Barra de Tarefas Superior CCStudio

**PARTE II – Tutorial de TI-RTOS em Ambiente Windows**

# TI-RTOS em Ambiente Windows

TI-RTOS é um sistema operacional em tempo real para microprocessadores que possui um *kernel* multitarefa em tempo real (multithreading preemptivo determinístico), serviços de sincronização, controlo de memória e tratamento de interrupções. O TI-RTOS possui diversos módulos, como por exemplo o UART, o I2C, o NDK (networking) que fornece uma pilha TCP/IP compatível com IPv4, o IPv6, o DNS, o DHCP que fornece igualmente um sistema de arquivos Fatfs ou USB, bem como serviços típicos de um *kernel* (Mutex, HeapBuff, Interrupções de Hardware e Software, Tasks, etc…). Para os processadores SIRATA, o TI-RTOS está incluído no pacote TI-RTOS SDK.

## Instalação do SDK TI-RTOS

Para os processadores da família SITARA, o *kernel* TI-RTOS está contido dentro do SDK (*Software Development Kit*) que é uma plataforma de software especifica para processadores integrados. Antes de efetuar o download deste, recomenda-se que seja efetuado, em primeiro lugar, a instalação da IDE CCS e só depois instalar o SDK, isto porque, ao instalar a IDE na raiz do computador irá ser criada uma pasta por padrão “C:\ti” que contem a instalação da IDE e, será dentro desta pasta que se deve instalar o SDK-TI-RTOS. Desta forma, e após a instalação da IDE CCS, para realizar o download do SDK-TI-RTOS, basta aceder ao link: “<https://www.ti.com/tool/PROCESSOR-SDK-AM335X>”.

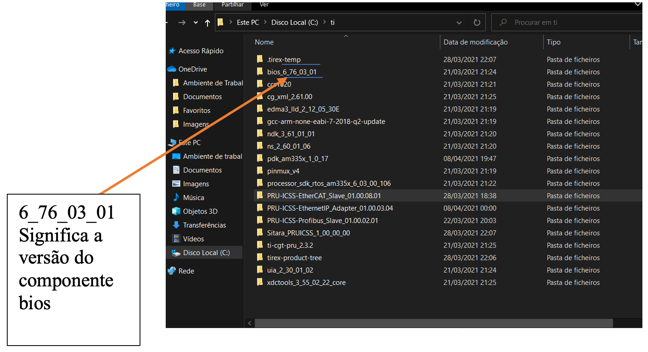
Quando a instalação do SDK terminar e com a IDE já instalada, o utilizador irá ter uma pasta na raiz do disco, assim como diversas páginas que são criadas automaticamente, como é apresentado na figura seguinte.

Figura 7 - Pasta C://ti

Em suma, o CCStudio está contido na pasta “ccs1020” que significa que a versão deste é a 10.20, o *kernel* TI-RTOS está presente na pasta “bios\_6\_76\_03\_01”, o módulo Network está na pasta “ndk\_3\_61\_01\_01” e os restantes módulos estão nas pastas “pdk\_am335x\_1\_0\_17”.

Ao terminar a instalação do SDK, o próximo passo é criar projetos exemplos relacionados com os módulos que estão acessíveis à placa que o utilizar esteja a utilizar.

## Real-Time Software Components (RTSC)

O projeto [RTSC](http://www.eclipse.org/rtsc/) fornece ferramentas [básicas](http://www.eclipse.org/rtsc/) e conteúdo de tempo de execução de baixo nível que permite o desenvolvimento de componentes em linguagem C. Este projeto vem com a adição do XDCtools, que contém todas as ferramentas necessárias para criar, testar, implantar, instalar e usar componentes do projeto RTSC.

Este produto é disponibilizado e integrado junto ao *kernel* TI-RTOS, incluindo ferramentas e padrões para o desenvolvimento de API, assim como, para compactar bibliotecas. Os componentes RTSC têm interfaces neutras em termos de hardware, podem ser configuradas em modo offline de forma a otimizar a memória e o desempenho e, é personalizável no ambiente de desenvolvimento por meio de uma linguagem de *script* (na IDE CCStudio o *script* de configuração é o “.cfg” como será apresentado a seguir).

