



Universidade Federal de Uberlândia
Faculdade de Engenharia Elétrica

PEDRO HENRIQUE COELHO ALVES

**ESTUDO DE TEMPO DE RESPOSTA DO GPIO DA BEAGLEBONE
BLACK**

Uberlândia
2017

PEDRO HENRIQUE COELHO ALVES

**ESTUDO DO TEMPO DE RESPOSTA DO GPIO DA BEAGLEBONE
BLACK**

COM E SEM SISTEMA OPERACIONAL (LINUX EMBARCADO)

Trabalho apresentado como requisito parcial de avaliação na disciplina Trabalho de Conclusão de Curso 2 do Curso de Engenharia Elétrica da Universidade Federal de Uberlândia.

Orientador: Fábio Vincenzi Romualdo da Silva

Assinatura do Orientador

Uberlândia
2017

Dedico este trabalho aos meus pais, pelo estímulo, carinho e compreensão.

AGRADECIMENTOS

À minha família, por acreditar no que eu poderia me tornar e no quão longe eu chegaria, por sempre investir e realizarem sacrifícios por mim, para que eu pudesse seguir em frente com os meus sonhos, e pelo apoio e amor incondicionais. Devo tudo à ela.

Aos meus amigos, pelos anos de memórias incríveis e bons momentos, que sempre me motivaram e me ajudaram a manter a cabeça erguida nas épocas mais difíceis e me proporcionaram alegria nas épocas mais prósperas.

Ao Prof. Fábio Vincenzi pelo suporte na criação deste trabalho, pelo ânimo ao receber a proposta do projeto e por sempre incentivar a busca pelos melhores resultados, colaborando de maneira essencial na minha formação como profissional.

RESUMO

A BeagleBone Black é uma plataforma de desenvolvimento profissional produzida pela Texas Instruments de baixo custo, em comparação com demais plataformas de sistemas embarcados de mesmas especificações técnicas, capaz de iniciar o Linux em menos de dez segundos e permitindo que o programador comece a trabalhar em seu projeto em menos de cinco minutos, utilizando apenas um único cabo USB e um computador como hardware extra.

Este documento realizará um estudo acerca do tempo de resposta da BeagleBone Black, abordando múltiplas combinações de ambientes de desenvolvimento integrado e linguagens de programação, afim de apurar e comparar seus respectivos desempenhos.

Um objetivo secundário deste trabalho é também produzir e multiplicar o conteúdo que existe a respeito da plataforma em português, que atualmente é escasso, afim de estimular o seu uso.

ABSTRACT

The BeagleBone Black is a low-cost professional development platform, compared to other embedded systems platforms, produced by Texas Instruments, capable of starting Linux in less than ten seconds and allowing the programmer to commence working on his project in less than five minutes, using only a single USB cable and a computer as extra hardware.

This practicality, as well as its technical specifications, makes the equipment a solid choice for most of the embedded projects that a developer is idealizing. However, the equipment has limitations on frequencies above 8kHz, making its standard use unsuitable for high speed applications.

This paper has conducted a study on the response time of the BeagleBone Black, addressing various combinations of development environments and programming languages, in order to ascertain and their respective performances.

A secondary objective of this work is also to produce and multiply the content that exists regarding this subject in Portuguese, which is scarce, in order to stimulate its use.

LISTA DE ILUSTRAÇÕES

Figura 1 – A BeagleBone Black	14
Figura 2 – Dimensões da BBB em milímetros	15
Figura 3 – O Raspberry Pi	18
Figura 4 – Os componentes mais significantes da BeagleBone Black	22
Figura 5 – Conectores da BeagleBone Black e suas funções	25
Figura 6 – Código em Python para teste de velocidade	27
Figura 7 – Resultado do teste de velocidade da BBB em Python pelo Cloud9	28
Figura 8 – Criando um projeto dentro da BBB	30
Figura 9 – Funções em C relacionadas aos GPIOs	31
Figura 10 – Código em C para teste de velocidade	31
Figura 11 – Executando e compilando o código	32
Figura 12 – Resultado do teste de velocidade da BBB em C pelo Eclipse	33
Figura 13 – Declaração de bibliotecas e variáveis	34
Figura 14 – Exportando o pino GPIO	34
Figura 15 – Definindo pino como saída	35
Figura 16 – Alternância de nível lógico	35
Figura 17 – Resultado em C utilizando as bibliotecas padrões	36
Figura 18 – Estabelecendo um ciclo infinito	37
Figura 19 – Resultado de um ciclo infinito	37
Figura 20 – A BeagleBone Black conectada a um cabo JTAG	38
Figura 21 – Aba do projeto com todas as ferramentas	40
Figura 22 – Configurações de compilação das ferramentas, parte 1	40
Figura 23 – Configurações de compilação das ferramentas, parte 2	41
Figura 24 – Configurações de compilação das ferramentas, parte 3	41
Figura 25 – Configurações de compilação das ferramentas, parte 4	42
Figura 26 – Pastas Debug e Release em destaque	42
Figura 27 – Configurações do projeto	43
Figura 28 – Arquivos incluídos no sistema	44
Figura 29 – Adicionando ferramentas como dependências	44
Figura 30 – Biblioteca “gpio_v2.c”	45

Figura 31 – Declarando um pino como entrada ou saída	46
Figura 32 – Definindo valor lógico de uma saída	46
Figura 33 – Lendo o valor de uma entrada	47
Figura 34 – Pinos do cabeçalho P9	48
Figura 35 – Declarando endereços	48
Figura 36 – Código com atrasos	49
Figura 37 – Console após depuração do código	50
Figura 38 – Executar código	50
Figura 39 – Programa em execução	50
Figura 40 – Resultado do teste de velocidade da BBB em metal puro pelo CCS	51
Figura 41 – Endereço de memória de um pino de entrada/saída	53
Figura 42 – Comando que determina pino como saída	54
Figura 43 – Definindo nível lógico alto	55
Figura 44 – Definindo nível lógico baixo	55
Figura 45 – Código para testes	56
Figura 46 – Comandos para teste de velocidade	57
Figura 47 – Resultado do mapeamento de memória do dispositivo	58
Figura 48 – Frequência Máxima de CPU	60
Figura 49 – Escolhendo uma frequência específica	60
Figura 50 – Estabelecendo economia de energia	61
Figura 51 – Experimento em C padrão no modo de economia de energia	61
Figura 52 – Experimento da memória do dispositivo no modo de economia de energia	63
Figura 53 – Pasta inicial da BeagleBone Black	69
Figura 54 – Opções de drivers	70
Figura 55 – Opções de drivers online	70
Figura 56 – Primeiro contato com Cloud9	71
Figura 57 – Criando um arquivo no Cloud9	72
Figura 58 – Arquivo nomeado para programação em Python	73
Figura 59 – Executável do MinGW	74
Figura 60 – Compiladores C/C++ do Eclipse	74
Figura 61 – Instalação dos compiladores	75
Figura 62 – Confirmando instalação	75
Figura 63 – Opções do Eclipse	76

Figura 64 – Eclipse extraído	76
Figura 65 – Configurando o Eclipse	76
Figura 66 – Eclipse CDT	77
Figura 67 – Criando um projeto em C	78
Figura 68 – Finalizando o projeto em C no Eclipse	79
Figura 69 – Criar novo arquivo de fonte	79
Figura 70 – Dando nome ao arquivo fonte	80
Figura 71 – Arquivo criado dentro do projeto	80
Figura 72 – Sistema remoto dentro do Eclipse	81
Figura 73 – Extraíndo o SER	81
Figura 74 – Perspectiva de sistemas remotos	82
Figura 75 – Estabelecendo uma nova conexão	82
Figura 76 – Configurando a conexão	82
Figura 77 – Estabelecendo pacotes de segurança SSH	83
Figura 78 – Conectando à BBB	83
Figura 79 – Escolhendo uma nova senha	84
Figura 80 – Bloco P2 destacado na BeagleBone Black	85
Figura 81 – Contato “chave”	85
Figura 82 – Cabeçalho posicionado sobre o bloco P2	86
Figura 83 – Cabeçalho soldado à placa	87
Figura 84 – Blackhawk USB100v2 JTAG Emulator	87
Figura 85 – Instalação do CCS, primeiro passo	88
Figura 86 – Instalação do CCS, segundo passo	89
Figura 87 – Instalação do CCS, terceiro passo	89
Figura 88 – Instalando extensão de compiladores ao CCS	90
Figura 89 – Criando uma nova conexão, parte 1	91
Figura 90 – Criando uma nova conexão, parte 2	91
Figura 91 – Configurando a conexão	92
Figura 92 – Botão de inicialização	93
Figura 93 – Conexão realizada com sucesso	94
Figura 94 – Explorando os exemplos, parte 1	95
Figura 95 – Explorando os exemplos, parte 2	95
Figura 96 – Exemplos disponíveis	96

Figura 97 – Configurações para compilar as bibliotecas	97
Figura 98 – Exemplo de base	98

LISTA DE TABELAS

Tabela 1 - Python	64
Tabela 2 - C: Biblioteca de uso específico	65
Tabela 3 - C: Biblioteca padrão	65
Tabela 4 - Metal Puro	66
Tabela 5 - Memória do dispositivo	66
Tabela 6 - Frequência da CPU	67

LISTA DE ABREVIATURAS E SIGLAS

BBB – BeagleBone Black

TI – Texas Instruments

CPU – Computador de Placa Única

PRU – Unidade em tempo real programável

IDE – Ambiente de desenvolvimento integrado

LED – Diodo emissor de luz

BM – Metal puro

PRU-ICSS - Unidade em tempo real programável e subsistemas de comunicação industrial

CC – Corrente contínua

V – Volt(s)

GPIO – Entradas e saídas de uso genérico

EDA – Projetos de circuitos integrados

CCS – Code Composer Studio versão 6

SW – StarterWare

ARM – Acorn RISC Machine

JTAG – Joint Test Access Group

SUMÁRIO

1 INTRODUÇÃO	14
2 DESENVOLVIMENTO	21
2.1 ESPECIFICAÇÕES TÉCNICAS DA BEAGLEBONE BLACK	21
2.2 UTILIZANDO A BEAGLEBONE BLACK	25
2.3 TESTES DE VELOCIDADE	26
2.3.1 PYTHON	27
2.3.2 C - BIBLIOTECA DE USO ESPECÍFICO	29
2.3.3 C - BIBLIOTECA PADRÃO	33
2.3.4 METAL PURO.....	38
2.3.5 MEMÓRIA DO DISPOSITIVO	52
2.3.6 ALTERANDO A FREQUÊNCIA DA CPU	59
2.4 DISCUSSÃO	63
3 CONCLUSÃO	68
4 ANEXOS	69
4.1 ANEXO I: CLOUD 9.....	69
4.2 ANEXO II: ECLIPSE IDE	74
4.3 ANEXO III: CABO JTAG E CODE COMPOSER STUDIO V6	85
5 REFERÊNCIAS.....	99

1 INTRODUÇÃO

Primeiramente, antes de entrar diretamente no conteúdo objetivo deste trabalho, é necessário responder algumas perguntas pertinentes que podem ocorrer ao leitor ao longo do texto. É necessário, para contextualizar o trabalho, justificar algumas dessas questões, como, por exemplo, os motivos pelos quais optar pela BeagleBone Black (BBB), suas diferenças dos demais microcontroladores programáveis presentes no mercado, o público alvo a quem essa ferramenta de desenvolvimento é destinada, quando utilizá-la e até mesmo quando não utilizá-la. Desta forma, será possível utilizar o conteúdo presente neste documento da melhor maneira possível, para as aplicações mais adequadas.

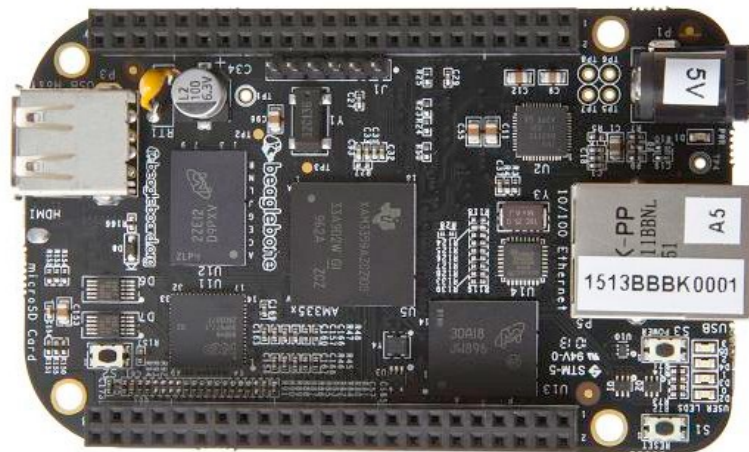


Figura 1 – A BeagleBone Black [1]

Nos dias de hoje, uma placa de microcontrolador programável custa por volta de US\$ 20,00, enquanto uma BBB, produzida pela Texas Instruments (TI) custa, em média, entre US\$ 45,00 e US\$ 55,00. No entanto, além de possuir um processador mais poderoso que a maioria, há outros fatores que justificam o custo extra [1]. Uma análise completa sobre esses recursos irá auxiliar no julgamento do desenvolvedor quando se trata da escolha de um determinado microcontrolador para um projeto específico.

Primeiramente, é importante ressaltar que a BBB é, além de um microcontrolador, um computador de placa única (CPU). Ou seja, todos os componentes necessários para seu funcionamento estão situados em uma única placa de circuito impresso. Isso significa que a BBB pode ser simplesmente conectada a um monitor, mouse e teclado e inicializará quase qualquer sistema operacional baseado em Linux em poucos segundos após ser ligada.

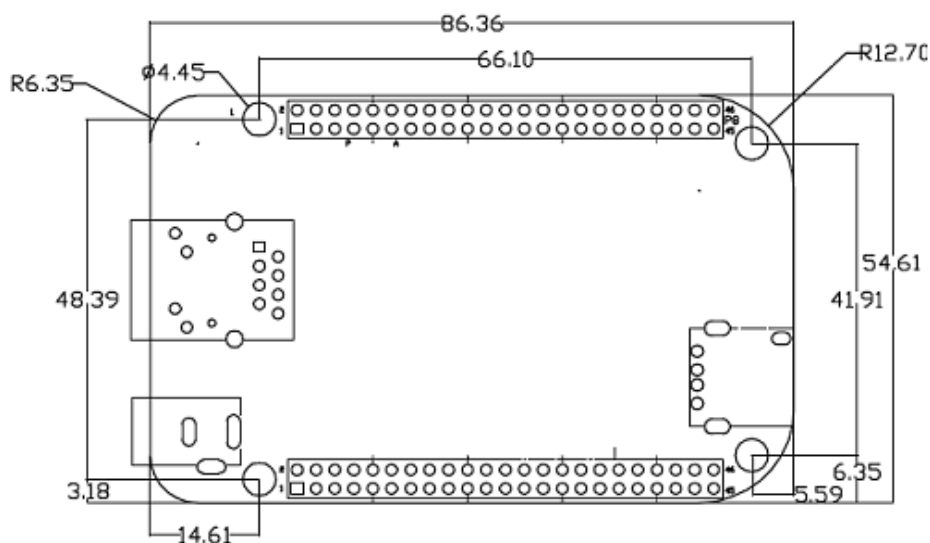


Figura 2 – Dimensões da BBB em milímetros [1]

Essa característica gera outro ponto positivo. Muitos dos programas desenvolvidos e disponíveis para o Linux podem ser executados com a BBB. Por exemplo, caso algum projeto exija que uma das portas USB acesse um webcam, basta carregar e compilar um programa de código aberto que permite que imagens capturadas pela câmera sejam salvas [1].

E quando o assunto é redes integradas, a BBB não apenas possui uma conexão Ethernet na própria placa, mas também já tem incluídas e disponíveis todas as ferramentas básicas de rede presentes nos pacotes do Linux. Podem ser usados serviços como FTP (Protocolo de Transferência de Arquivos), Telnet (protocolo de comunicação baseado em texto interativo bidirecional), SSH (*Secure Shell* – protocolo de rede criptográfico), ou até mesmo hospedar um servidor web próprio na placa [1].

Devido a esses recursos de rede, também é possível, e mais fácil, acessar os projetos eletrônicos realizados remotamente através da Internet. Por exemplo, em um projeto que mantém registro de dados coletados por algum sensor pode ter essas informações transferidas usando um cliente FTP, ou até mesmo um programa que envie esses dados por e-mail automaticamente para o projetista. Esse acesso remoto também permite que o desenvolvedor entre no sistema para atualizar o código [1].

Além disso, sem a necessidade de qualquer equipamento extra, a placa é capaz de acompanhar a data e a hora do dia e pode também atualizar os mesmos através de servidores de tempo na Internet fazendo o uso protocolos específicos, garantindo assim que essa seja uma forma precisa de utilizar o tempo em quaisquer projetos [1].

Assim como outros computadores, plataformas de Linux embarcado possuem um sistema de arquivos embutido, então armazenar, organizar e recuperar dados são tarefas razoavelmente triviais para a BBB [1].

Outra grande vantagem é a possibilidade de escrever códigos nas linguagens mais comuns e difundidas atualmente, como C, C++, Python, Perl, Ruby e até mesmo Shell script [1].

E diferente de um típico microcontrolador que possui um processador de 8-bit (como, por exemplo, a maioria dos modelos de Arduino), plataformas com Linux embarcado são capazes de compartilhar seu processamento entre tarefas e programas que estão sendo executados simultaneamente. Isso significa que projetos que possuem tarefas maiores ou mais demoradas não precisam parar outras funções até que as mesmas sejam concluídas [1].

Muito foi citado até agora sobre ferramentas e recursos relacionados ao Linux, mas o próprio sistema operacional em si tem seus pontos positivos. Não há falta de suporte quando se trata de conteúdo e informações em comunidades online e sites de ajuda, que costumam auxiliar na solução de problemas em momentos mais difíceis [1].

Quanto ao público alvo, a BBB é uma opção a ser considerada para pessoas que desejam transformar seus conceitos e ideias em engenharia em um produto, projeto, protótipo ou trabalho real. No entanto, integrar programas e hardware nem sempre é uma tarefa fácil, e a dificuldade envolvida na implementação depende da sofisticação que o projeto necessita [2].

Tendo isso em mente, a comunidade que gira em torno da BBB trabalha incessantemente todos os dias para garantir que a plataforma se torne acessível para qualquer pessoa interessada em integrar a placa em seus projetos, sejam essas estudantes, inventores, artistas, engenheiros e até mesmo indivíduos que a utilizem por passatempo [2].

Para usuários mais avançados, com maiores conhecimentos sobre eletrônica e programação, a BBB permite personalizações e desenvolvimentos adicionais para cumprir as necessidades de certos projetos. Mas, novamente, esse processo não é trivial, uma vez que alto nível de programação, assim como a utilização do Linux podem ser, em casos, complicados [2].

Exemplos de aplicação em que uma BBB é uma das melhores escolhas disponíveis são sistemas de gerenciamento e automação residencial, robôs, redes de sensores, trabalhos com conexão de Internet e similares. A placa possui poder de processamento suficiente para produzir a maior parte das aplicações que a imaginação do usuário permita criar em um dispositivo embarcado [2].

A maior vantagem da BBB sobre sistemas embarcado mais tradicionais, como Arduino, PIC e microcontroladores AVR, por mais que não estejam no mesmo patamar e seja um tanto injusto

comparar um sistema tão sofisticado com estas outras tecnologias, é aparente quando a alavancagem proporcionada pelo sistema operacional Linux é levada em conta. Por exemplo, ao criar um sistema de automação residencial utilizando a BBB, caso exista a vontade de fazer com que certa informação do projeto seja disponibilizada via Internet, pode-se simplesmente instalar o servidor de rede Apache. Então, o usuário pode utilizar sua linguagem de programação favorita dentro das que são suportadas pela placa para que a informação desejada possa ser obtida e compartilhada. E aprofundando mais ainda neste quesito, agora que o projeto deixa dados disponíveis na rede, é possível instalar um servidor secure shell para que os mesmos fiquem protegidos com poucas linhas de comando no Linux [2].