O principal benefício da utilização do RTSC é a possibilidade de este padronizar a entrega do conteúdo de destino, tornando mais fácil a inclusão do conteúdo nos aplicativos. Desta forma, o RTSC fornece uma ponte entre o conteúdo de destino do componente, escrito em C/C++ e, uma variedade de ferramentas de desenvolvimento que geram o componente ao longo do seu ciclo de vida, isto é, desde a criação, integração e teste, até à monitorização em tempo real do conteúdo de destino no equipamento final implantado.

Os utilizadores do RTSC podem ser divididos em dois tipos desenvolvedores: os consumidores, que integram os pacotes de conteúdo de destino como, algoritmos DSP, drivers de dispositivo, pilhas TCP / IP, e sistemas operacionais em tempo real e nos quais os utilizadores como nós se enquadram; e os produtores, que criam os pacotes usados pelos consumidores.

A imagem seguinte fornece uma ideia base de como os pacotes são “utilizados pelos consumidores”, ou seja, por nós. Por exemplo, para configurar a UART no projeto a ser desenvolvido é necessário incluir os arquivos “.h” nas propriedades do projeto, como explicitado na introdução ao CCStudio, mas também é necessário importar os componentes e módulos da UART que estão na pasta “c:/ti/pdk/packages/ti/drv/uart”.

Esta importação, que está representada na figura seguinte, explicita que, dentro da pasta do periférico a importar (UART, por exemplo) existem módulos e componentes, desenvolvidos pelos desenvolvedores RTSC e nós clientes RTSC, que vamos utilizá-los através do *script* “.cfg” existente no projeto.

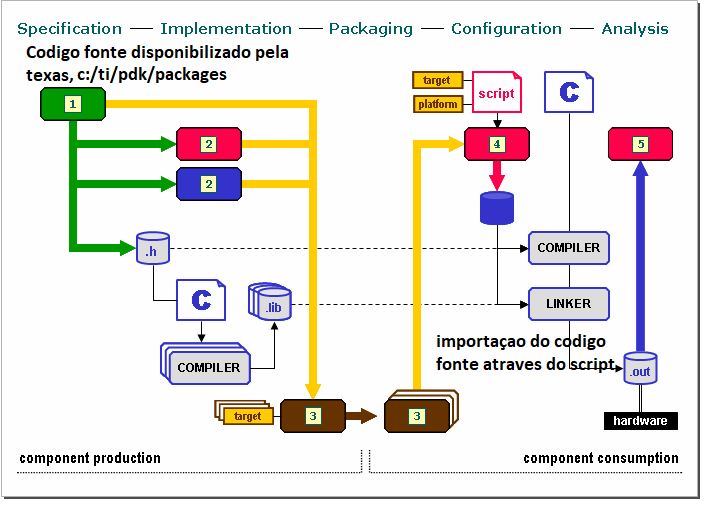


Figura 8 - Importação dos componentes e módulos da UART

É recomendável a leitura destes dois tutoriais para entender como o script funciona com a integração no CCStudio: “<http://rtsc.eclipseprojects.io/docs-tip/XGCONF_User%27s_Guide>” e “<http://rtsc.eclipseprojects.io/docs-tip/RTSC%2BCCStudio_v4_QuickStart>”.

## Relação entre SYS/BIOS e o RTSC

Como referido no ponto anterior, os pacotes SYS / BIOS são “importados” da mesma maneira que os pacotes RTSC, pelo que, por padrão, a convenção usada consiste em um padrão de nomenclatura hierárquico, onde cada nível é separado por um ponto ("."). Normalmente, o nível mais alto do nome é o fornecedor ("ti"), seguido pelo produto ("sysbios") e, por último, pelos nomes dos módulos e submodelos (por exemplo, "knl").

A configuração de um aplicativo é armazenada num um ou mais arquivos de *script* com uma extensão de arquivo “.cfg” e são analisados pelo XDCtools, indo gerar o código-fonte C correspondente, cabeçalho C e arquivos de comando do linker que são compilados e vinculados ao aplicativo final.

## Criação de projetos exemplos

Para dar início à criação de projetos exemplo é necessário primeiro perceber como é que a pasta “c:/ti/pdk/packages” (*platform development kit*), que é apresentada na imagem seguinte, está organizada.

Deste modo, dentro desta existe uma pasta com o nome “ti” que contém o software de baixo nível dos periféricos disponíveis que necessita de ser compilado antes de iniciar os projetos. Existem duas formas de compilar estes projetos: Na pasta “C:\ti\pdk\_am335x\_1\_0\_17\packages” executa-se o comando no terminal “gmake all” onde este vai proceder à compilação do código fonte de todos os periféricos existentes na pasta “ti”; ou, outra forma é, compilar periférico a periférico, isto é, dentro da pasta “ti” é necessário entrar na pasta “*drv*”, depois na pasta “uart” e é dentro desta página que se pode executar o comando “gmake all” para compilar o código fonte do periférico em específico.

Duas notas importantes sobre a compilação anteriormente explicada.

1. Ao executar a primeira compilação, depois da primeiro instalação do SDK-RTOS, é recomendável proceder á compilação geral de todos os módulos, e para isso basta ir a pasta “C:\ti\pdk\_am335x\_1\_0\_17\packages” e executar o comando “gmake all”, como anteriormente referido.
2. É necessário fazer sempre um *rebuild* de um determinado periférico quando se procede a uma alteração do código fonte. Este *rebuild* pode ser realizado localmente, na pasta do periférico, executando os seguintes comandos “gmake clean” e de seguida “gmake all”. Se, por exemplo, for necessário modificar o código da Uart, torna-se necessário ir à pasta “C:\ti\pdk\_am335x\_1\_0\_17\packages\ti\drv\uart” e executar os referidos comandos.

Uma imagem com texto, captura de ecrã, preto

Descrição gerada automaticamentePor padrão os projetos exemplos não vêm criados, sendo desta forma necessário criá-los com a ajuda de um *script*, tendo-se de direcionar para a pasta “c:/ti/pdk\_am335x\_ xx\_xx/packages”.

Figura 9 - Pasta Padrão pdk\_am335x

Após esta explicação inicial acerca da pasta “c:/ti/pdk/packages”, dá-se início à criação de projetos exemplo. Deste modo e, em primeiro lugar, é necessário editar o *script* “pdkProjectCreate.bat”, que pode abri-lo com um editor de texto à escolha. Como é possível observar na imagem seguinte, no cabeçalho do script está alocada toda a informação e os comandos importantes para usar o *script*, sendo necessário, para configurar corretamente o script, modificar as variáveis a partir da linha 436 até a linha 496.

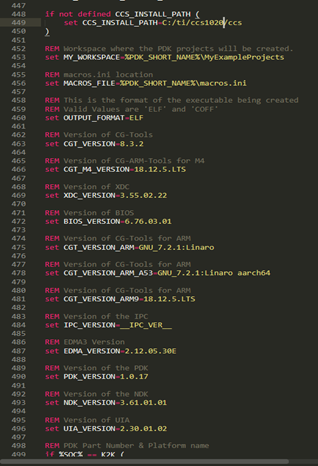


Figura 10 - Variáveis a modificar no script CreateProject

Quanto à variável CCS\_INSTALL\_PATH, por exemplo, é necessário modificar a versão do CCStudio instalado, versão esta que é facilmente visível assim como a do Kernel, PDK, etc.

Com as variáveis do script corretamente definidas, é necessário seguir os seguintes passos para criar projetos exemplos:

1. Uma imagem com texto

   Descrição gerada automaticamenteAbrir um terminal Windows na pasta “C:\ti\processor\_sdk\_rtos\_am335x\_6\_03\_00\_106” e executar o seguinte comando “setupenv.bat” para configurar o “SDK build Enviroment configure”. O objetivo deste comando é configurar as variáveis de ambiente que sdk\_rtos necessárias para criar os projetos no CCStudio.

Figura 11 - ESCREVER

1. ***Sem fechar o terminal*** mudar para a pasta onde o pdk\_am335x está instalado. Pode executar o seguinte comando “cd C:\ti\pdk\_am335x\_1\_0\_17\packages” para mover para a respetiva pasta.
2. Para configurar as variáveis de ambiente temporárias é necessário executar o comando no terminal “pdksetupenv.bat”.
3. Agora é necessário fazer o build de todos os periféricos como anteriormente explicado., para tal execute o comando “gmake all”.

Nesta fase, este processo irá demorar algum tempo a ser realizado. Há a opção de executar o comando “gmake all -j4” para ativar o paralelismo entre cores da cpu do computador, contudo não é recomendado executá-lo para uma primeira vez.