Essa maleabilidade quando se trata de implementação de programas uma vez que o hardware já está pronto pode, potencialmente, poupar semanas de trabalho em desenvolvimento de código. Além disso, como todas essas ferramentas são padronizadas, o cliente pode ter a segurança de que esses mesmos artifícios que ele está utilizando encontra-se presente em milhões de programas e máquinas ao redor do mundo [2].

O Linux também fornece o usuário com suporte de driver para os periféricos USB e adaptadores, assim podem ser conectadas câmeras, adaptadores de rede sem fio e diversos outros periféricos diretamente na plataforma, sem a necessidade de desenvolver um programa de driver complexo e caro para que os mesmos funcionem [2].

E vale a pena também citar, afim de comparação, o Raspberry Pi, criado pela Raspberry Pi Foundation. Este microcontrolador se assemelha muito à BBB, uma vez que ambos utilizam sistemas operacionais baseados no Linux, sendo um CPU, e possuem especificações técnicas parecidas. No entanto, há algumas diferenças. De acordo com o autor Matt Richardson, a mais visível é que o Raspberry Pi tem o propósito de encorajar gerações mais jovens a se interessarem e aprenderem mais sobre como computadores funcionam e programá-los, sendo assim mais simplificado e didático. Portanto, o hardware, os programas e a documentação são voltados para esse objetivo. Em contrapartida, a BBB possui um aspecto mais profissional, destinado mais às pessoas interessadas em placas de desenvolvimento para sistemas embarcados em Linux e, por isso, tem mais opções de hardware para conexão, além de um processador mais poderoso [1].

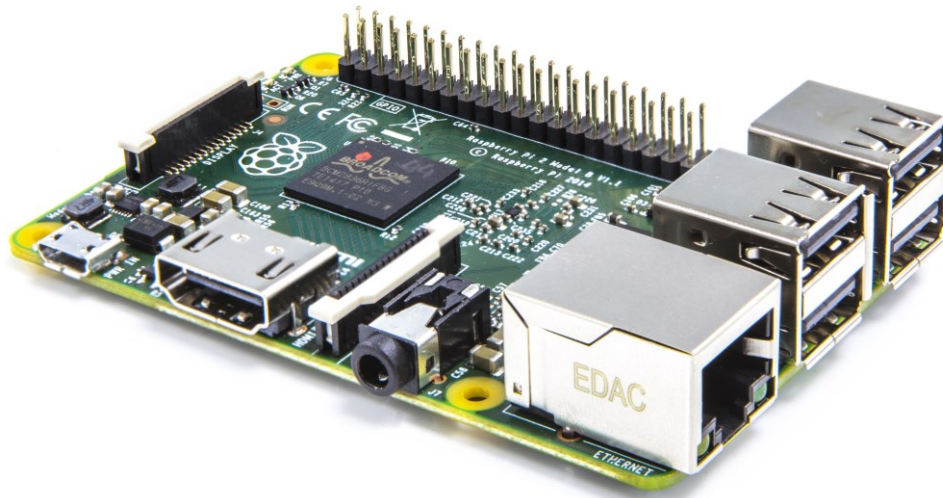


Figura 3 – O Raspberry Pi 2 [2]

Porém, por mais que a BBB seja recomendada para tantos casos distintos, há também situações para as quais ela não foi projetada. Uma delas é a reprodução de vídeos em alta definição. A sobrecarga de processamento de descodificação de programas e reprodução de vídeos codificados é imensa e está além da capacidade da placa em resoluções elevadas de vídeo. Já o Raspberry Pi, citado no parágrafo anterior, possui essa habilidade em função do seu processador Broadcom BCM2835, projetado especialmente para aplicações de multimídia [2]. A segunda aplicação para a qual a BeagleBone Black não é recomendada é para processamentos em tempo real ou controle previsível [2]. Esta limitação é, inclusive, objeto de estudo do presente trabalho.

O sistema operacional da placa, por padrão, não é preemptivo, o que significa que uma vez que o processador inicia a execução de uma tarefa do mesmo, ela não pode ser interrompida. Isto se torna problemático caso, por exemplo, o usuário deseje obter a amostra de um sensor de forma precisa a cada milionésimo de segundo. Caso surja a necessidade de coletar certo dado e o núcleo está ocupado com uma tarefa diferente, a mesma não pode e não será interrompida [2]. Portanto, em seu estado padrão, a BBB não é a plataforma ideal para projetar aplicações de sistemas em tempo real em frequências muito altas. Todavia, existem versões em tempo real do Linux disponível, ou até outros sistemas operacionais, destinadas a desenvolvedores de nível avançado. E além desta solução, existe também outra opção. Dentro do processador AM335x da BeagleBone há também dois microcontroladores disponíveis, chamados de unidades em tempo real programáveis (do inglês, programmable real-time units, ou PRU), que podem ser utilizadas para programar aplicações interfacial em tempo real [2].

Ambas as alternativas serão abordadas e estudadas, para fins de comparação. É importante deixar claro, no entanto, que há soluções dedicadas disponíveis para controle e amostra de dados em tempo real. Um exemplo é a plataforma Stellaris ARM, também da TI, que pode, em certos casos, ser mais apropriada.

Não obstante, como já foi bem fundamentado e argumentado previamente, a BBB é um microcontrolador programável extremamente rico em recursos e que tem um grande potencial para atender a fins sofisticados. Logo, descartar sua aplicação em detrimento de uma barreira que é, de certa forma, transponível é uma ação um tanto estólida.

Então, este documento visa expandir o leque de aplicações da BeagleBone, abordando diferentes metodologias, analisando e comparando seus resultados, afim de auxiliar desenvolvedores que necessitam de um sistema capaz de coletar amostras de dados e efetuar ações em tempo real, onde a velocidade é um fator crítico.

Sistemas são descritos como sendo em tempo real caso consigam fornecer uma resposta válida dentro de uma janela temporal bem específica, normalmente na ordem de microssegundos ou até mesmo nano segundos, dependendo de quantas tarefas o sistema executa.

Para fortalecer a motivação do objeto de estudo, é relevante citar exemplos onde o produto do mesmo pode ser aplicado. Primeiramente, é importante distinguir a relevância do sistema em tempo real dentro do projeto entre crítico (também conhecidos como rígido) e não-crítico (também conhecido como moderado).

Sistemas em tempo real críticos são aqueles usados em aplicações cujo todo o sistema ou projeto pode falhar como resultado do atraso de alguma operação ou tarefa, como direção elétrica de veículos, robôs auto equilibrados, um marca-passo, proteção de linhas de alta tensão e a navegação de uma aeronave [3].

Sistemas em tempo real não-críticos, por sua vez, também possui parâmetros para o tempo de cumprimento de operações e tarefas, mas problemas ao correspondê-los não acarreta na falha total do sistema, apenas em uma menor redução da qualidade do serviço. São exemplos transmissão de vídeo ao vivo, comunicação de telefonia móvel, leitores de multimídia e, inclusive, o núcleo principal do Linux pode, caracteristicamente, cumprir os requerimentos de um sistema não-crítico em certas tarefas como a reprodução de áudio [3].

Um trabalho como este também é importante para popularizar a BBB como uma alternativa de microcontrolador para a realização de um projeto, não só para engenheiros, mas também para estudantes de nível acadêmico variados, ou até mesmo inventores em geral, que possuem uma ideia que desejam concretizá-la.

Nos tempos de hoje, avanços tecnológicos foram capazes de trazer o desenvolvimento de projetos como robôs e demais processos automatizados à realidade. É essencial transmitir essa ideia de que os indivíduos, com acesso a um pedaço a mais de informação, são capazes de criar, de construir máquinas que conseguem rolar, andar, nadar e até mesmo voar com uma funcionalidade normalmente associada aos seres vivos inteligentes [3].

Finalmente, outro fator que justifica a escolha desse tema é a escassez de conteúdo sobre o mesmo. Poucas pesquisas ou trabalhos abrangem esse assunto de maneira completa e centralizada. Ademais, além de boa parte desta temática estar difundida, a mesma é quase inexistente em português. Materiais sobre a BeagleBone estão, em geral, disponíveis apenas em inglês, seja em bibliografias, comunidades de membros, vídeos instrucionais ou apostilas disponibilizadas pelo fabricante.

Em suma, encontrar um volume razoável de informações que seja o suficiente para estudar, compreender e aplicar o básico do que se é possível fazer e criar com a BBB em português já é relativamente difícil. Então, a produção deste conteúdo, que é um trabalho mais específico, pode ajudar pessoas que tem dificuldades ao lidar com o inglês a alcançar um nível mais avançado de desenvolvimento e expandir suas capacidades relacionadas ao uso da plataforma em questão.

2 DESENVOLVIMENTO

Afim de tornar claro e construir um progresso crescente ao longo da leitura deste trabalho, o desenvolvimento encontra-se dividido em quatro grandes esferas, ordenadas do mais básico e essencial para o uso da BeagleBone até a implementação de um sistema que seja capaz de responder em tempo real.

A primeira esfera aborda detalhes como as especificações técnicas do microcontrolador, sua documentação e uma breve descrição sobre cada uma de suas partes mais essenciais, de sorte que o hardware não seja fonte de dúvidas ao longo do texto.

Em seguida, há também um conteúdo opcional presente no anexo. O mesmo é caracterizado como opcional, pois é destinado a desenvolvedores iniciantes, que ainda não possuem o conhecimento técnico suficiente para começar a programar e executar seus códigos. Com este conteúdo, o leitor será orientado a usar a BBB da forma convencional através de um computador, ou seja, utilizando o Linux por um ambiente de desenvolvimento integrado (do inglês, *Integrated Development Environment* – IDE) de duas formas diferentes, com duas linguagens diferentes. Além dessas duas formas, há também a programação em metal puro (traduzido pelo autor do termo *Bare Metal*, ou BM), que utiliza os recursos de processamento e hardware da placa, mas sem executar o sistema operacional.

Posteriormente, a terceira esfera é composta por uma série de testes de velocidade, realizados utilizando metodologias diferentes, cada qual com suas próprias características, vantagens, desvantagens e resultados. Os dados obtidos através desses testes serão coletados, exibidos e explanados separadamente, demonstrando as capacidades da BBB em cada um deles.

Por fim, a quarta esfera encerrará o estudo. Todas as informações apuradas, analisadas e discutidas, serão listadas em conjunto, comparando as metodologias utilizadas.

Em suma, ao fim documento, o leitor terá sido introduzido à BeagleBone Black e suas informações vitais, terá acesso a manuais de como utilizá-la das formas mais convencionais e práticas, conhecerá diferentes metodologias de abordagem para suas aplicações e concluirá com conhecimento suficiente para ser capaz de decidir qual dos caminhos é o melhor para determinado projeto ou sistema que esteja pensando em desenvolver.

2.1 Especificações técnicas da BeagleBone Black

Antes de começar a utilizar a placa, conectar e programar o dispositivo para todas as suas possíveis funções, é de grande importância observá-lo mais de perto. Vale a pena citar também

o fato de que existem duas versões da placa no mercado, a BeagleBone, a versão original, e a BeagleBone Black, mais nova, com um processador mais rápido, mais memória e demais melhorias. A maior parte dos processos abordados neste documento funciona da mesma forma em ambas as placas, no entanto foi escrito especialmente para a segunda versão.

A BBB possui diversas partes, evidentemente. A seguir, na Figura 4, estão representadas as mais relevantes e, em seguida, as mesmas são brevemente descritas.

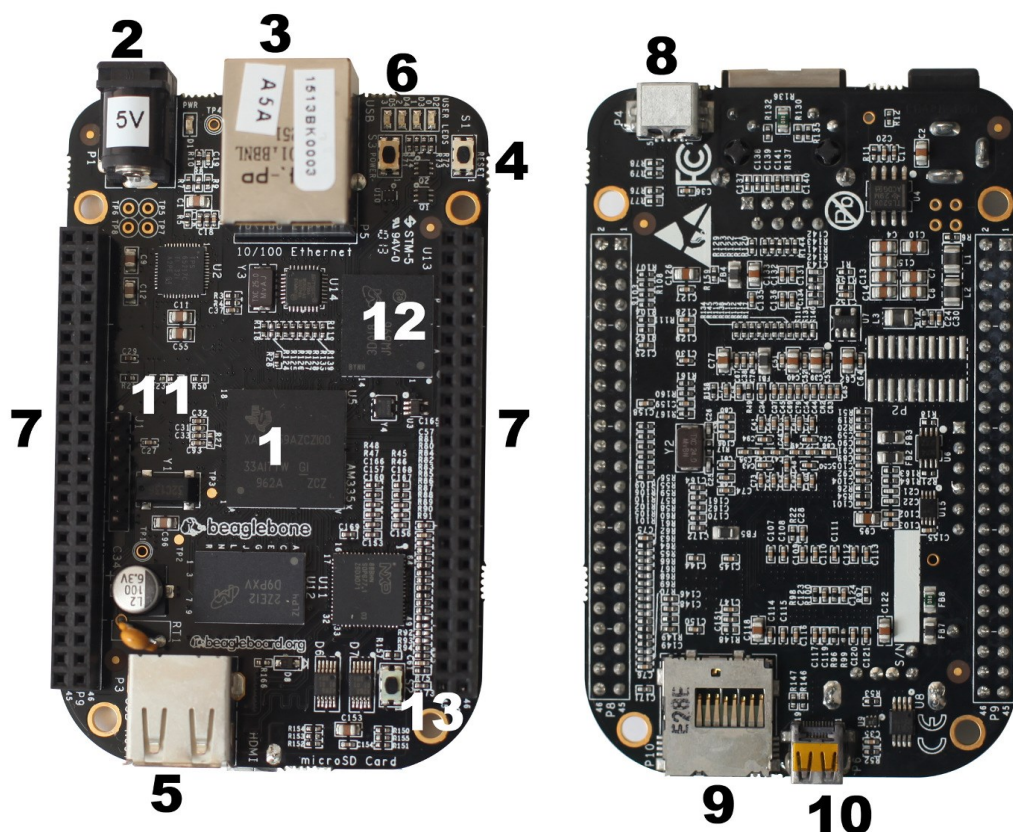


Figura 4 – Os componentes mais significantes da BeagleBone Black [1]

1. O processador. É essencialmente o cérebro de toda operação. Atarefado com a maior parte do trabalho pesado, o processador da primeira versão da BeagleBone possui propriedades semelhantes às do iPhone 4. Colocando esta comparação em números, é um 720MHz ARM Cortex-A8 equipado com 256 MB de DDR2 RAM. Já o modelo mais recente possui um chip de 1GHz com 512MB de DDR3 RAM [1].
2. Conector para fonte. A BBB necessita de 5 volts e 500 mA de corrente contínua para operar. A maioria das fontes genéricas de 5V CC com um conector de 2.1 milímetros é capaz de alimentar a placa. Bem ao lado do conector há um pequeno chip de proteção, caso alguém forneça à BBB mais do que 5 e menos do que 12 V acidentalmente, não deixando que a

placa ligue caso muita tensão surja. Mesmo assim, é melhor garantir-se de que a fonte produz apenas 5 V antes de conectá-la [1].

3. Porta ethernet. Esta é uma porta ethernet RJ45 padrão, extremamente útil para projetos que precisam de acesso à Internet. A ela pode ser conectado um cabo de rede diretamente de um roteador, ou até mesmo compartilhar a conexão WiFi de um computador conectando-se a ele através dela [1].
4. Botão de reiniciar. Ao pressionar este botão, a placa é reinicializada, assim como um computador. É melhor que a reinicialização seja feita de maneira correta através do sistema operacional, para evitar que arquivos sejam corrompidos. No entanto, o botão é útil para situações onde o sistema para de responder [1].
5. Porta USB. Assim como um computador, a BeagleBone possui uma porta USB. Isto permite que o usuário atribua à placa uma grande variedade de hardware adicional, incluindo mouse, teclado, câmeras e adaptadores de rede sem fio [1].
6. LEDs de bordo. Próximo ao conector para fonte, há um LED que indica quando tensão é fornecida à placa. Além deste, há também 4 LEDs ao lado do botão de reiniciar. Diferentemente do primeiro citado, estes outros 4 são programáveis. Na configuração padrão da placa, o LED 0 (o número de cada um está impresso sobre ele) funciona como uma “batida de coração”, mostrando que o sistema está em operação. O LED 1 irá piscar sempre que o cartão MicroSD está sendo acessado. O LED 2 irá piscar quando a CPU estiver ativa, e o LED 3 quando a memória flash estiver sendo acessada [1].
7. Cabeçalhos de expansão. Estes dois cabeçalhos de expansão, chamados de P8 e P9, permitem que o usuário integre a BBB a projetos eletrônicos. Os pinos podem ser configurados para um grande número de diferentes funções, como será abordado futuramente [1].
8. Porta Mini USB. Esta porta USB permite que a BBB atue de forma diferente quando conectada a um computador. O computador não apenas irá alimentar a placa através do USB, mas também irá comunicar-se com ela. Assim também é possível acessar as informações contidas na BeagleBone, uma vez que ela irá aparecer como um dispositivo de armazenamento ao ser conectada com um computador. Neste caso, a velocidade de processamento será reduzida para diminuir o consumo de energia. Logo, para aplicações mais rápidas é recomendado utilizar o conector [1].

9. Entrada para cartão MicroSD. Diferente da maioria dos computadores, a BeagleBone não possui um disco rígido, então no lugar dele é utilizado um MicroSD para armazenar o sistema operacional, programas e dados [1].
10. Porta Micro HDMI. Para conectar a BBB a uma tela ou televisão, é utilizada a porta Micro HDMI. A aparência a torna facilmente confundível com a porta mini USB, então se o usuário encontrar problemas em conectar o cabo a ela, provavelmente está tentando utilizar a porta errada [1].
11. Cabeçalho serial. Enquanto ambas as versões da BeagleBone possuem saídas seriais para acesso do terminal, apenas a BBB tem uma dessas portas seriais disponíveis em seu próprio cabeçalho. Esta parte é feita especificamente para conectar um cabo como o FTDI TTL-232 para atividades que envolvam a utilização do terminal baseado em texto via USB [1].
12. Memória flash de bordo. Com a memória flash de bordo é possível ligar e inicializar a BBB sem um MicroSD inserido na placa. Em manuais técnicos, essa memória é referida como eMMC [1].
13. Botão de inicialização. Apertar e segurar esse botão enquanto a BBB estiver ligada instrui o hardware a inicializar o sistema através dos dados inseridos no cartão MicroSD, ao invés de utilizar o que está presente na memória flash de bordo [1].

E ainda se tratando de hardware, existem particularidades mais específicas dentro das enumeradas acima. Por enquanto, a mais relevante são os pinos citados no número 7. Todo projeto que precisa controlar algum circuito elétrico ou eletrônico fará uso desses pinos, que são divididos em cabeçalhos P8 e P9.

Há, ao todo, 92 pinos divididos igualmente entre P8 e P9 (2x46). No entanto, nem todos são entradas e saídas de uso genérico (*general-purpose input/outputs*, ou GPIO), uma vez que alguns possuem uma configuração pré-determinada. Dentre eles, 27 possuem são projetados para cumprir tarefas específicas, entre modulação de largura de pulsos, entradas analógicas, alimentação de tensão, comunicação serial UART, entre outros [2].

Já os 65 pinos restantes estão disponíveis para serem multiplexados de maneiras distintas, de acordo com a necessidade do projetista. A Figura 5 enumera cada um desses pinos e os relaciona a seus respectivos propósitos.