Se nenhuma mensagem de erro apareceu no terminal, está tudo configurado para se puder criar os projetos. Posto isto, execute o comando “pdkProjectCreate.bat”, indo este comando criar os projetos para todas as placas da família SITARA. É possível configurar e criar os projetos para a placa em questão, uma vez que o *script* “pdkCreateProject.bat” tem parâmetros para configurar a placa desejada. O próprio script apresenta alguns exemplos bastante esclarecedores neste aspecto.

Uma imagem com texto

Descrição gerada automaticamente

Figura 12 - Exemplos de comandos para a criação de projetos

Os exemplos criados através do script são criados e colocados por padrão na pasta “C:\ti\pdk\_am335x\_1\_0\_17\packages\MyExampleProjects”. Se tudo correu como esperado, na pasta “MyExamplesProjects” existem diversos exemplos que servirão como ponto de partida.

## Configuração de Drives e Periféricos

Quando se procede à instalação do “*SDK-RTOS”* nem todos os periféricos ou pinos estão configurados, como tal, é necessário utilizar o “PINMUX TOOL” que é uma ferramenta desenvolvida pela Texas que nos ajuda visualmente a configurar os módulos e os respetivos pinos, tornando mais fácil a sua configuração. A ferramenta está disponível no site oficial da texas, sendo esta de fácil instalação.

Para configurar um módulo é então necessário:

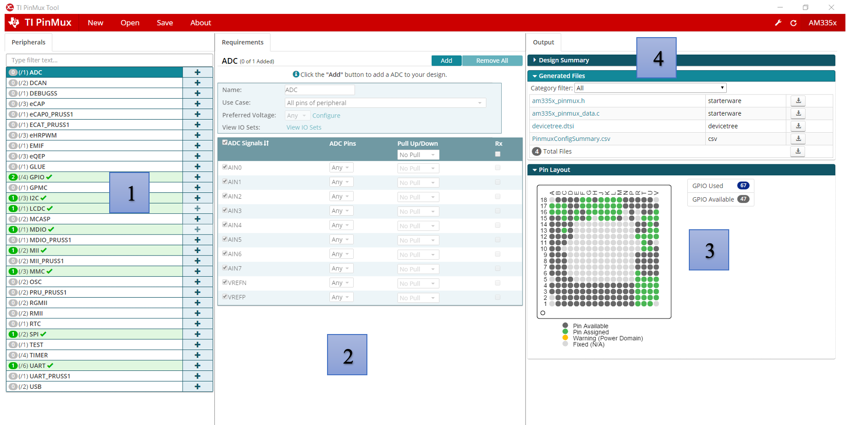
1. Abrir a ferramenta “PINMUX TOOL” e no separador “*Open existing design*” basta clicar em “*Open*” e navegar até a pasta “pdk\_am335x\_1\_0\_17\packages\ti\starterware\tools\pinmux\_config\am335” que é onde se encontram os arquivos referentes às placas suportadas pelo SDK instalado.
2. Abrir o arquivo correspondente à placa utilizada (nesta demostração é a beaglebone\_blackconfig.pinmux). O aspeto visual será muito semelhante ao da seguinte figura.

Figura 13 - PINMUXTOOLS Layout

Na figura 8, o número 1 corresponde à aba com os periféricos disponíveis, em que os que já se encontram utilizados e corretamente configurados apresentam a cor verde contudo, caso não se visualize esta cor mas sim a cor vermelha, significa esta que existe algum erro nos periféricos; o número 2 corresponde à aba onde é possível procede à configuração dos pinos disponíveis para um determinado módulo (no caso do processador Am335x, existem diversos multiplexadores de pinos, pois o pino X pode funcionar como Uart ou I2c); o número 3 corresponde à aba que apresenta o *layout* dos pinos do processador (a cor verde significa que já estão em atualização). Os pinos por padrão são indexados por colunas e linhas e ao colocar o rato sobre o círculo verde correspondente ao pino em questão, é aberta uma caixa com informações úteis sobres os pinos, por exemplo, o pino E16 está configurado com o nome “uart0\_txd” e está na funcionalidade 0 (na configuração dos GPIO’s será explicada a numeração dada às funcionalidades dos pinos); por fim, o número 4 corresponde ao separador onde são apresentados os ficheiros gerados pelo próprio software. Os ficheiros relevantes para trabalhar com TI-RTOS são os ficheiros *startware* e o ficheiro “devicetree.dtsi” que, neste último caso, torna-se útil para quando se executa Linux na *beaglebone*, uma vez que é descrito como um arquivo que pertence ao kernel do Linux e que tem de ser compilado para se tornar com um arquivo legível pelo kernel do Linux.