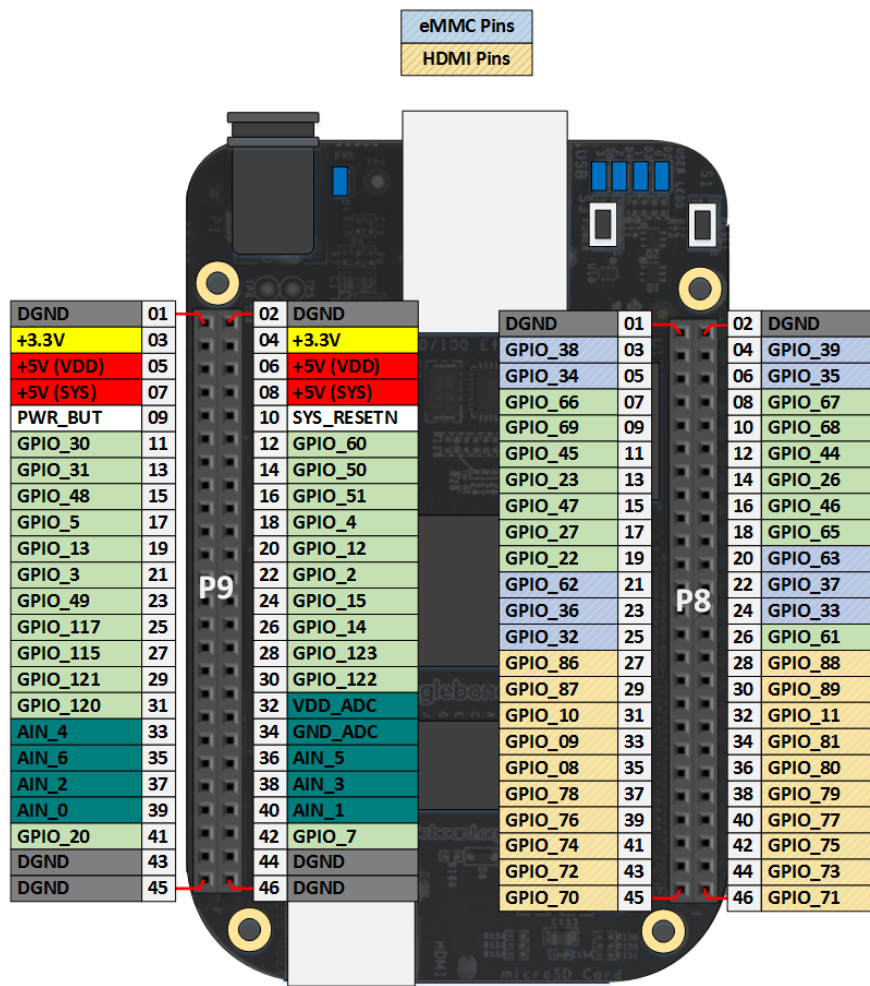


Figura 5 – Conectores da BeagleBone Black e suas funções

2.2 Utilizando a BeagleBone Black

Agora que o hardware foi brevemente introduzido, é preciso esclarecer como é possível conectar e controlar a BeagleBone Black. Como a placa, como já citado, é, em si, um computador completo, uma das formas de começar a utilizá-la é ligando-a a uma fonte de alimentação, teclado, mouse e tela. O Linux será iniciado e, através de seus terminais, é possível acessar os periféricos da placa normalmente.

No entanto, para novo usuários, ainda inexperientes com o Linux, e pouco familiares à BeagleBone Black, a melhor forma, e também mais prática, de utilizar a placa é conectando-a a um outro computador através do cabo micro USB. Esse processo permite que o sistema operacional seja acessado por uma plataforma de desenvolvimento secundária, ao invés de manipular o Linux diretamente.

Outra vantagem deste método, que o faz ser escolhido em diversos casos mesmo quando se é um projetista avançado, é o conjunto de periféricos que ficam disponíveis para o uso em determinados projetos.

Além disso, há uma variedade de plataformas compiladoras e ambientes de desenvolvimento integrado como opções para que a BeagleBone Black seja usada através do computador, como Cloud9, CodeComposerStudio, Eclipse CDT IDE, Linaro ARM Toolchain, Qt Creator, entre outros. Cada um com suas próprias características e ferramentas que podem atender melhor, ou pior, o desenvolvedor dependendo de suas preferências e aplicações.

Afim de atender às necessidades de usuários mais novos, este documento possui uma série de guias e tutoriais presentes no Anexo. Os mesmos ensinam a configurar o software da placa para que o leitor seja capaz de reproduzir os testes aqui presentes e contidos no item “2.3 Testes de Velocidade”.

O Anexo I e II mostram, passo-a-passo, a configurar os ambientes de desenvolvimento integrado Cloud9 e Eclipse, respectivamente, para programação da placa através do Linux. Já o Anexo III aborda o Code Composer Studio e o uso do cabo JTAG, possibilitando a programação sem o uso do sistema operacional.

2.3 Testes de Velocidade

Antes de dar início aos testes de velocidade é importante esclarecer como os mesmos serão feitos e explicar a ideologia por trás dos critérios escolhidos, afim de tornar o experimento mais tangível e também para que sua relevância para a prática seja melhor compreendida.

A BeagleBone Black pode ser usada para um grande número de aplicações distintas, como já foi dito anteriormente. A placa pode ser usada, por exemplo, para criar um servidor em rede, usando apenas programação, uma fonte de alimentação e acesso à Internet. Já outros projetos exigem o uso dos periféricos, que são portas de entradas e saídas de informações da CPU.

Ligados a esses periféricos podem estar sensores, que enviam dados para o computador. Existe uma variedade muito grande de sensores que podem ser usados em possíveis projetos, como de temperatura, posição, aceleração, força, corrente, tensão e intensidade luminosa. Ou também podem ser ligados atuadores, que recebem dados do computador. Os mesmos realizam ações comandadas pela BBB com base nas informações recebidas pelos sensores.

Todo esse processo pode ser explicado pelo seguinte modelo: em uma fábrica, existe um tanque de mistura de produtos, que precisa ser mantido em uma temperatura pré-determinada. Ultrapassar essa temperatura estraga o produto, enquanto ficar muito abaixo dela, reduz sua

qualidade. Para controlar esse processo é instalado um sensor de temperatura, que mantém a placa atualizada quanto ao estado atual do tanque, e dois atuadores, um que é fonte de aquecimento e um que é fonte de resfriamento. Após programada, a placa é capaz de manipular seus atuadores para manter o tanque na temperatura ideal de operação, ao receber informações e enviar comandos.

No entanto, alguns processos precisam que as leituras e reações sejam realizadas em períodos de tempo muito curtos. Ou seja, os periféricos da CPU precisam ser ativados de forma rápida o suficiente para não comprometer o funcionamento do sistema. Logo, para fazer testes de velocidade e análise de tempo de resposta para esse tipo de projeto, é preciso fazer o uso das unidades de entradas e saídas de uso geral da BeagleBone Black, também conhecidos como GPIOs, que são periféricos que podem ser usados tanto para enviar quanto para receber dados, manipulando-se apenas o código.

Para isso, não são necessários atuadores e sensores sofisticados ou de montagens de circuitos mais elaborados. Existe uma forma simples de mensurar o tempo que o microcontrolador leva para iniciar e interromper uma operação. Para isso, basta ligar uma de suas saídas a uma resistência em uma placa para protótipos, conectar essa resistência ao terra e, em seguida, fazer com que a BBB envie comandos de nível lógico alto e baixo, repetidamente, e, por fim, capturar a frequência desses comandos através de um osciloscópio.

Desta forma, esta metodologia será capaz de determinar o tempo de resposta de cada um dos procedimentos utilizados. Além disso, como a execução será a mesma para todos os casos, o método é uma forma imparcial de obter os resultados desejados, não favorecendo nenhuma abordagem.

2.3.1 Python

Para o caso deste estudo, o objetivo de interesse é testar e potencializar o tempo resposta da BeagleBone. A primeira abordagem será o teste de velocidade quando se programa em Python e o código é executado através do Cloud9.

```
1 import Adafruit_BBIO.GPIO as GPIO
2 import time
3
4 GPIO.setup("P8_11", GPIO.OUT)
5
6 while True:
7     GPIO.output("P8_11", GPIO.HIGH)
8     GPIO.output("P8_11", GPIO.LOW)
9
```

Figura 6 – Código em Python para teste de velocidade

O propósito do código da Figura 6 é bem simples. Como explicado na seção 2.1, a BBB possui dois cabeçalhos de expansão, chamados de P8 e P9. Os pinos destes cabeçalhos estão enumerados de forma simples na Figura 5 e relacionados às suas devidas funções. Como pode ser visto nesta última figura citada, o pino 11 do cabeçalho P8 (P8_11) é um GPIO, ou seja, pode ser programável como uma saída ou entrada para uso genérico.

A linha 1 do código tem a função de importar a relação existente entre cada um dos pinos e seus nomes. A linha 4 é responsável por definir o pino P8_11 como saída. As linhas 7 e 8 manipulam o pino, hora deixando seu nível lógico alto, hora baixo. A linha 6 faz com que o processo efetuado pelas 7 e 8 se repita indefinidamente. O código não possui nenhuma pausa ou delay, então efetuará essas operações da forma mais rápida possível.

A montagem do circuito físico não passa de um resistor ligado ao pino P8_11 e, em seguida, ao GND da placa (pino P8_01, que não precisa ser declarado no código). Um osciloscópio foi então conectado ao resistor para que fosse possível visualizar a forma de onda gerada.

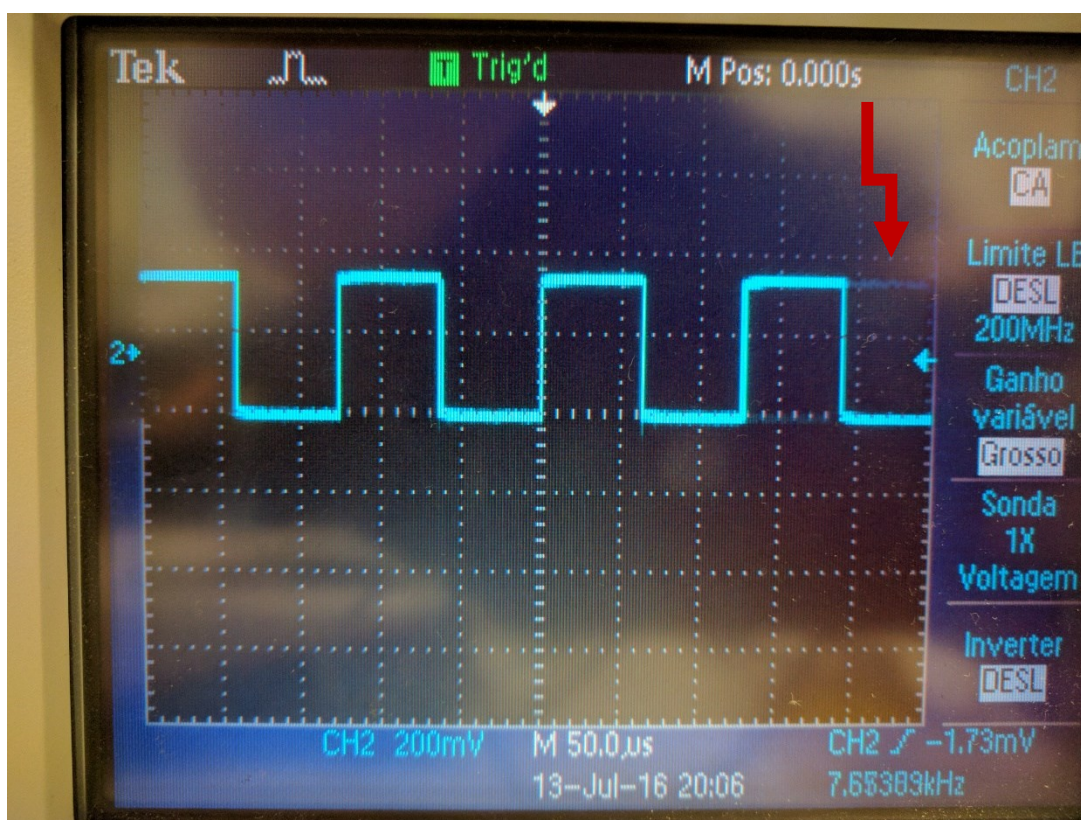


Figura 7 – Resultado do teste de velocidade da BBB em Python pelo Cloud9

Através da Figura 7 é possível concluir que a velocidade atingida foi uma frequência de 7.6kHz. No entanto, vale ressaltar o *jitter*, ligeiramente mais claro que a forma de onda, apontado pela seta vermelha. Ou seja, por mais que o sistema tenha atingido esta velocidade, o sistema não pode ser projetado para atuar nesta frequência, principalmente em aplicações de tempo real crítico.

Logo, por mais que esta abordagem tenha suas vantagens em questão de flexibilidade de linguagens e a grande praticidade de estar quase que imediatamente disponível para a produção de projetos, a velocidade atingida pelo sistema é limitada para certas aplicações.

No entanto, deve-se considerar um fator que pode estar segurando o poder do sistema em atuar com maior velocidade. O fato de que Python é uma linguagem de alto nível e interpretada, ou seja, é uma linguagem de programação em que o código fonte é executado por um programa de computador, chamado de interpretador, e apenas em seguida é executado pelo sistema operacional ou, no caso, processador, o que deixa o processo mais lento.

2.3.2 C – Biblioteca de uso específico

Para tentar contornar essa desvantagem apresentada pela linguagem Python, agora a abordagem será feita em C. Como uma introdução, a biblioteca a ser utilizada é uma de uso específico, desenvolvida pelo engenheiro Derek Molloy, da Escola de Engenharia Eletrônica em Dublin City University. A intenção ao usar a mesma é para elevar o nível do estudo gradativamente, para que o leitor possa acompanhar a evolução do uso da BBB. Esta biblioteca facilita muito o trabalho do programador, possibilitando comandos mais simples e diretos, resultando em um código mais compacto e de fácil entendimento.

Além disso, apesar do objetivo ser o mesmo que o anterior, desta vez o compilador a ser usado será o Eclipse. Não existem diferenças expressivas entre o Cloud9 e o Eclipse IDE, fora certas peculiaridades e características, como, por exemplo, a necessidade de conexão com a Internet durante todo o uso do Cloud9 (o que não ocorre no Eclipse), interface, configurações e outros. Fora isso, o ambiente de desenvolvimento em si não causará diferenças no tempo de resposta final, uma vez que o programa é executado na BBB em ambos os casos.

No entanto, o processo para que o Eclipse funcione é bem mais longo e trabalhoso do que o Cloud9, uma vez que os compiladores específicos de linguagem C/C++ devem ser instalados manualmente de forma independente do IDE. Porém, ele permanece simples e está bem ilustrado neste documento no Anexo II.

Uma vez que o ambiente esteja configurado corretamente, dentro do Eclipse, no diretório à esquerda em “My Home” dentro de “Sftp Files”, é preciso criar uma pasta chamada GPIO (na verdade, o nome da pasta é opcional, o usuário pode escolher qualquer nome com o qual se sinta confortável para intitular seu projeto). Nesta pasta, deve-se criar quatro novos arquivos com os seguintes nomes: “build”, “SimpleGPIO.cpp”, “SimpleGPIO.h” e “TestApplication.cpp”. O resultado final se parecerá com a Figura 8.

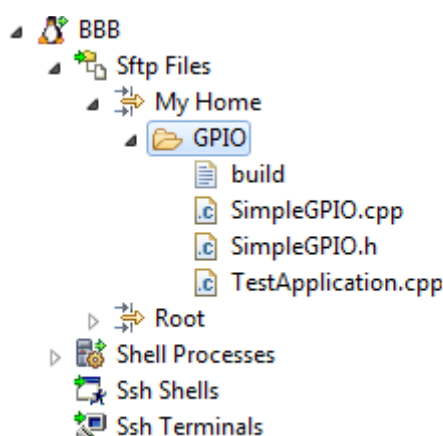


Figura 8 – Criando um projeto dentro da BBB

Em seguida, o usuário deve acessar o site do github no endereço contido na referência [7]. Esse site contém as bibliotecas necessárias para criar um código capaz de manipular os GPIO da BeagleBone Black. Para importá-las para o Eclipse, basta abrir cada uma das pastas contidas no endereço acima, selecionar todo o conteúdo dentro delas e colar dentro de seus respectivos arquivos criados no programa e salvo. Por exemplo, o conteúdo presente em [8] deve ser copiado e colado no arquivo de nome “SimpleGPIO.cpp” e salvo (os arquivos podem ser salvos pelo atalho Ctrl + S).

Finalmente, depois de repetir o processo para todos os arquivos criados, é necessário baixar do github o que está nomeado como “TestApplication” (sem a terminação “.cpp”) clicando em “View Raw”. Então o arquivo deve ser importado para a pasta GPIO também, o que pode ser feito simplesmente ao clicar e arrastar.

Estas bibliotecas que acabaram de ser incluídas na pasta simplificam bastante o código para o desenvolvedor, uma vez que possuem diversas funções montadas e declaradas. Todas podem ser encontradas e lidas no arquivo SimpleGPIO.cpp, onde tem uma pequena descrição. Alguns exemplos seguem na Figura 9.

```

        int gpio_export(unsigned int gpio)
int gpio_set_dir(unsigned int gpio, PIN_DIRECTION out_flag)
        int gpio_set_value(unsigned int gpio, PIN_VALUE value)
int gpio_get_value(unsigned int gpio, unsigned int *value)

```

Figura 9 – Funções em C relacionadas aos GPIOs

Estas funções servem, respectivamente para declarar e tornar disponível algum pino, definir a direção de algum pino, seja entrada ou saída, manipular o estado de alguma saída, como nível lógico alto ou baixo, ou ler o valor de alguma entrada.

Então, para colocar estas funções em prática, é necessário criar um novo arquivo com o fim “.c” ou “.cpp”. No caso, o nome deste arquivo será “test.cpp”. O código contido nele é como segue na Figura 10.

```

#include "SimpleGPIO.h"    //declare the library
using namespace std;

unsigned int LEDGPIO = 60; // GPIO1_28 = (1x32) + 28 = 60

int main(int argc, char *argv[]){
    gpio_export(LEDGPIO);    //exporting the pin
    gpio_set_dir(LEDGPIO, OUTPUT_PIN);    // Setting LED is an output

    for(int i=0; i<100000; i++) {
        gpio_set_value(LEDGPIO, HIGH);
        gpio_set_value(LEDGPIO, LOW);    // Turning the pin on/off several times
    }

    gpio_unexport(LEDGPIO);    // unexport the LED
    return 0;
}

```

Figura 10 – Código em C para teste de velocidade

Da mesma forma como feito para o Cloud9, o único propósito do código é definir o nível lógico de uma saída como alto e então baixo, sem pausas ou atrasos. Essa operação será realizada 100.000 vezes, enquanto a condição para o “for” for verdadeira. É importante salvar o código assim que o mesmo estiver pronto.

Para compila-lo, é necessário digitar “cd” no terminal (isto levará o desenvolvedor para a pasta Home da BBB) e então “ls” (mostrará as pastas disponível na atual pasta do desenvolvedor). A

pasta GPIO (no caso do exemplo dado) estará listada entre estas demais pastas. Para abri-la, basta digitar “cd GPIO” no terminal.

Afim de executar o código, estando na pasta GPIO, o usuário precisa digitar “g++ -O2 -Wall *nome-do-código.cpp* SimpleGPIO.cpp -o *nome-do-código*”. Como dito anteriormente, o nome do código exemplificado é “test”.