1. Ativar o módulo DCAN0, UART\_0 e 2, UART\_PRUSS1, I2C, GPIO, MMI, MMC, SPI. Nesta fase é necessário fazer o *download* do ficheiro “am335x\_pinmux\_data.c” (*startware*), gerado pela ferramenta PINMUXTOOLS, para uma pasta à sua escolha, tendo em atenção que este ficheiro será necessário utilizar numa fase seguinte. Na figura seguinte é exibido a ativação dos módulos acima mencionados.

Uma imagem com mesa

Descrição gerada automaticamente

Figura 14 - PINMUX configuração de periféricos

1. Substituir o arquivo “am335x\_beagleboneblack\_pinmux\_data.c” existente na pasta “C:\ti\pdk\_am335x\_1\_0\_17\packages\ti\starterware\board\am335x” pelo arquivo gerado anteriormente, sendo necessário fazer certas alterações.

Uma imagem com texto, placa

Descrição gerada automaticamentePrimeiro, abrir o arquivo “am335x\_pinmux\_data.c” e renomear o *array* do tipo *struct*, que se encontra no final do ficheiro. Neste exemplo procedeu-se à alteração do nome “gAM335xPinmuxData” para “gBbbPinmuxData”, como é apresentado na figura seguinte.

Figura 15 – Array de estruturas de definições

De seguida é necessário renomear o arquivo “am335x\_pinmux\_data.c” alterado para “am335x\_beagleboneblack\_pinmux\_data.c” e substituir na pasta “C:\ti\pdk\_am335x\_1\_0\_17\packages\ti\starterware\board\am335x” pelo ficheiro já existente. Nesta etapa é recomendado guardar o ficheiro “am335x\_beagleboneblack\_pinmux\_data.c” antigo, por uma questão de segurança.

Verifique se o arquivo “am335x\_pinmux.h” existente na pasta “C:\ti\pdk\_am335x\_1\_0\_17\packages\ti\starterware\board\am335x”, e tem de ser declarado com um array externo no arquivo “am335x\_beagleboneblack\_pinmux\_data.c”, ou seja, tem de existir no arquivo “am335x\_pinmux.h” uma variável chamada “extern pinmuxBoardCfg\_t gBbbPinmuxData”.

1. Uma imagem com texto

   Descrição gerada automaticamenteAdicionar os *pinmux* aos periféricos para que possam ser iniciados e configurados na placa com um código exemplo. Na pasta “C:\ti\pdk\_am335x\_1\_0\_17\packages\ti\board\src\bbbAM335x”, abra o arquivo “bbbAM335x\_pinmux.c” e inicie os módulos como mostram as figuras seguintes.

Figura 16 - Pinmux Uart 2 e Uart Pru 1

Uma imagem com texto

Descrição gerada automaticamente

Figura 17 - Pinmux DCAN0

1. **PASSO OPCIONAL** – Existem duas formas de iniciar os módulos. Na pasta mencionada no passo anterior existe um arquivo designado por “bbbAM335x.c” que apresenta uma função responsável por iniciar o *clock* nos respetivos periféricos, “Board\_STATUS e Board\_moduleClockInit ()”. Desta forma, no exemplo que é exposto, procedeu-se à configuração da UART\_2 e consequente configuração do pinmux, sendo necessário nesta fase iniciar o *clock*, contudo esta iniciação pode ser encetada nesta função ou então no próprio projeto a desenvolver. Recomenda-se configurar o pinmux neste arquivo se se tratar de um periférico usado em todos os projetos, caso contrário é recomendado então configurar no próprio projeto como irá ser demostrado mais adiante.
2. Por último é necessário fazer novamente um *rebuild* total. Na pasta “C:\ti\pdk\_am335x\_1\_0\_17\packages” execute os seguintes comandos “gmake clean -j4” e de seguida “gmake all -j4”. O “-j4” torna-se facultativo uma vez que ativa o paralelismo do processador do computador *host* tornando a operação mais rápida. Neste passo efetuou-se um “*rebuild all*” uma vez que se modificou diversos periféricos, contudo, caso se modifique apenas a UART, basta realizar apenas um *rebuild* nessa pasta.