Por fim, para compilar, basta digitar “./*nome-do-código*”. Caso ocorra algum problema relacionado a permissão negada, deve-se digitar “sudo su -c ./*nome-do-código*”. O exemplo segue na Figura 11.

```
root@beaglebone:~/GPIO# cd
root@beaglebone:~# ls
GPIO
root@beaglebone:~# cd GPIO
root@beaglebone:~/GPIO# g++ -O2 -Wall test.cpp SimpleGPIO.cpp -o test
root@beaglebone:~/GPIO# sudo su -c "./test"
```

Figura 11 – Executando e compilando o código

A montagem do circuito físico é a mesma que foi utilizada anteriormente com o Cloud9, sendo composta apenas de um resistor ligado a um pino e, em seguida, ao GND da placa. O mesmo osciloscópio foi então conectado ao resistor para que fosse possível visualizar a forma de onda gerada.

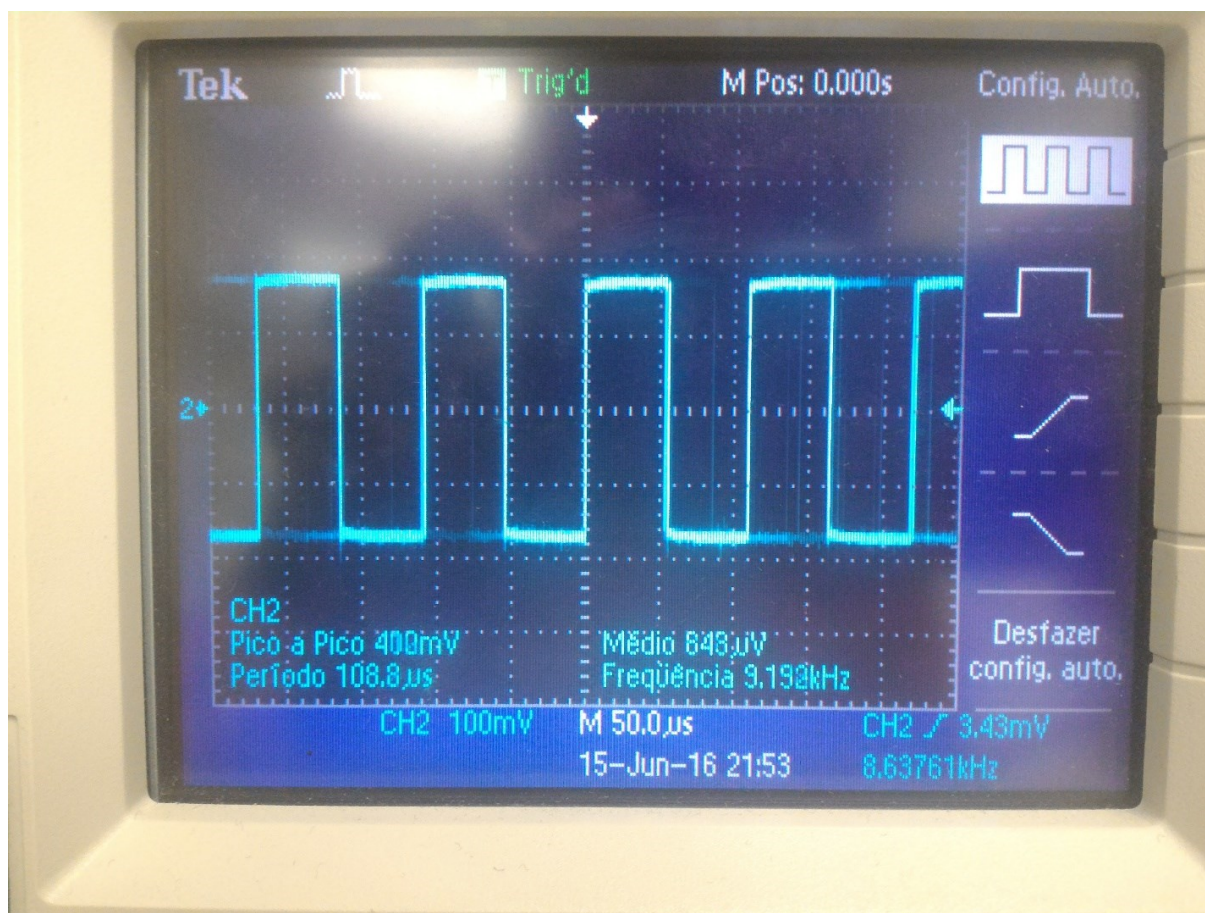


Figura 12 – Resultado do teste de velocidade da BBB em C pelo Eclipse

A velocidade obtida está disposta no canto inferior direito da tela do osciloscópio. O resultado de 8.6kHz é 13% maior do que o que foi obtido no teste anterior. No entanto, a mesma particularidade quando se trata de *jitter* persiste, como mostram os espectros da forma de onda. É possível quantificar e medir o mesmo, superpondo ondas através da persistência ativa do osciloscópio, mas sua existência ainda torna inviável o desenvolvimento de projetos que atuem na frequência obtida.

Uma desvantagem visível do Eclipse com relação ao Cloud9 é seu processo de configuração trabalhoso que, apesar de tudo, precisa ser feito apenas uma vez. E como vantagens podem ser apontadas os pacotes de Secure Shell implementados na conexão, tornando-a mais segura.

2.3.3 C – Biblioteca padrão

Como dito anteriormente, a biblioteca usada no último item, também baseada em C, foi a “SimpleGPIO.h”. A mesma contém comandos muito mais simples e diretos, de certa forma traduzidos do C convencional, para facilitar o uso da BeagleBone Black.

Agora, o teste a ser realizado é, em comparação, um pouco mais complexo e com programação mais volumosa, utilizando a linguagem em C sem auxílios de bibliotecas. De maneira análoga, o processo é parecido com o que foi apresentado em 2.3.2, uma vez que a lógica de programação ainda é a mesma. Logo, o primeiro passo é, certamente, declarar as bibliotecas e variáveis que serão utilizadas.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <errno.h>
5  #include <unistd.h>
6  #include <fcntl.h>
7  #include <poll.h>
8
9  using namespace std;
10
11  int fd, len;
12  int gpio = 60;
13  char buf[64];
14  #define SYSFS_GPIO_DIR "/sys/class/gpio"
```

Figura 13 – Declaração de bibliotecas e variáveis

Em seguida, é preciso exportar o pino GPIO ao qual a resistência será conectada para avaliação de velocidade.

```
16  int main()
17  {
18      // Exporta o gpio
19      fd = open(SYSFS_GPIO_DIR "/export", O_WRONLY);
20      if (fd < 0) {
21          perror("gpio/export");
22          return fd;
23      }
24
25      len = snprintf(buf, sizeof(buf), "%d", gpio);
26      write(fd, buf, len);
27      close(fd);
```

Figura 14 – Exportando o pino GPIO

Então, o GPIO deve ser declarado como entrada ou saída.

```

29 // Configura a Direção do gpio
30 snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/direction", gpio);
31
32
33 fd = open(buf, O_WRONLY);
34 if (fd < 0) {
35     perror("gpio/direction");
36     return fd;
37 }
38
39
40 write(fd, "out", 4);
41
42
43 close(fd);
44
45 snprintf(buf, sizeof(buf), SYSFS_GPIO_DIR "/gpio%d/value", gpio);
46 fd = open(buf, O_WRONLY);

```

Figura 15 – Definindo pino como saída

E, por fim, o comando de alternância de nível lógico do pino, sem funções de atraso, o mais rápido possível.

```

48 while(i <= 1000000)
49 {
50     write(fd, "0", 2);
51     write(fd, "1", 2);
52     i++;
53 }

```

Figura 16 – Alternância de nível lógico

Depois de executar o código com o hardware devidamente conectado, o resultado da onda gerada foi como segue na figura a seguir.

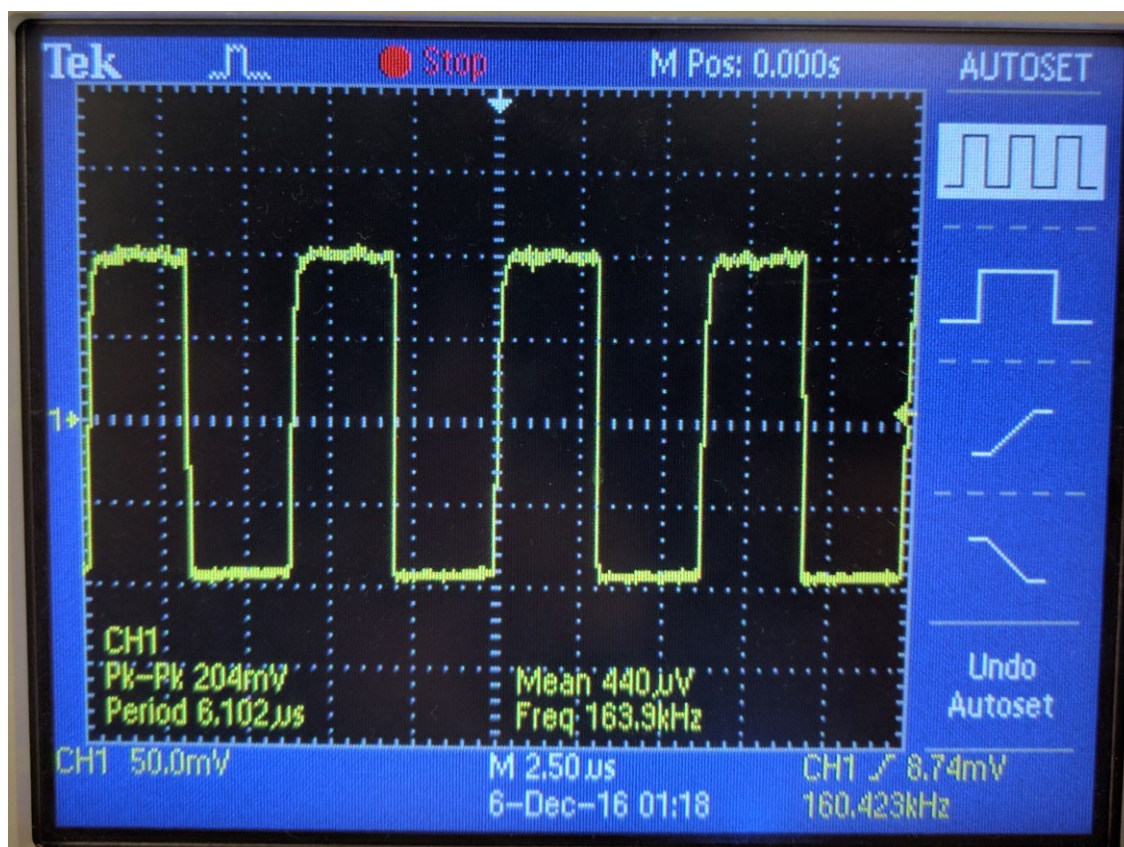


Figura 17 – Resultado em C utilizando as bibliotecas padrões

O tempo de resposta do C padrão superou o da biblioteca de uso específico em quase 19 vezes. Já era de se esperar que este experimento apresentasse um desempenho melhor do que os demais, uma vez que bibliotecas são conjuntos de códigos pré-escritos, dados de configuração, sub-rotinas, classes, valores e demais especificações.

No geral, uma biblioteca é também uma coleção de implementações de comportamento. No caso do C para uso específico de utilização das portas de entrada e saída da BeagleBone Black, cria-se para o programador uma interface mais bem definida, intuitiva e de uso simplificado. Inclusive, o código para teste de velocidade da mesma ficou significativamente menor do que o da linguagem C padrão, mesmo ambos realizando exatamente a mesma tarefa.

Porém, esta ferramenta de personalização de bibliotecas, que facilita o trabalho do desenvolvedor, coloca um fardo extra sobre o microcontrolador, que deve reinterpretar estes novos comandos antes de realizar uma ação, fazendo com que C deixe de ser uma linguagem imperativa simples.

Outro fator interessante que pode ser concluído da discrepância obtida entre estes tempos de resposta é que, analogamente, esse fenômeno pode ocorrer para qualquer linguagem de programação, caso seja possível assumir que sua forma padrão seja a versão mais simples e

otimizada que existe. Logo, quando um dos requerimentos de um projeto for uma elevada frequência de realização de tarefas, é importante ter em mente que linguagens personalizadas, por mais que facilitadoras, podem ser um empecilho.

E agora que o assunto de otimização de código foi tocado, existem outras variáveis que podem ser testadas. Para o caso deste experimento, como a tarefa sendo executada é extremamente direta e simples, não há muitos ajustes que possam ser feitos para melhorar seu desempenho, mas mesmo assim, certos comandos ainda podem ser reduzidos. Por exemplo, uma alteração arbitrária é possível de ser feita, como transformar o comando de um ciclo finito para um ciclo infinito.

```
48 while(1)
49 {
50     write(fd, "0", 2);
51     write(fd, "1", 2);
52 }
```

Figura 18 – Estabelecendo um ciclo infinito

Ao estabelecer a tarefa como operante indefinidamente, é possível remover o acréscimo feito à variável *i* em cada repetição do *why*. O resultado é exibido na figura seguinte.

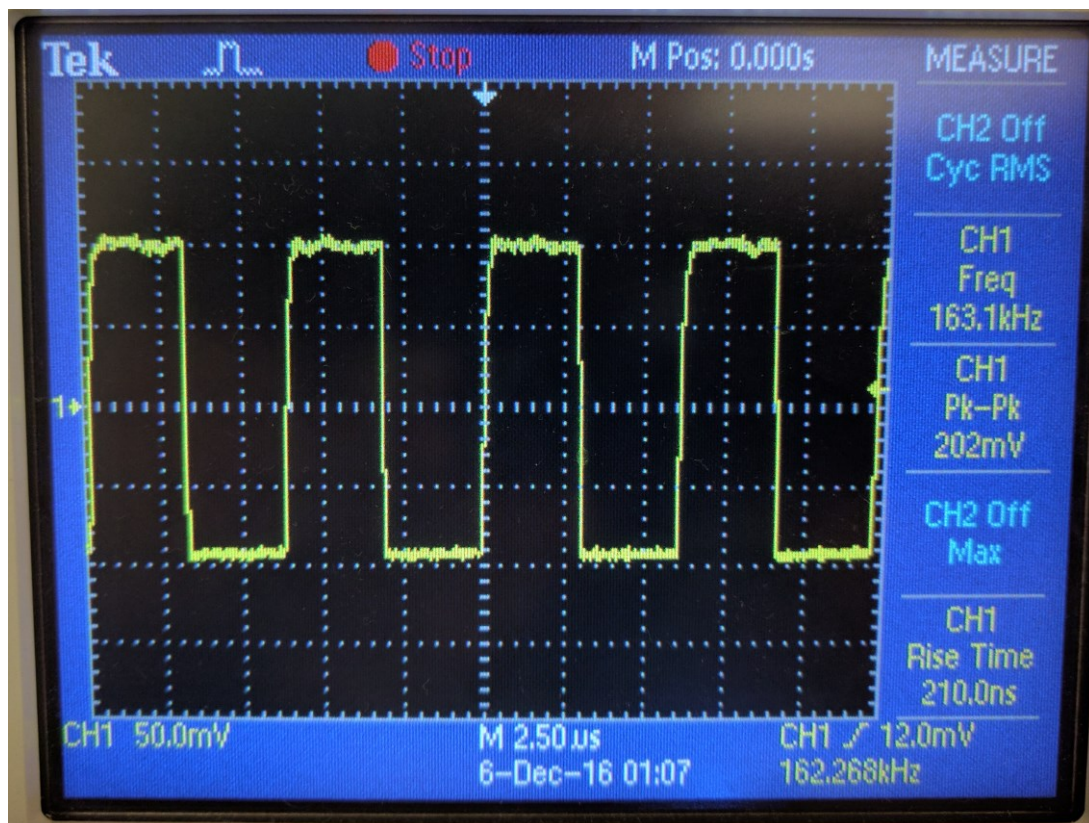


Figura 19 – Resultado de um ciclo infinito

A frequência de operação foi de 160.4kHz para 162.3kHz, o que representa uma melhoria de quase 1,4%. A diferença aparenta ser insignificante, a ponto de provar que os ciclos finito e infinito tem um número muito similar de instruções. No entanto, um avanço como este para um código que parecia não ter mais caminhos para a otimização é, na verdade, relevante, uma vez que programas mais longos e complexos certamente receberiam um impacto maior de pequenas alterações como esta, caso acumuladas.

Por fim, outro fator que deve ser ressaltado é a estabilidade do sistema. O *jitter* é um efeito colateral existente e decorrente do uso do Linux, que pode ser medido mesmo quando for, aparentemente, irrelevante. No entanto, na imagem obtida pelo osciloscópio e na frequência de amostragem na tela do mesmo, não há grandes variações ou impurezas resultadas por ondas secundárias.

2.3.4 Metal Puro

Como dito anteriormente ao longo da introdução, ao justificar os motivos deste trabalho, várias versões do Linux são, por padrão, sistemas operacionais não-preemptivos [2]. Isto significa que ele tem certas dificuldades para realizar operações em tempo real em frequências elevadas, por exemplo as obtidas nos testes anteriores, como alterar o estado de um GPIO em alta velocidade. No entanto, a placa ainda tem recursos, em termos de processamento, superiores, sendo limitada apenas pelo sistema operacional.

Todavia, há uma maneira de utilizar a BBB como um microcontrolador poderoso e ainda cheio de recursos, ao invés de se prender a um computador completo gerenciado pelo Linux. Este processo é chamado de programação, ou ambiente, em metal puro (BM), o que significa que o sistema ou projeto irá atuar diretamente no hardware.

A programação em BM permite o desenvolvimento e operações de sistemas em tempo real, tempos de resposta extremamente curtos e de alta performance, e são geralmente utilizados para otimizar programas e aplicações.

O que torna este processo possível é o emulador JTAG. O JTAG implementa padrões de instrumentação no próprio chip em projetos de circuitos integrados (*electronic design automation*, ou EDA) como uma ferramenta digital complementar de simulação. Ele especifica o uso dedicado de uma porta de compilação implementando uma interface de comunicações seriais para acesso de baixa sobrecarga, sem a necessidade de acesso externo direto para o endereço do sistema e barramentos de dados.

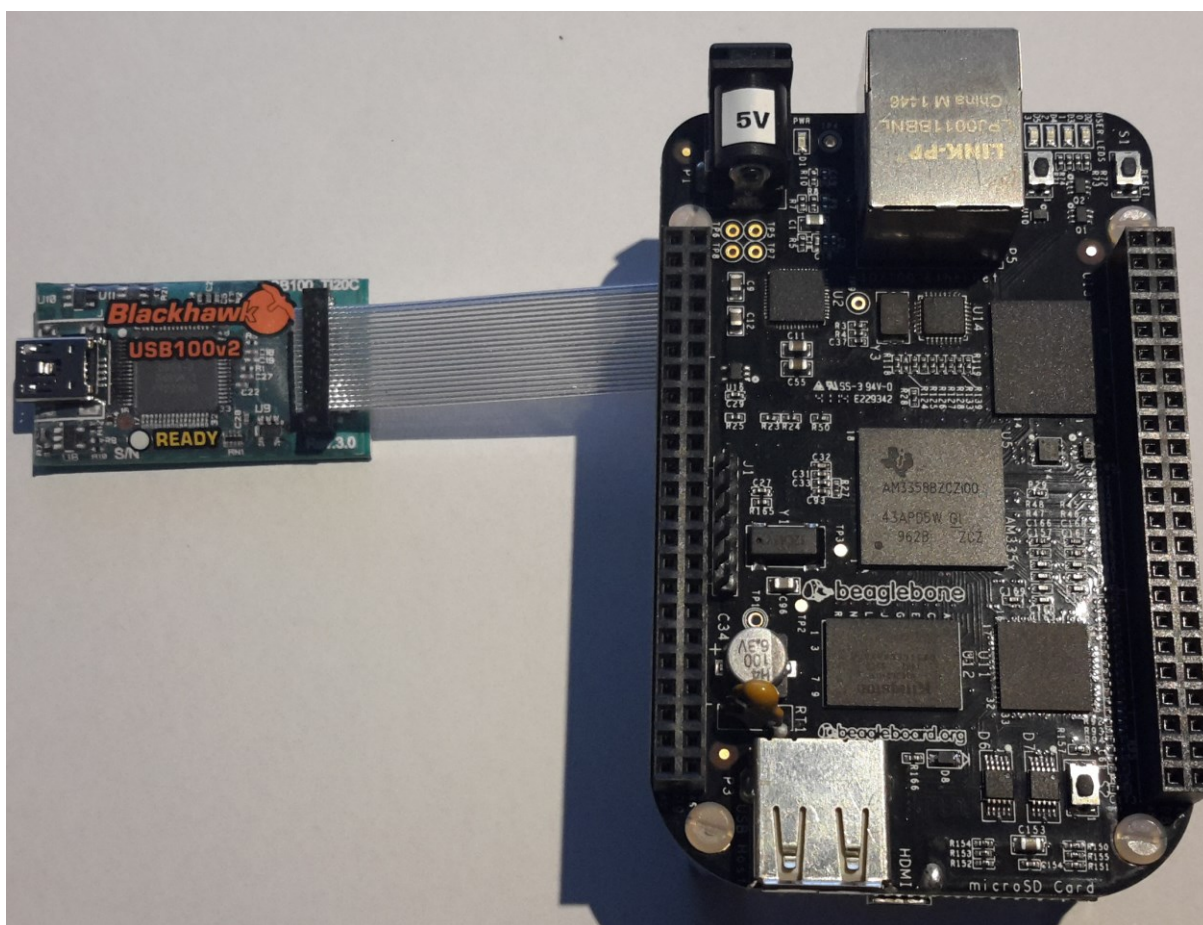


Figura 20 – A BeagleBone Black conectada a um cabo JTAG [5]

O manual de como implementar o cabo JTAG e instalar e configurar corretamente o Code Composer Studio v6 está presente, passo-a-passo, no Anexo III.

Agora, afim de fazer o teste de velocidade o exemplo contido no CCS “gpioLEDBlink” será importado e utilizado como base. Este processo de importação de exemplo poupa o trabalho de abrir as bibliotecas uma a uma e incluí-las no projeto, então sempre se economiza tempo ao fazê-lo. Porém, mesmo as bibliotecas estando disponíveis, ainda é preciso importar e compilar outras ferramentas.

Para importar o exemplo, é necessário repetir os últimos passos do Anexo III, para que o usuário se depare novamente com a janela de “*Discovered Projects*” (ou Projetos Encontrados), clicando em qualquer espaço vazio da aba “Project Explorer”, então em “Import”, “CCS Projects” e procurar pela pasta do SW novamente. No entanto, desta vez devem ser selecionadas as pastas “drivers”, “platform”, “system” e “utils” para importação. O “Project Explorer” deve estar como mostra a Figura 21.

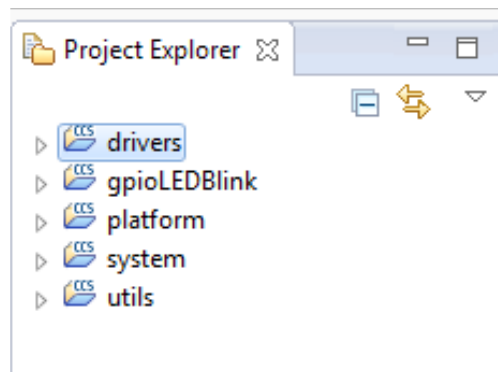


Figura 21 – Aba do projeto com todas as ferramentas

Para compilar cada uma das ferramentas, é preciso selecionar cada uma delas e clicar no ícone com um martelo logo abaixo das opções do menu principal. Ou então, clicar com o botão direito em cada uma delas e selecionar a opção “Build Project”. Mas antes disto é importante certificar-se de que elas estão configuradas da forma correta. Para verificar, basta clicar em cada uma com o botão direito e selecionar a opção “Properties”, como exemplificado pelas figuras 22 a 25.

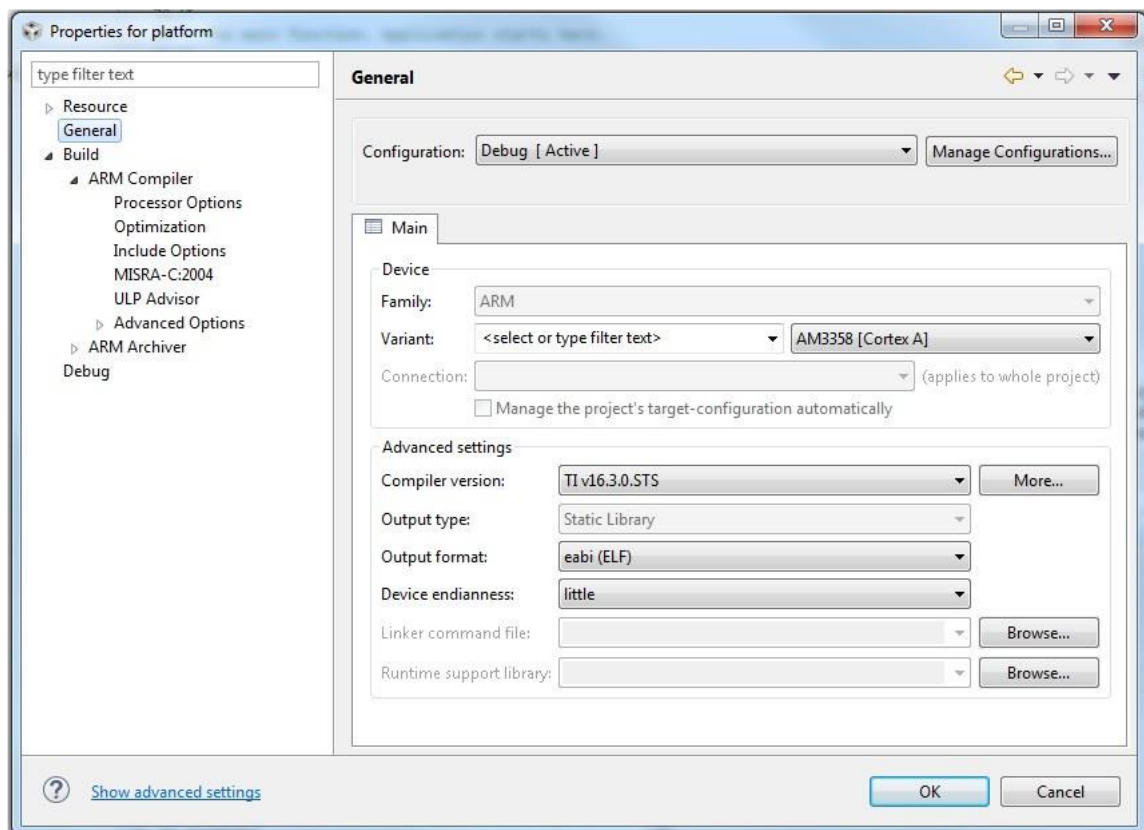


Figura 22 – Configurações de compilação das ferramentas, parte 1

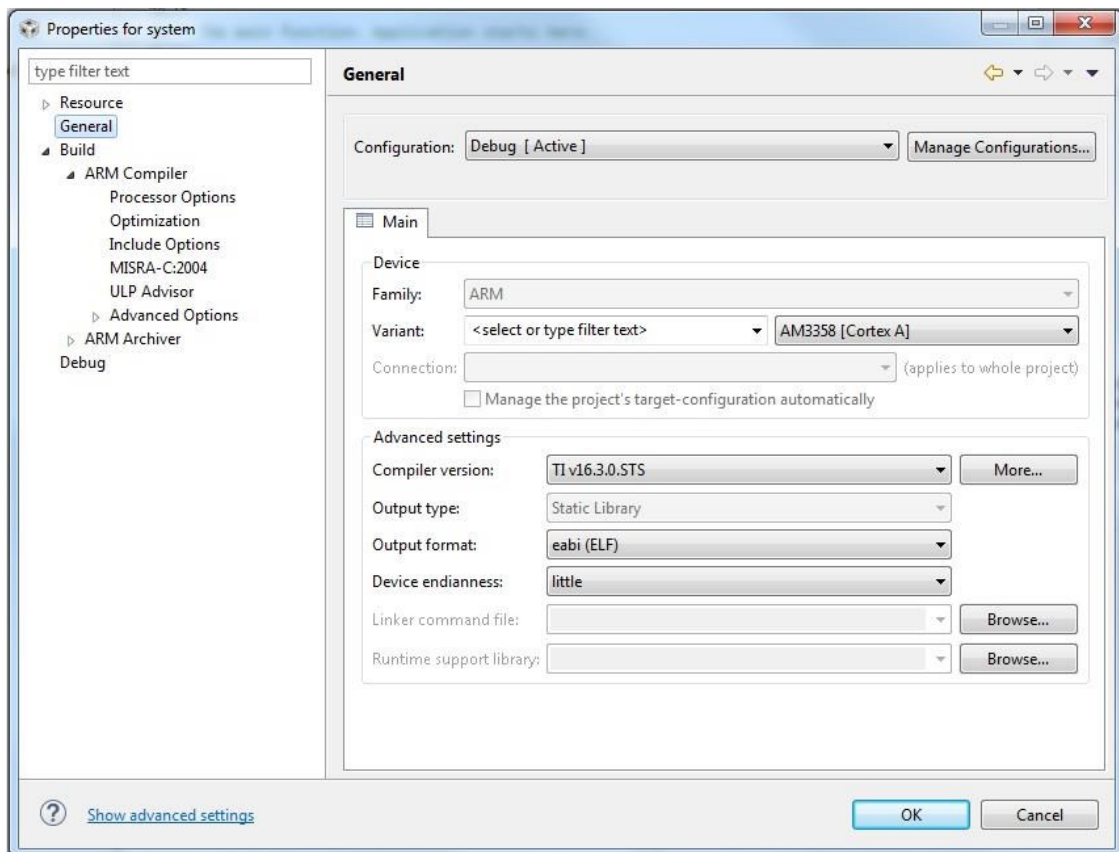


Figura 23 – Configurações de compilação das ferramentas, parte 2

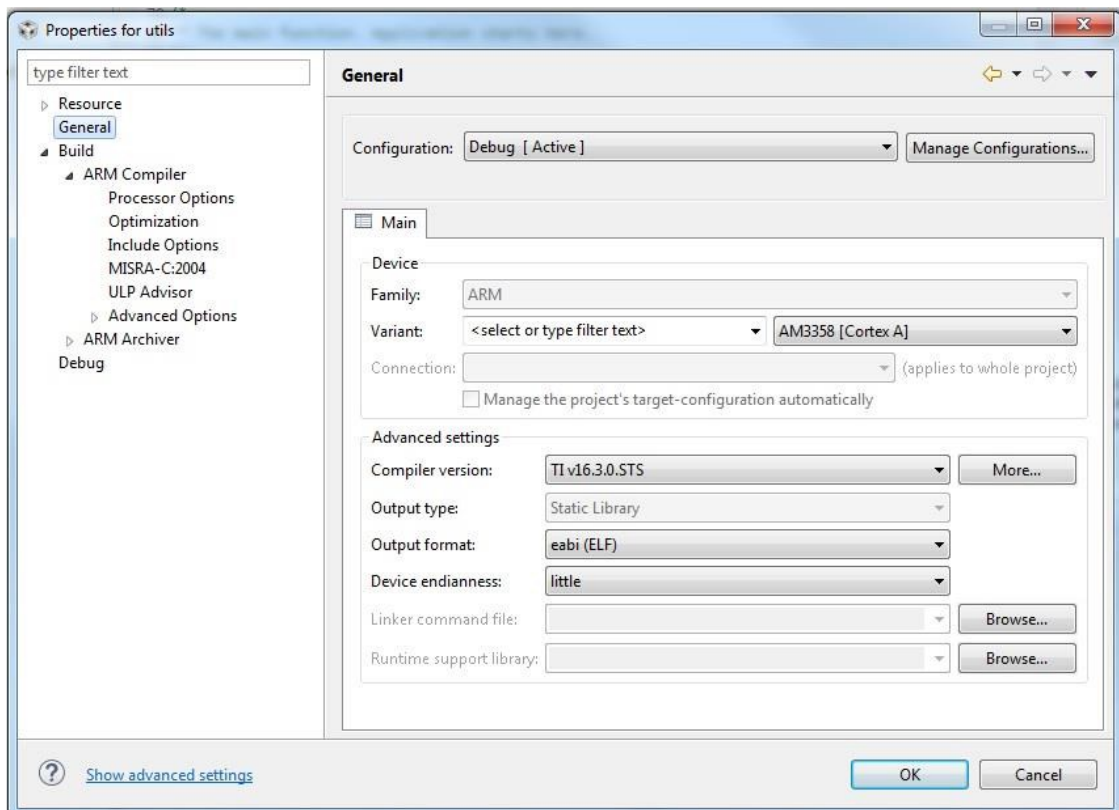


Figura 24 – Configurações de compilação das ferramentas, parte 3

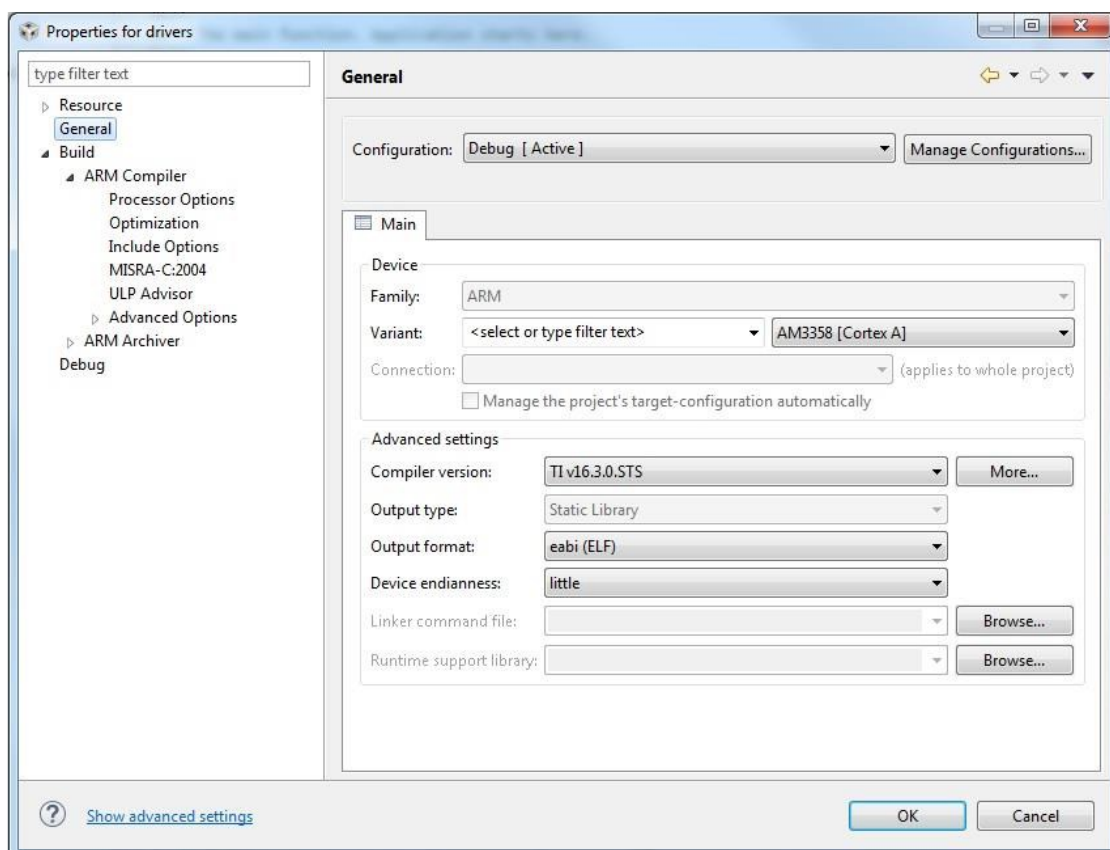


Figura 25 – Configurações de compilação das ferramentas, parte 4

As ferramentas devem ser compiladas sem quaisquer problemas. No entanto, caso algum ocorra, é válido tentar mudar a opção “Compiler version” de TI v16.3.0.STS para TI v5.2.5, e então tentar compilar novamente. Caso ainda não esteja funcionando, é necessário expandir a pasta de bibliotecas e procurar pelas pastas “Debug” e “Release”, como na Figura 26. Se elas estiverem vazias, o usuário pode tentar apaga-las e tentar compilar as ferramentas novamente. Na improvável ocorrência de erros ainda persistirem, é possível que o usuário não esteja utilizando a versão 6.1.2 do CCS, e há problemas com relação a compatibilidade.

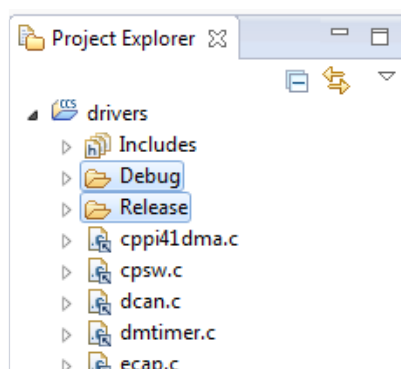


Figura 26 – Pastas Debug e Release em destaque

Assim que as ferramentas forem compiladas corretamente compiladas, o projeto em si será manipulado. Primeiramente, suas configurações (botão direito em “gpioLEDBlink”, e então “Properties”). As configurações devem ser como mostra a Figura 27.

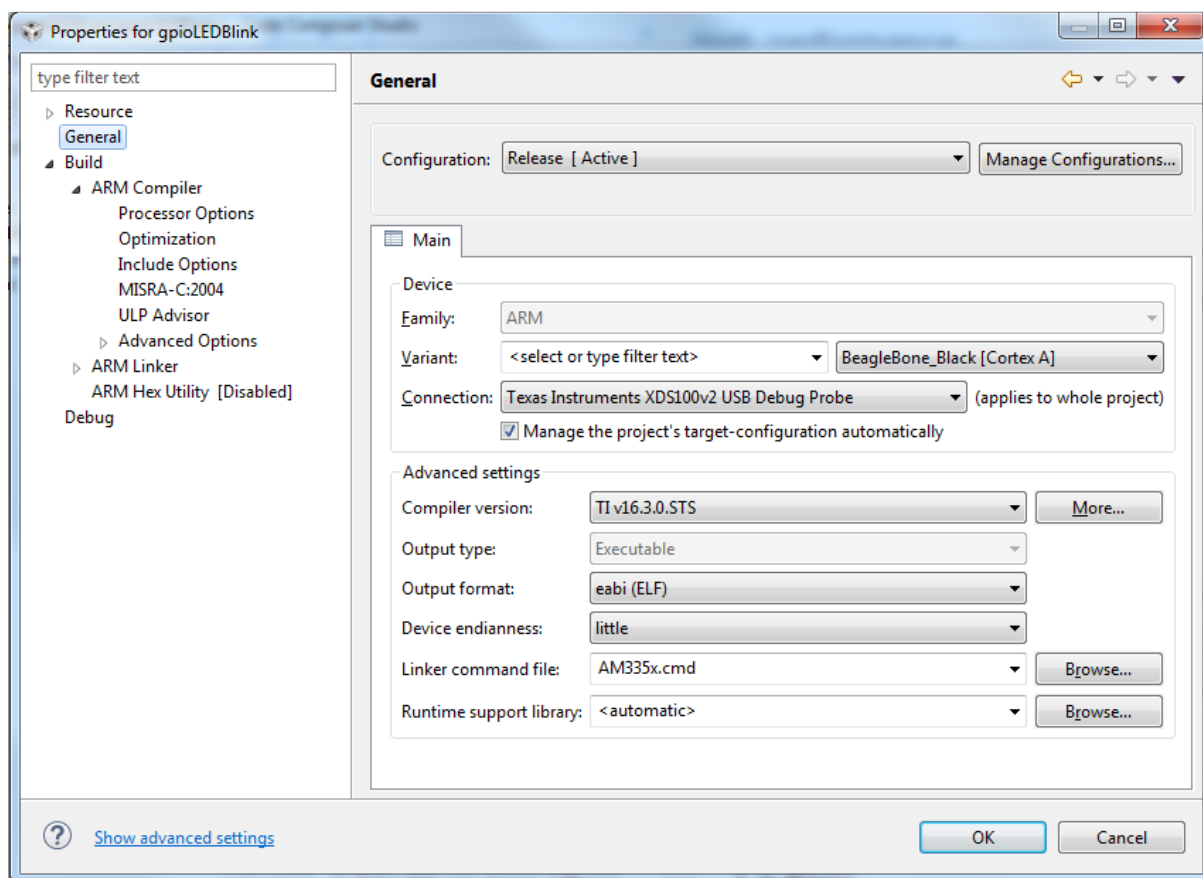


Figura 27 – Configurações do projeto

Ainda nesta janela, pode-se conferir se as ferramentas estão corretamente embutidas no projeto. Na aba da esquerda, na opção “File Search Path”, “drivers”, “system”, “platform” e “utils” devem estar listados como na Figura 28.

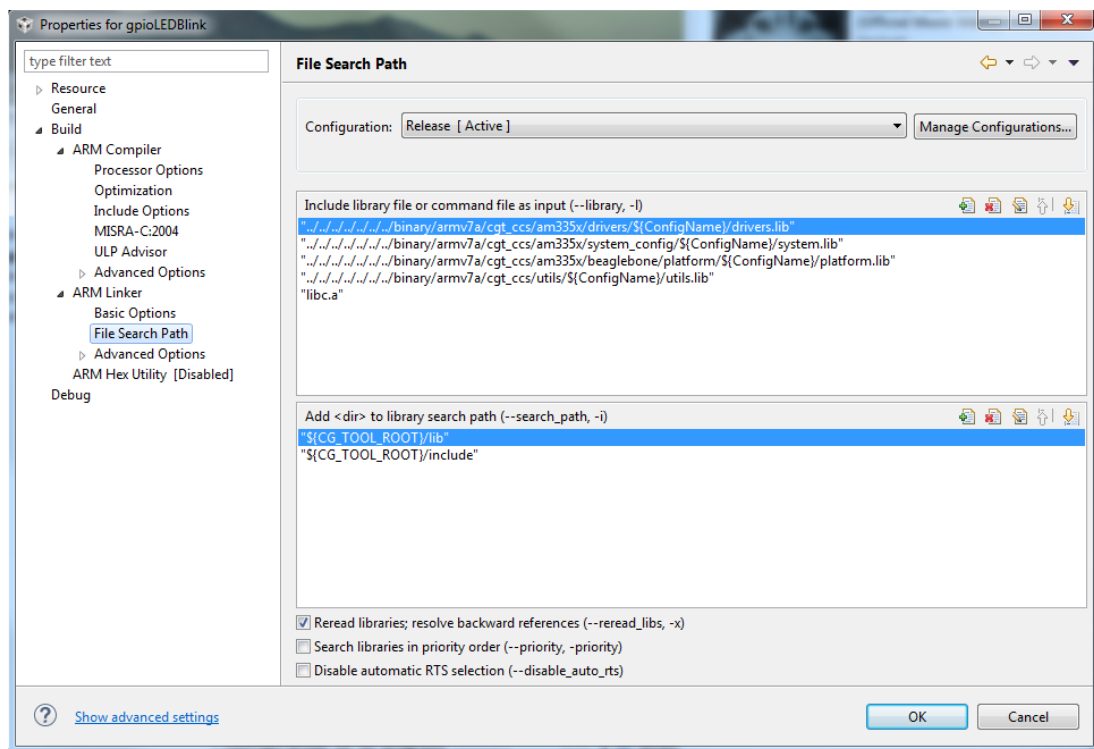


Figura 28 – Arquivos incluídos no sistema

Caso não estejam (na verdade, até mesmo caso estejam), é recomendado adicioná-los como dependências. Na aba “Build”, em “Dependencies”, o usuário deve clicar em “Add...”, selecionar as ferramentas e então aplicar as mudanças.

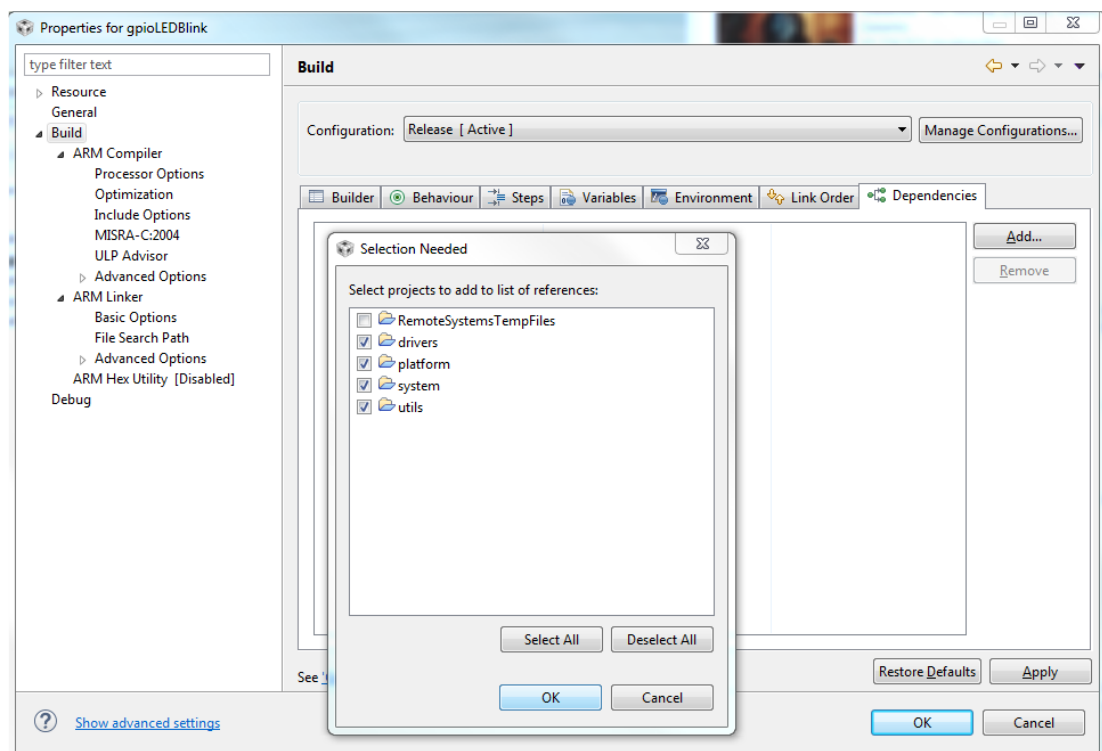


Figura 29 – Adicionando ferramentas como dependências

Finalmente, agora é possível alterar o código do exemplo para que o mesmo se encaixe no objetivo deste documento. Para tal, é necessário explorar as bibliotecas relacionadas aos GPIOs. A análise e o estudo cauteloso destas bibliotecas, na verdade, permite que o desenvolvedor amplie suas possibilidades de aplicações e se equipe melhor com os recursos presentes no CCS, descobrindo novas funções e como as mesmas funcionam.

Para o caso do uso dos GPIOs, a maior parte dos comandos necessários para a programação de um projeto está presente na biblioteca “gpio_v2.c”, encontrada ao expandir a pasta “drivers”. Para abri-la, basta clicar duas vezes com o botão esquerdo do mouse sobre ela.

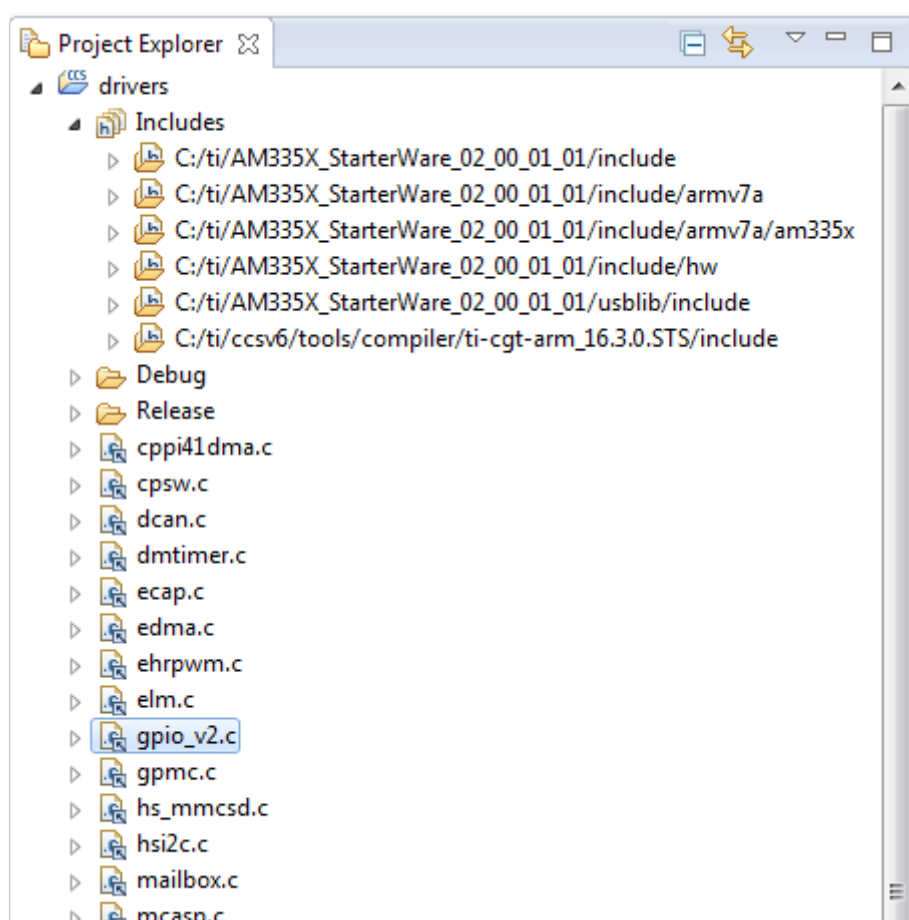


Figura 30 – Biblioteca “gpio_v2.c”

Aqui podem ser encontradas informações extremamente relevantes, com boa parte dos comandos comentados com uma breve explicação em inglês, o que ajuda no entendimento de como cada função trabalha. Seguem alguns exemplos.