## Iniciação de um projeto exemplo:

O objetivo neste ponto passa por demostrar como se pode importar, compilar e executar um projeto exemplo como o criado anteriormente, sendo necessário que o utilizador tenha presente o que foi explicado na Parte I, acerca do CCStudio.

De modo a complementar esta fase, será apresentado um exemplo onde se irá importar um projeto exemplo para trabalhar com a UART 1 relacionada ao PRU 0. A importação de um projeto exemplo, ainda que se trate de um exemplo e seja algo diferente do projeto pretendido, torna-se mais fácil o seu desenvolvimento uma vez que a configuração do RTCS já vem pré-definida por padrão. Os próximos passos descrevem o processo de importação, configuração e compilação de um projeto.

1. Uma imagem com texto

   Descrição gerada automaticamenteCom o CCStudio aberto, num workspace à escolha, clique em “Project-> Import CCS Projects”, indo este passo abrir uma janela como mostrado na figura seguinte. Selecione o diretório “C:\ti\pdk\_am335x\_1\_0\_17\packages\MyExampleProjects” criado anteriormente e selecione o projecto com o nome “UART\_FwTestExtLb\_bbbAm335x\_armTestProject”, como apresentado na figuraa 12 e clique em Finish. O projeto deverá aparecer na Aba “Project explorer”, localizada à esquerda do ecrã e com uma configuração idêntica ao da figura13.

Figura 18 - Importação de um projeto para o Workspace do CCS

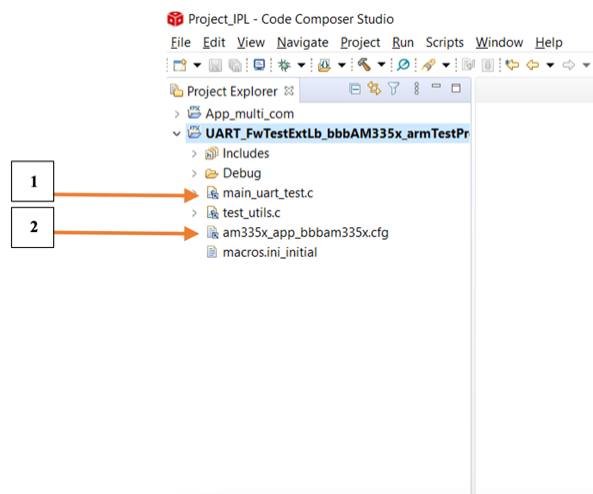


Figura 19 - Arquivos do projeto importado

1. Nesta etapa, é necessário que o utilizador constate os arquivos que o projeto importado apresenta. Observando a figura 13, esta apresenta uma pasta com os “*includes*” correspondentes aos diretórios presentes nas configurações do projeto (como explicado no tópico de introdução do CCStudio), assim como, outros dois arquivos importantes identificados como os números 1 e 2, respetivamente. No arquivo 1 é possível encontrar-se a função *main* assim como a criação das *tasks* que o projeto incorpora, enquanto no arquivo 2, sendo este um arquivo com extensão .cfg, contêm as configurações do RTSC (debatido no tópico 2.2 deste tutorial).
2. Uma imagem com texto

   Descrição gerada automaticamentePor padrão os arquivos que o projeto possui não estão fisicamente no *workspace* do utilizador, mas sim num *link* correspondente os arquivos originais pelo que, quando se efetuam alterações num dos projetos que possuam o arquivo “main\_uart\_test.c”, ambos os projetos serão alterados, podendo não ser o desejado pelo utilizador. Desta forma é recomendável apagar os arquivos que estão “linkados” efetuando uma cópia dos mesmos para o projeto tendo de abrir as propriedades do projeto em “Linked Resources”. A forma mais fácil de realizar este passo é, apontar a localização dos projetos, de seguida eliminá-los, ir á pasta onde estes se encontram e voltar a copiá-los para o projeto.