```

122 void GPIODirModeSet(unsigned int baseAdd,
123                     unsigned int pinNumber,
124                     unsigned int pinDirection)
125 {
126     /* Checking if pin is required to be an output pin. */
127     if(GPIO_DIR_OUTPUT == pinDirection)
128     {
129         HWREG(baseAdd + GPIO_OE) &= ~(1 << pinNumber);
130     }
131     else
132     {
133         HWREG(baseAdd + GPIO_OE) |= (1 << pinNumber);
134     }
135 }

```

Figura 31 – Declarando um pino como entrada ou saída

```

166 /**
167  * \brief This API drives an output GPIO pin to a logic HIGH or a logic LOW
168  *        state.
169  *
170  * \param baseAdd    The memory address of the GPIO instance being used
171  * \param pinNumber   The number of the pin in the GPIO instance
172  * \param pinValue    This specifies whether a logic HIGH or a logic LOW
173  *                   should be driven on the output pin
174  *
175  * 'pinNumber' can take one of the following values:
176  * (0 <= pinNumber <= 31)\n
177  *
178  * 'pinValue' can take one of the following values:
179  * - GPIO_PIN_LOW - indicating to drive a logic LOW(logic 0) on the pin.
180  * - GPIO_PIN_HIGH - indicating to drive a logic HIGH(logic 1) on the pin.
181  *
182  * \return None
183  *
184  */
185
186 void GPIOPinWrite(unsigned int baseAdd,
187                  unsigned int pinNumber,
188                  unsigned int pinValue)
189 {
190     if(GPIO_PIN_HIGH == pinValue)
191     {
192         HWREG(baseAdd + GPIO_SETDATAOUT) = (1 << pinNumber);
193     }
194     else
195     {
196         HWREG(baseAdd + GPIO_CLEARDATAOUT) = (1 << pinNumber);
197     }
198 }

```

Figura 32 – Definindo valor lógico de uma saída

```

200 /**
201  * \brief   This API determines the logic level(value) on a specified
202  *          GPIO pin.
203  *
204  * \param   baseAdd    The memory address of the GPIO instance being used
205  * \param   pinNumber   The number of the pin in the GPIO instance
206  *
207  * 'pinNumber' can take one of the following values:
208  * (0 <= pinNumber <= 31)\n
209  *
210  * \return  The pin value of the specified pin of the GPIO instance.
211  */
212
213 unsigned int GPIOPinRead(unsigned int baseAdd,
214                          unsigned int pinNumber)
215 {
216     return(HWREG(baseAdd + GPIO_DATAIN) & (1 << pinNumber));
217 }

```

Figura 33 – Lendo o valor de uma entrada

Outro detalhe acerca do código de exemplo, é que ele manipula um dos LEDs embutidos em bordo, ou seja, é impossível analisar sua forma de onda através de um osciloscópio graças à maneira como é feita sua montagem física. Logo, é preciso alterar a saída para um dos pinos, para assim então conectá-la a um resistor, ao qual pode ser conectada uma ponteira de prova para leitura de dados.

Porém, os pinos não são traduzidos de forma fácil pela biblioteca. Por exemplo, não é possível utilizar o pino P9_12, que é um GPIO, simplesmente declarando “P9_12” (como feito em Python). Para descobrir seu endereço no registrador, é necessário consultar a folha de dados da BeagleBone Black.

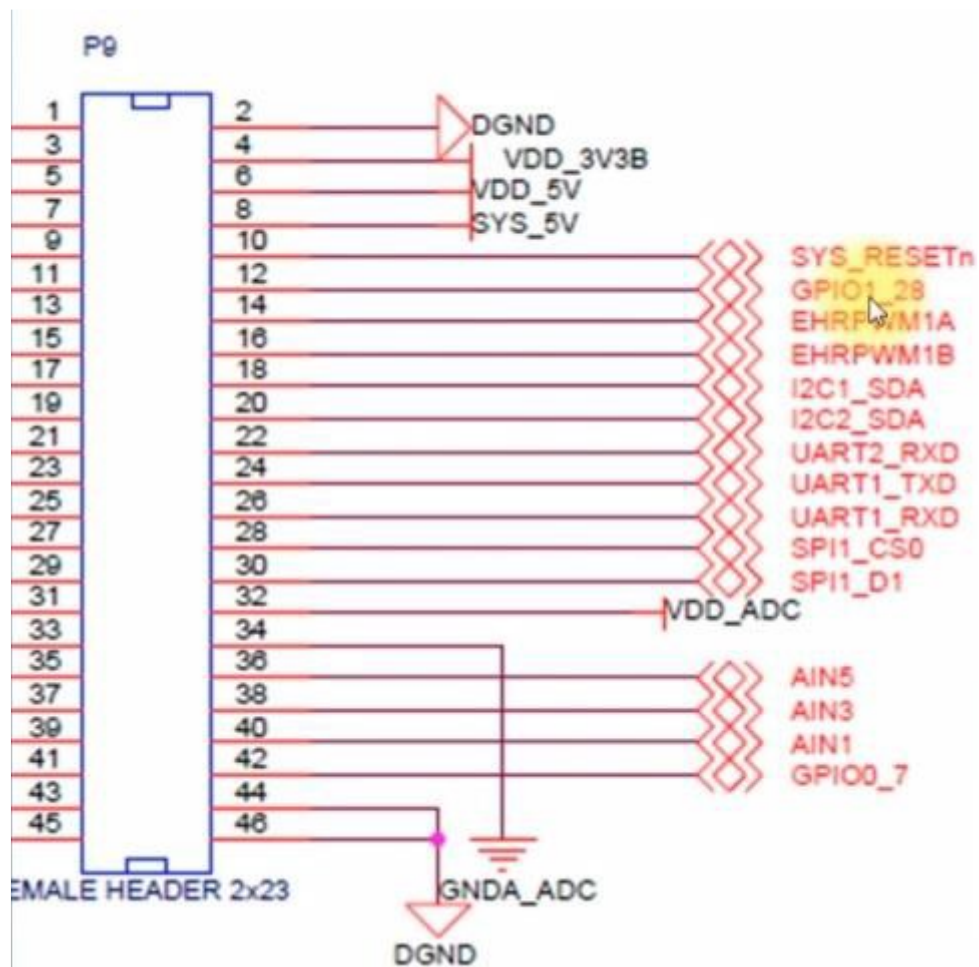


Figura 34 – Pinos do cabeçalho P9

Como o endereço deste pino é GPIO1_28, para ativá-lo deve ser declarado primeiro o endereço do registrado, e então o número do pino.

```

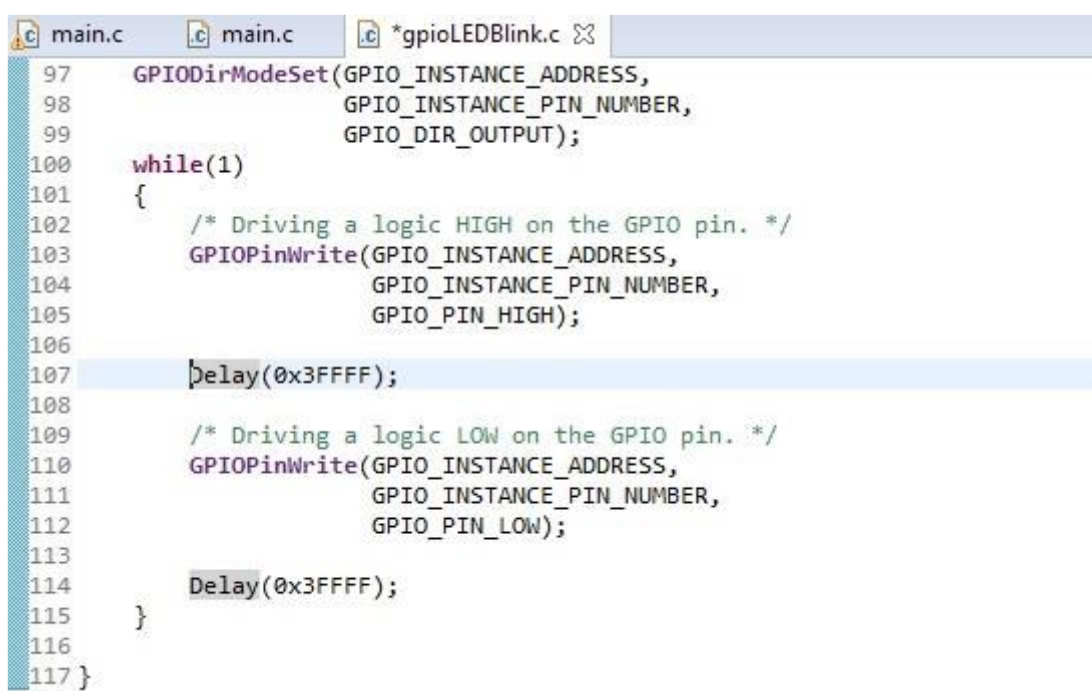
67 ****
68 #define GPIO_INSTANCE_ADDRESS      (SOC_GPIO_1_REGS)
69 #define GPIO_INSTANCE_PIN_NUMBER  [(28)]
70 ****

```

Figura 35 – Declarando endereços

Agora, para saber o quão rápido este pino pode ser engatilhado (definido níveis lógicos distintos sequencialmente), basta criar um laço de repetição, seja infinita ou finita, e remover todos os atrasos do código, presentes nas linhas 107 e 114 da Figura 36. Os atrasos podem também ser definidos para “0”, caso, por algum motivo, o usuário queira preservá-los no código. Porém, mesmo com os atrasos sendo definidos para 0, ou um número decimal muito pequeno, o tempo que o compilador leva para acessar a função “Delay” e retornar ao resto do programa pode

influenciar o tempo de resposta o suficiente para causar um impacto negativo em algum sistema em tempo real crítico. Logo, para o teste, estas funções de Delay serão simplesmente removidas.



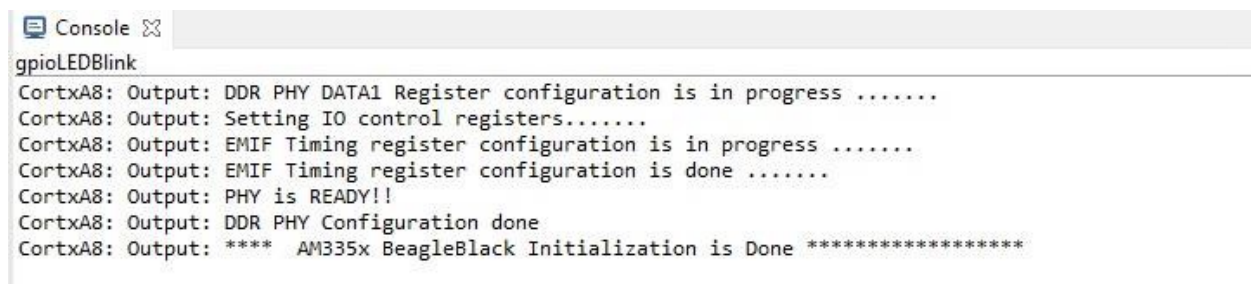
```
97  GPIODirModeSet(GPIO_INSTANCE_ADDRESS,  
98                  GPIO_INSTANCE_PIN_NUMBER,  
99                  GPIO_DIR_OUTPUT);  
100 while(1)  
101 {  
102     /* Driving a logic HIGH on the GPIO pin. */  
103     GPIOPinWrite(GPIO_INSTANCE_ADDRESS,  
104                 GPIO_INSTANCE_PIN_NUMBER,  
105                 GPIO_PIN_HIGH);  
106  
107     Delay(0x3FFFF);  
108  
109     /* Driving a logic LOW on the GPIO pin. */  
110     GPIOPinWrite(GPIO_INSTANCE_ADDRESS,  
111                 GPIO_INSTANCE_PIN_NUMBER,  
112                 GPIO_PIN_LOW);  
113  
114     Delay(0x3FFFF);  
115 }  
116  
117 }
```

Figura 36 – Código com atrasos

Agora, para depurar, compilar e executar o código, o JTAG deve estar conectado e a BBB sido ligada enquanto o botão de inicialização estivesse pressionado. A fim de iniciar a depuração, basta clicar no ícone verde de inseto no topo da interface, sob o menu principal, ou clicar com o botão direito no projeto e selecionar a opção “Debug as...” e então “Code Composer Debug Session”.

A primeira vez que o CCS realizar esta tarefa pode demorar diversos minutos. No entanto, é extremamente improvável que erros ocorram nesta etapa caso o JTAG tenha sido configurado corretamente.

Depois de obter sucesso na depuração, o console deve se parecer como mostra a Figura 37, e o programa estará pronto para ser executado.



```

Console
gpioLEDBlink
CortexA8: Output: DDR PHY DATA1 Register configuration is in progress .....
CortexA8: Output: Setting IO control registers.....
CortexA8: Output: EMIF Timing register configuration is in progress .....
CortexA8: Output: EMIF Timing register configuration is done .....
CortexA8: Output: PHY is READY!!
CortexA8: Output: DDR PHY Configuration done
CortexA8: Output: **** AM335x BeagleBlack Initialization is Done ****

```

Figura 37 – Console após depuração do código

Para executar o programa, basta clicar na seta verde sob o menu principal, ou utilizar a tecla de atalho F8.

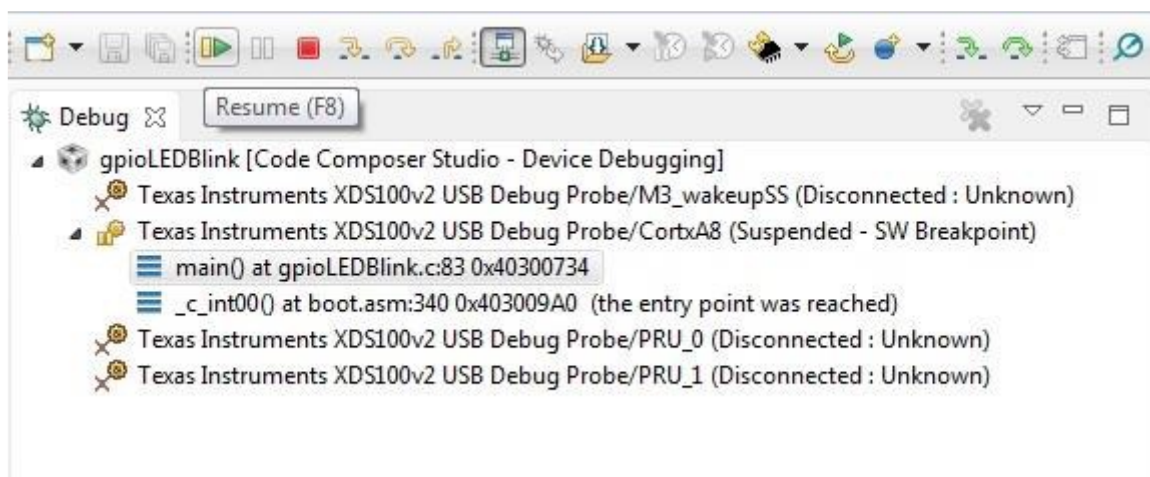


Figura 38 – Executar código

O programa irá ser executado, e o pino estará alternando em sua frequência máxima.

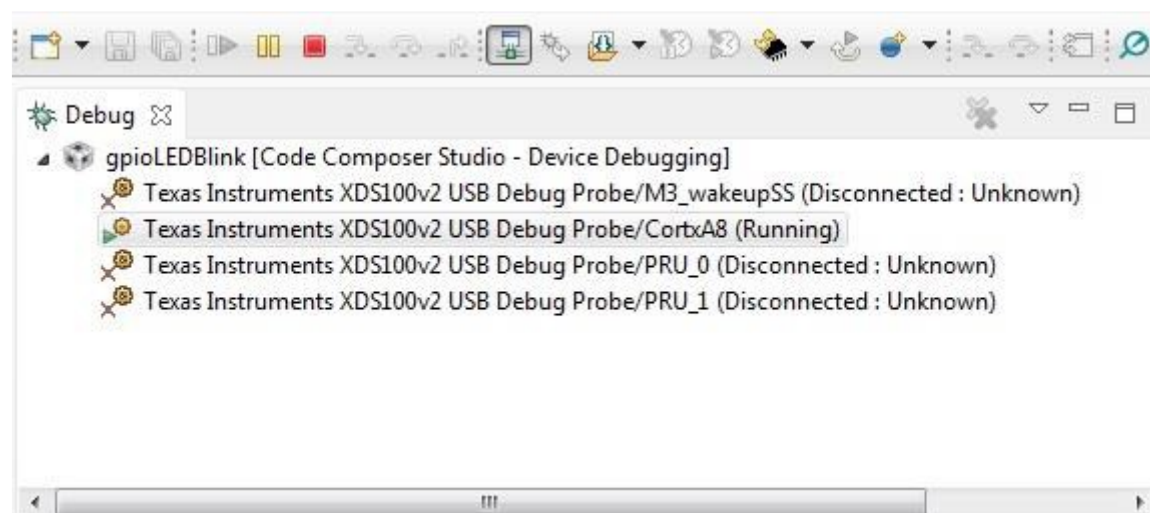


Figura 39 – Programa em execução

O resultado da forma de onda obtida no osciloscópio é demonstrado na Figura 40.

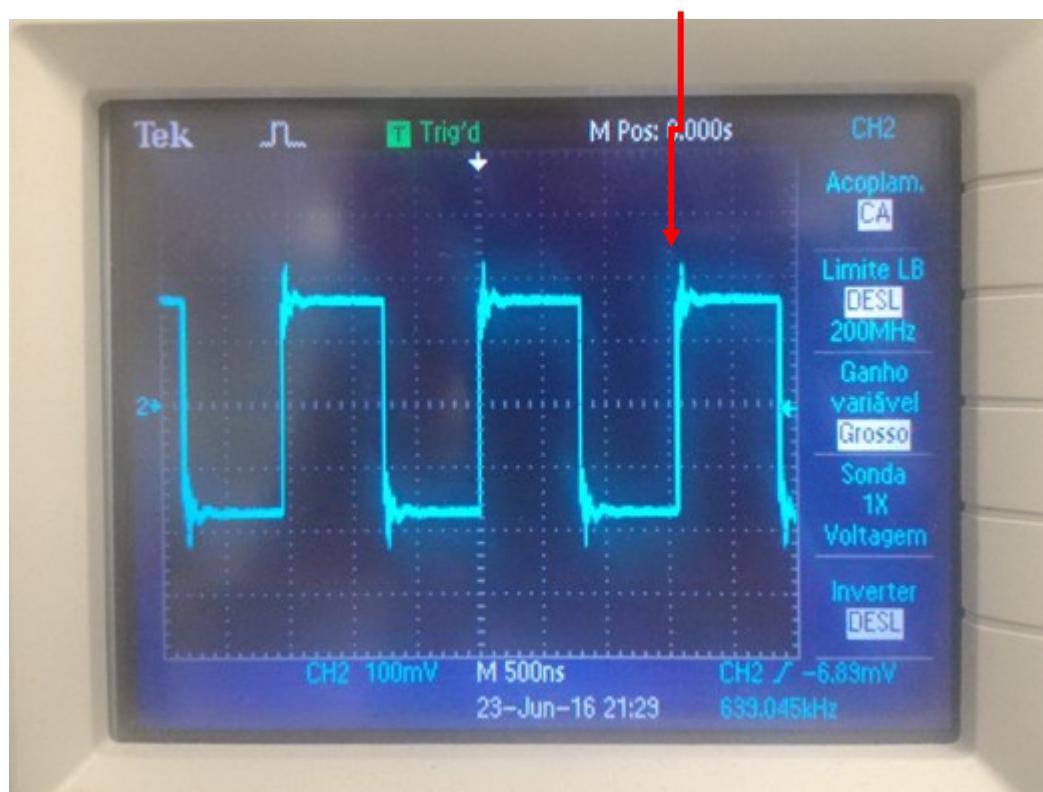


Figura 40 – Resultado do teste de velocidade da BBB em metal puro pelo CCS

Agora que o sistema operacional está fora do caminho e não mais representa uma barreira, foi possível alcançar a velocidade de 639kHz, 74 vezes maior que o teste programado em C utilizando a biblioteca específica e aproximadamente 4 vezes maior que o teste programado em C padrão, que são as duas formas mais difundidas de uso da placa. Além disso, o sistema apresenta grande estabilidade (uma vez que não há espectros secundários da forma de onda, ou ausência de *jitter*), duas características positivas que não foram obtidas nos testes regulares do Linux. O pico apontado pela seta vermelha na Figura 40 é indesejável, no entanto esperado para frequências tão altas como esta. Vale citar também que o programa foi capaz de alternar de um nível lógico para outro em apenas 900 nano segundos, sendo adequado para aplicações em tempo real que exigem este tempo de resposta.

Além destas, há outra vantagem que pode ser ressaltada sobre esta abordagem. O CCS é um programa de desenvolvimento da TI, mesma fabricante da BeagleBone Black e, em caso de problemas, a empresa fornece suporte para o desenvolvedor tanto em software quanto em hardware simultaneamente.

Em contrapartida, este processo possui certas desvantagens que devem ser mencionadas. A primeira dela é o uso essencial do JTAG. O conector não apenas é uma peça extra de hardware que deve ser adquirida à parte, como também representa um custo adicional e é trabalhoso de se usar. É possível, inclusive, estragar certos componentes da placa se uma solda descuidada for feita.

Ademais, desfazer-se do uso do Linux responde com uma grande velocidade, mas também descarta as vantagens que o sistema operacional traz, com relação a pacotes de extensão e ferramentas, que foram citadas anteriormente.

2.3.5 Memória do dispositivo

Na BeagleBone Black, o sistema operacional Linux extrai os pinos de entrada e saída como arquivos. Afim de controlar um determinado pino ou periférico da placa, o usuário pode programar comandos de acesso a estes respectivos arquivos. Por exemplo, para desligar um dos LEDs azuis, posicionados em alguns dos cantos do hardware, que ligam automaticamente ao conectar a BBB a uma fonte de energia, é possível navegar, através de linhas de comandos no prompt do sistema operacional, para `/sys/class/leds/beaglebone:green:usr0`. O arquivo de disparo presente neste diretório controla o comportamento do LED que, por padrão, está programado para efetuar piscadas de “batimento cardíaco”.

Para que esta opção seja alterada, pode-se utilizar o comando `echo none > trigger`, para desligar o LED, e `echo default-on > trigger`, para ligá-lo. O processo de programação, seja em C, ou Python, como analisados e testados em 2.3.1 e 2.3.2, por exemplo, é análogo. O código substitui a necessidade de ligar e desligar os pinos de forma manual, digitando os comandos, um por um, toda vez que for necessário manipular os pinos, fazendo isso de forma automática e sem atrasos.

Como resultado desta abstração dos pinos em arquivos, é possível observar a praticidade e facilidade em programar e desenvolver projetos simples. No entanto, quanto mais abstrações houverem, tornando o processo mais intuitivo para o programador, mais lenta será a resposta da placa para executar os comandos, e mais CPU será utilizada na interpretação dos mesmos. Para aplicações em geral, este fato não costuma ser caracterizado como uma desvantagem. Porém, é possível que um projeto em especial necessite de um tempo de resposta muito maior do que os que foram obtidos de demonstrados neste estudo, aplicando os métodos tradicionais e mais práticos de programação.

Afim de atingir maiores velocidades, é possível criar uma espécie de atalho entre comando e execução, viabilizando que o código trabalhe mais próximo do hardware. Para isso, pode-se utilizar o mapeamento de memória para acessar os registros de entrada e saída diretamente. Tendo esta finalidade em mente, é necessário declarar, ou definir, dentro do código os endereços de memória dos registros que controlam os pinos de entrada e saída que serão utilizados.

Este tipo de informação, assim como demais dados que ainda serão úteis para esta aplicação, estão presentes no documento “*technical reference manual of the AM335x-processor*”, ou, em português, manual de referência técnica do processador AM335x, hardware utilizado na BeagleBone Black. O manual é um documento disponibilizado pela Texas Instruments de aproximadamente 4100 páginas.

Primeiramente, é necessário ir para o Capítulo 2, “*Memory Map*”, onde será possível encontrar uma extensa lista contendo todos os endereços de memória de vários registros que, em geral, ditam o comportamento do processador. Seguindo adiante algumas páginas dentro deste capítulo, sob a aba “*Region Name*”, estará a primeira porta que pode ser usada como entrada e saída, como exemplificado na figura abaixo:



ARM Cortex-A8 Memory Map

www.ti.com

Table 2-2. L4_WKUP Peripheral Memory Map (continued)

Region Name	Start Address (hex)	End Address (hex)	Size	Description
GPIO1	0x4804_C000	0x4804_CFFF	4KB	GPIO1 Registers
	0x4804_D000	0x4804_DFFF	4KB	Reserved

Figura 41 – Endereço de memória de um pino de entrada/saída [4]

Além da lista ditar a função do pino, que no caso é um GPIO, ela fornece outros dados relevantes que podem ser utilizados no código para declarar ou definir as portas que serão acionadas, como os endereços de início e de fim e a extensão, em KB, que o pino ocupa no mapa periférico de memória.

Agora que foi visto como obter as informações necessárias para declarar as portas GPIO de interesse, o próximo passo é, basicamente, descobrir os comandos que podem ser passados a elas. Para o experimento em questão, acerca do tempo de resposta, são essenciais: habilitar a porta como saída, definir nível lógico alto e definir nível lógico baixo. Tais operações estão

igualmente disponíveis no manual de referência técnica do processador, no capítulo 25: “General-Purpose Input/Output”.

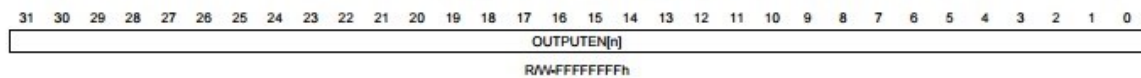


25.4.1.16 GPIO_OE Register (offset = 134h) [reset = FFFFFFFFh]

GPIO_OE is shown in Figure 25-22 and described in Table 25-21.

The GPIO_OE register is used to enable the pins output capabilities. At reset, all the GPIO related pins are configured as input and output capabilities are disabled. This register is not used within the module, its only function is to carry the pads configuration. When the application is using a pin as an output and does not want interrupt generation from this pin, the application can/has to properly configure the Interrupt Enable registers.

Figure 25-22. GPIO_OE Register



LEGEND: R/W = Read/Write; R = Read only; W1toC1 = Write 1 to clear bit; -n = value after reset

Table 25-21. GPIO_OE Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	OUTPUTEN[n]	R/W	FFFFFFFh	Output Data Enable 0x0 = The corresponding GPIO port is configured as an output. 0x1 = The corresponding GPIO port is configured as an input.

Figura 42 – Comando que determina pino como saída [4]

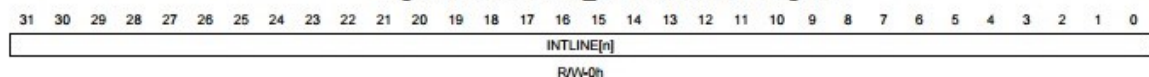
O comando “GPIO_OE” controla a capacidade de um pino em se tornar entrada ou saída. No modo “reset”, todos os pinos GPIO tem esta capacidade desabilitada. Para manipular o pino, basta seguir as instruções da aba “Description” da imagem anterior: o bit 0 irá defini-lo como saída, enquanto o bit 1 irá defini-lo como entrada. A partir deste ponto, novos comandos se tornam disponíveis para a porta declarada.

25.4.1.26 GPIO_SETDATAOUT Register (offset = 194h) [reset = 0h]

GPIO_SETDATAOUT is shown in Figure 25-32 and described in Table 25-31.

Writing a 1 to a bit in the GPIO_SETDATAOUT register sets to 1 the corresponding bit in the GPIO_DATAOUT register; writing a 0 has no effect. A read of the GPIO_SETDATAOUT register returns the value of the data output register (GPIO_DATAOUT).

Figure 25-32. GPIO_SETDATAOUT Register



LEGEND: R/W = Read/Write; R = Read only; W1toCl = Write 1 to clear bit; -n = value after reset

Table 25-31. GPIO_SETDATAOUT Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	INTLINE[n]	R/W	0h	Set Data Output Register 0x0 = No effect 0x1 = Set the corresponding bit in the GPIO_DATAOUT register.

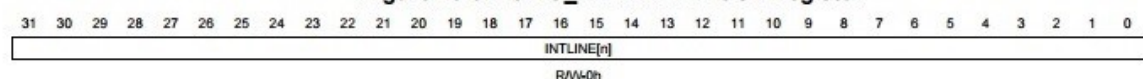
Figura 43 – Definindo nível lógico alto [4]

25.4.1.25 GPIO_CLEARDATAOUT Register (offset = 190h) [reset = 0h]

GPIO_CLEARDATAOUT is shown in Figure 25-31 and described in Table 25-30.

Writing a 1 to a bit in the GPIO_CLEARDATAOUT register clears to 0 the corresponding bit in the GPIO_DATAOUT register; writing a 0 has no effect. A read of the GPIO_CLEARDATAOUT register returns the value of the data output register (GPIO_DATAOUT).

Figure 25-31. GPIO_CLEARDATAOUT Register



LEGEND: R/W = Read/Write; R = Read only; W1toCl = Write 1 to clear bit; -n = value after reset

Table 25-30. GPIO_CLEARDATAOUT Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	INTLINE[n]	R/W	0h	Clear Data Output Register 0x0 = No effect 0x1 = Clear the corresponding bit in the GPIO_DATAOUT register.

Figura 44 – Definindo nível lógico baixo [4]

Dados podem ser escritos no registro de saída de forma síncrona com o relógio de interface. O registro pode ser acessado com operações de ler e escrever ou através do uso de um protocolo alternativo de ligar e desligar. Esta ferramenta permite que o programador defina nível lógico alto, ou baixo, de bits específicos do registro, com um único comando para cada operação. Os

comandos são “GPIO_SETDATAOUT” e “GPIO_CLEARDATAOUT” para ativar e desativar, respectivamente, a saída de dados.

Quando configurado como entrada (alterando o bit desejado em *GPIO_OE* para 1, ao invés de 0), o estado da entrada pode ser lido através do bit correspondente no registro “*GPIO_DATAIN*”. Os dados de entrada também são colhidos de forma síncrona com o relógio de interface e, então, captados no registro depois de dois ciclos do relógio (os ciclos necessários para sincronizar e escrever dados).

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/mman.h>
4  #include <sys/stat.h>
5  #include <fcntl.h>
6  #include "beaglebone_gpio.h"
7
8  int main(int argc, char *argv[]) {
9      volatile void *gpio_addr = NULL;
10     volatile unsigned int *gpio_oe_addr = NULL;
11     volatile unsigned int *gpio_setdataout_addr = NULL;
12     volatile unsigned int *gpio_cleardataout_addr = NULL;
13     unsigned int reg;
14
15     int fd = open("/dev/mem", O_RDWR);
16
17     printf("Mapping %X - %X (size: %X)\n", GPIO1_START_ADDR,
18     GPIO1_END_ADDR, GPIO1_SIZE);
19
20     gpio_addr = mmap(0, GPIO1_SIZE, PROT_READ | PROT_WRITE,
21     MAP_SHARED, fd, GPIO1_START_ADDR);
22
23     gpio_oe_addr = gpio_addr + GPIO_OE;
24     gpio_setdataout_addr = gpio_addr + GPIO_SETDATAOUT;
25     gpio_cleardataout_addr = gpio_addr + GPIO_CLEARDATAOUT;
26
27     if(gpio_addr == MAP_FAILED) {
28         printf("Unable to map GPIO\n");
29         exit(1);
30     }
31     printf("GPIO mapped to %p\n", gpio_addr);
32     printf("GPIO OE mapped to %p\n", gpio_oe_addr);
33     printf("GPIO SETDATAOUTADDR mapped to %p\n",
34     gpio_setdataout_addr);
35     printf("GPIO CLEARDATAOUT mapped to %p\n",
36     gpio_cleardataout_addr);
37
38     reg = *gpio_oe_addr;
39     printf("GPIO1 configuration: %X\n", reg);
40     reg = reg & (0xFFFFFFFF - PIN);
41     *gpio_oe_addr = reg;
42     printf("GPIO1 configuration: %X\n", reg);
43