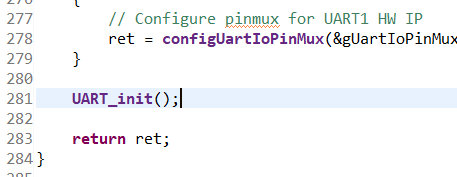
Figura 20 – Linked Resources do projeto

1. Uma imagem com texto

   Descrição gerada automaticamenteNeste ponto abra o arquivo “.cgf” em modo script e compare com o que foi explicado na secção 2.2 deste tutorial. Desta forma irá ter uma visão de baixo nível onde os módulos e os pacotes do drive sys/bios e RTSC são importados para o projeto.

Figura 21 - Arquivo .cfg do projeto criado

1. De seguida torna-se necessário fazer alterações ao código para colocar a UART 1 integrada com o PRU0 a funcionar. Assim, abra o arquivo “main\_uart\_test.c”, aceda à função “*main*” que se encontra em último (pode ignorar o resto do código por agora), e verifique que a função “Board\_initUART” está a ser chamada, isto porque, esta adquire a função de iniciar os periféricos da placa de desenvolvimento.

Posteriormente, pressione as teclas “control + left mouse” ao mesmo tempo e será redirecionado para a implementação da função. Aqui, sugere-se que o utilizador coloque antes do *return* da função a designação “UART\_init ();” de modo a iniciar a UART0, tendo este passo como objetivo fazer o *debug*, uma vez que a UART0 é a UART compartilhada com chip FTDI que contêm um conversor serial para USB juntamente com o protocolo JTAG. Note que, não seria necessário chamar a função referida se a linha 247 “//boardCfg |= BOARD\_INIT\_UART\_STDIO;” se apresentasse descomentada, mas, como a linha referida necessita de estar comentada no exemplo de estudo, primeiro é necessário configurar a UART1 e o PRU e só depois iniciar a UART0 em modo “UART\_STDIO” (existem 2 API’s para trabalhar com UART na família SITARA, consulte os documentos do periférico para obter maior detalhe).

Note também que, nas funções “status=PRCMModuleEnable(CHIPDB\_MOD\_ID\_PRU\_ICSS,1U,0U);” e “status = PRCMModuleEnable (CHIPDB\_MOD\_ID\_UART, 1U, 0U);”, o objetivo destas funções é iniciar os periféricos, assim, se o tivéssemos feito anteriormente, no tópico 2.5 não seria necessário incluir estas funções no projeto, pois o módulo já estaria inicializado ao compilar o projeto.

1. Direcionando novamente o foco para a função “*main*” é possível analisar que existe uma *task* a ser criada, “task = Task\_create (taskFxn, &taskParams, &eb);” onde a função “taskFnx” é executada pela *task* a ser formada. Assim, dirija-se à implementação da função e apague o conteúdo da mesma e, copie o código da função “taskFxn”, como mostrado na figura seguinte.

Uma imagem com texto

Descrição gerada automaticamente

Figura 23 - Código a inserir na função taskFnx

1. Uma imagem com texto, eletrónica, captura de ecrã

   Descrição gerada automaticamenteDe modo a obter a visualização seguinte, é necessário ligar um conversor usb\_serial ao pino D16 (rx) e D17 (tx)e proceder ao carregamento do programa na placa.

Figura 24 - Console Terminal projeto exemplo

## Configuração do SYS/BIOS de forma visual

## Ferramentas de Depuração do TI-RTOS

Em conjunto com o Code Composer Studio (CCS), o TI-RTOS Kernel fornece várias ferramentas para auxiliar a depuração e otimização de aplicativos *multithread*. Nessas ferramentas inclui-se o Run-time Object Viewer (ROV) sendo uma ferramenta que permite verificar o estado do sistema operacional incluindo: as tarefas do próprio sistema; o *mailbox*; os semáforos; o ponto de situação de uma tarefa, se está pronta, em execução ou bloqueada; as tarefas que se encontram bloqueadas num IPC; ou se as tarefas excederam os seus limites de pilha. Assim como o CCS que inclui, de igual modo, uma ferramenta chamada RTOS Analyzer que captura o comportamento em tempo real de uma tarefa podendo exibir a execução do *thread,* a carga da CPU, eventos do sistema operacional e informações de log definidas pelo usuário.

1. O arquivo binário do projeto só fica criado ao fazer o build do projeto. [↑](#footnote-ref-1)