```

Figura 45 – Código para testes

O código presente na figura anterior possui todas as declarações e definições necessárias para que seja possível o acesso do registro do pino P9_12 através da memória do dispositivo, fazendo uso das informações presentes no manual e ressaltadas neste documento. Além disto, o mesmo foi desenvolvido no Cloud9. As informações necessárias para que se tenha conhecimento de como acessar este terminal estão presentes no Anexo I.

Também foram incluídas mensagens de erro, caso haja algum tipo de falha ao mapear esta memória. As mesmas são necessárias, pois, para que seja possível ter acesso ao arquivo “/dev/mem”, é necessário desenvolver o projeto no modo “root”. Isso significa que o projetista precisa ter controle total de administrador sobre a placa e a capacidade de usar comandos como “sudo su”, o que, em alguns casos, pode não ser possível.

```
44     printf("Start toggling PIN \n");
45     while(1) {
46
47         *gpio_setdataout_addr= PIN;
48         *gpio_cleardataout_addr = PIN;
49     }
50
51     close(fd);
52     return 0;
53 }
```

Figura 46 – Comandos para teste de velocidade

Em seguida, foi instruído que o pino tivesse seu nível lógico alternado entre alto e baixo, sem funções de atraso ou outra operação que pudesse tomar tempo do processador, em um loop infinito.

A montagem do circuito é, de forma padrão, o pino alvo conectado a um resistor, e o circuito acoplado a um osciloscópio para que seja possível avaliar a forma de onda obtida através do método em questão.

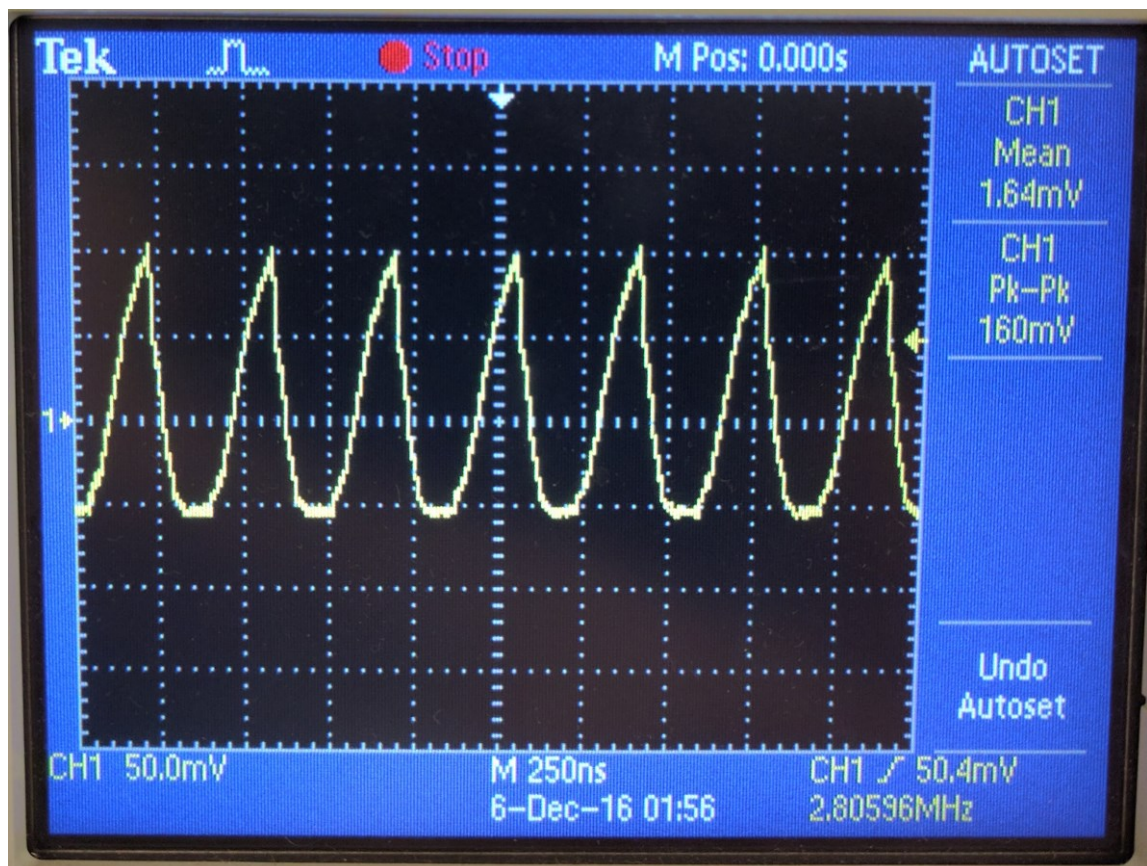


Figura 47 – Resultado do mapeamento de memória do dispositivo

Certamente, a frequência de 2.8MHz é um resultado sem precedentes para o processador de 1GHz da BeagleBone Black, mais de 4 vezes maior do que a metodologia de metal puro, que elimina a necessidade de utilizar um sistema operacional.

Vale ressaltar que não era esperado que o resultado do teste de mapeamento de memória do dispositivo superasse a aplicação em metal puro. Certas peculiaridades certamente influenciaram como fatores limitantes para o uso do JTAG como, por exemplo, o compilador CCSv6 que, por ter muitas ferramentas e informações sendo dispostas para o desenvolvedor através do terminal, tornou a aplicação mais lenta. Além disso, o código estava sendo executado na memória RAM, não em Flash. Outra característica é que o código estava sendo executado em debug, que não otimiza o código para que sejam geradas informações para depuração do mesmo, facilitando a identificação de erros e aumentando a produtividade, e não release, que faz, justamente, o oposto. Logo, os ambientes de execução não ofereciam, e nem poderiam oferecer, uma vez que as metodologias requerem situações diferentes, as mesmas condições de testes.

Outras características que valem ser apontadas são a estabilidade do sistema, que tem como produto uma forma de onda sem espectros ou grandes oscilações na frequência final vista através do osciloscópio, e a rapidez de, aproximadamente, 200 nano segundos para alternar entre nível lógico alto e baixo.

No entanto, também é importante ressaltar as limitações deste método. Primeiramente, há apenas quatro pinos que podem ser mapeados desta forma, segundo o manual de referência técnica do processador. Além disto para que seja possível utilizar este método, é necessário trabalhar com a placa no modo “root”.

Além disto, o fato de que a forma de onda está triangular, e não quadrada, assim como os demais resultados, aponta para a certeza de que o processador está trabalhando em uma frequência superior à de capacidade de atuação dos terminais. Ou seja, encontrou-se uma barreira em que a velocidade de acionamento do software é maior do que a do hardware. Mas, mesmo assim, aplicações onde não haja esta constante troca de nível lógico, em que a saída reproduza uma onda com frequência tão alta, o tempo de resposta ainda pode ser aproveitado.

2.3.6 Alterando a frequência da CPU

A BeagleBone Black, assim como outros computadores, possui reguladores que podem ser usados para perfilar e adequar a razão de seu desempenho sobre o consumo de energia na qual a mesma está operando. Por exemplo, se um projeto a ser criado com a placa utiliza uma fonte limitada de energia, desconectada de uma rede de alimentação, como uma bateria comum, e que exigisse baixos requisitos de processamento, a frequência do clock pode ser reduzida para economizar energia. O contrário também é válido. Caso um projeto não tenha barreiras quanto à alimentação e precisa de uma grande velocidade de processamento, essa frequência pode ser configurada para trabalhar em seu máximo.

Acessando o terminal do Linux (através dos passos contidos no Anexo I ou Anexo II), é possível encontrar informações sobre o estado atual da BBB digitando *cpufreq-info*.

Vale também ressaltar que esta ferramenta também funciona sob demanda. Ou seja, se a frequência da CPU está configurada para um certo valor, mas o código exige uma tarefa específica acima deste limite (conhecido como *up_threshold*, ou limiar superior), então a frequência da CPU será automaticamente aumentada.

No caso dos testes de velocidade apresentados neste documento é diferente. Os experimentos aqui presentes são comandos simples que fazem com que a BBB trabalhe em seu atual estado de configuração. Logo, para maximizar todos os resultados é preciso configurar, manualmente, a

frequência na qual se deseja operar. No contexto desta pesquisa, a frequência já havia sido estabelecida para a melhor performance antes de realizar os testes cujos resultados estão aqui representados. Logo, caso algum leitor tenha tentado reproduzir algum dos experimentos e não obteve o resultado esperado, é possível que ainda seja necessário realizar este ajuste.

Existem duas formas de manipular a frequência afim de adequar a operação da BBB para o projeto desejado. A primeira é através de perfis reguladores pré-estabelecidos. São eles: *conservative*, *ondeman*, *userspace*, *powersave* e *performance*. Os mesmos podem ser estabelecidos através do comando `sudo cpufreq-set -g perfil` (substituindo a palavra “perfil” pelo que se enquadra para o projeto). Como no exemplo a seguir, que foi utilizado para obter os resultados obtidos:

```
root@beaglebone:~# sudo cpufreq-set -g performance
root@beaglebone:~# cpufreq-info
cpufrequtils 008: cpufreq-info (C) Dominik Brodowski 2004-2009
Report errors and bugs to cpufreq@vger.kernel.org, please.
analyzing CPU 0:
  driver: generic_cpu0
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency: 300 us.
  hardware limits: 300 MHz - 1000 MHz
  available frequency steps: 300 MHz, 600 MHz, 800 MHz, 1000 MHz
  available cpufreq governors: conservative, ondemand, userspace, powersave, performance
  current policy: frequency should be within 300 MHz and 1000 MHz.
                    The governor "performance" may decide which speed to use
                    within this range.
  current CPU frequency is 1000 MHz (asserted by call to hardware).
  cpufreq stats: 300 MHz:56.18%, 600 MHz:0.13%, 800 MHz:0.21%, 1000 MHz:43.48% (12)
root@beaglebone:~#
```

Figura 48 – Frequência Máxima de CPU

Os perfis são ferramentas úteis que decidem, de forma autônoma, a frequência que a placa irá usar para operar um código, atendendo às necessidades de desenvolvedores que não sabem ao certo que frequência utilizar, porém conhecem as necessidades finais do projeto.

A segunda forma de manipular a frequência é estabelecendo um número fixo para o microcontrolador, através do comando `sudo cpufreq-set -f XMHz` (sendo X qualquer número entre 300 e 1000, alcance sugerido para a placa).

```

root@beaglebone:~# sudo cpufreq-set -f 600MHz
root@beaglebone:~# cpufreq-info
cpufrequtils 008: cpufreq-info (C) Dominik Brodowski 2004-2009
Report errors and bugs to cpufreq@vger.kernel.org, please.
analyzing CPU 0:
  driver: generic_cpu0
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency: 300 us.
  hardware limits: 300 MHz - 1000 MHz
  available frequency steps: 300 MHz, 600 MHz, 800 MHz, 1000 MHz
  available cpufreq governors: conservative, ondemand, userspace, powersave, performance
  current policy: frequency should be within 300 MHz and 1000 MHz.
                    The governor "userspace" may decide which speed to use
                    within this range.
  current CPU frequency is 600 MHz (asserted by call to hardware).
  cpufreq stats: 300 MHz:35.90%, 600 MHz:16.19%, 800 MHz:0.13%, 1000 MHz:47.78% (15)
root@beaglebone:~# █

```

Figura 49 – Escolhendo uma frequência específica

Uma vez que os resultados até agora foram obtidos utilizando a frequência máxima, alguns testes serão refeitos, para explorar a eficácia do método e sua utilidade, tanto para projetos que exijam desempenho, quanto para os que necessitam consumir menos energia, utilizando o perfil *powersave*.

```

root@beaglebone:~# sudo cpufreq-set -g powersave
root@beaglebone:~# cpufreq-info
cpufrequtils 008: cpufreq-info (C) Dominik Brodowski 2004-2009
Report errors and bugs to cpufreq@vger.kernel.org, please.
analyzing CPU 0:
  driver: generic_cpu0
  CPUs which run at the same hardware frequency: 0
  CPUs which need to have their frequency coordinated by software: 0
  maximum transition latency: 300 us.
  hardware limits: 300 MHz - 1000 MHz
  available frequency steps: 300 MHz, 600 MHz, 800 MHz, 1000 MHz
  available cpufreq governors: conservative, ondemand, userspace, powersave, performance
  current policy: frequency should be within 300 MHz and 1000 MHz.
                    The governor "powersave" may decide which speed to use
                    within this range.
  current CPU frequency is 300 MHz (asserted by call to hardware).
  cpufreq stats: 300 MHz:59.80%, 600 MHz:0.34%, 800 MHz:0.57%, 1000 MHz:39.29% (11)
root@beaglebone:~# █

```

Figura 50 – Estabelecendo economia de energia

Primeiro, repetindo 2.3.3, utilizando as bibliotecas em C padrão.

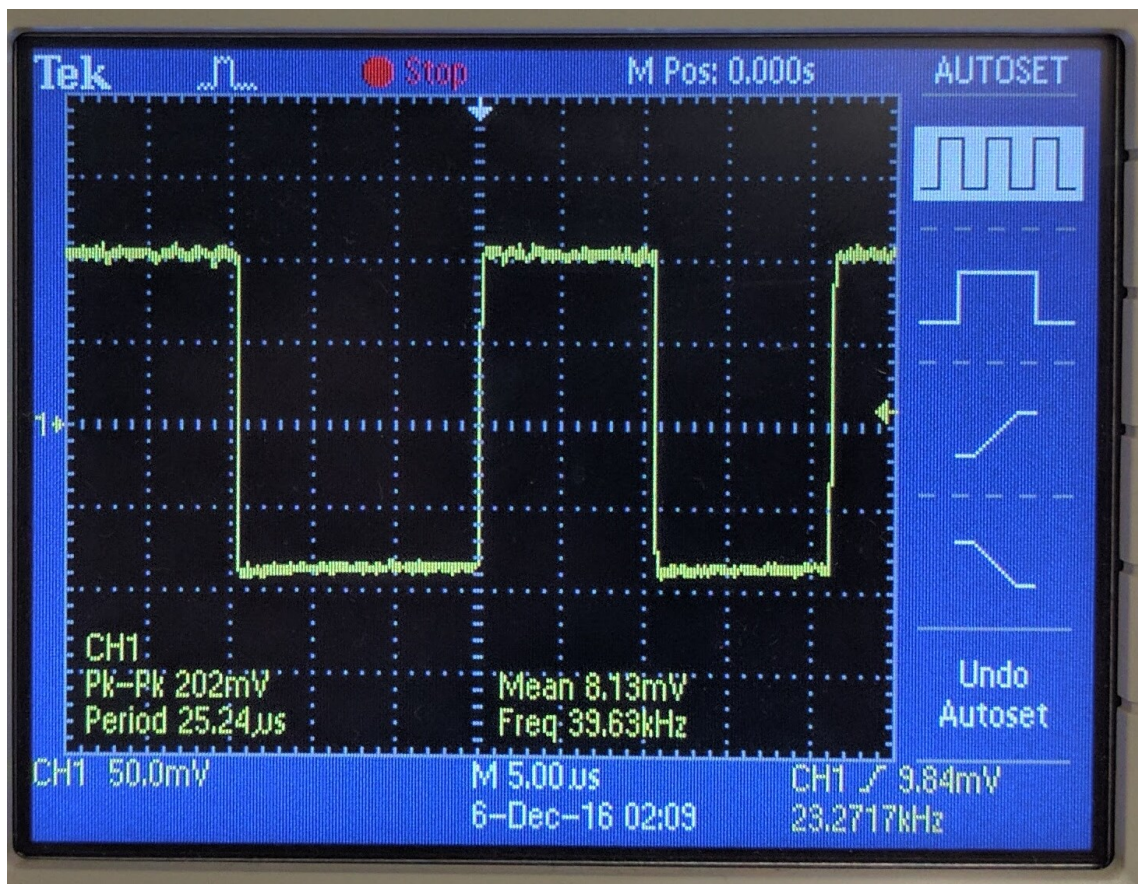


Figura 51 – Experimento em C padrão no modo de economia de energia

Agora, repetindo 2.3.5, que usa a memória do dispositivo.

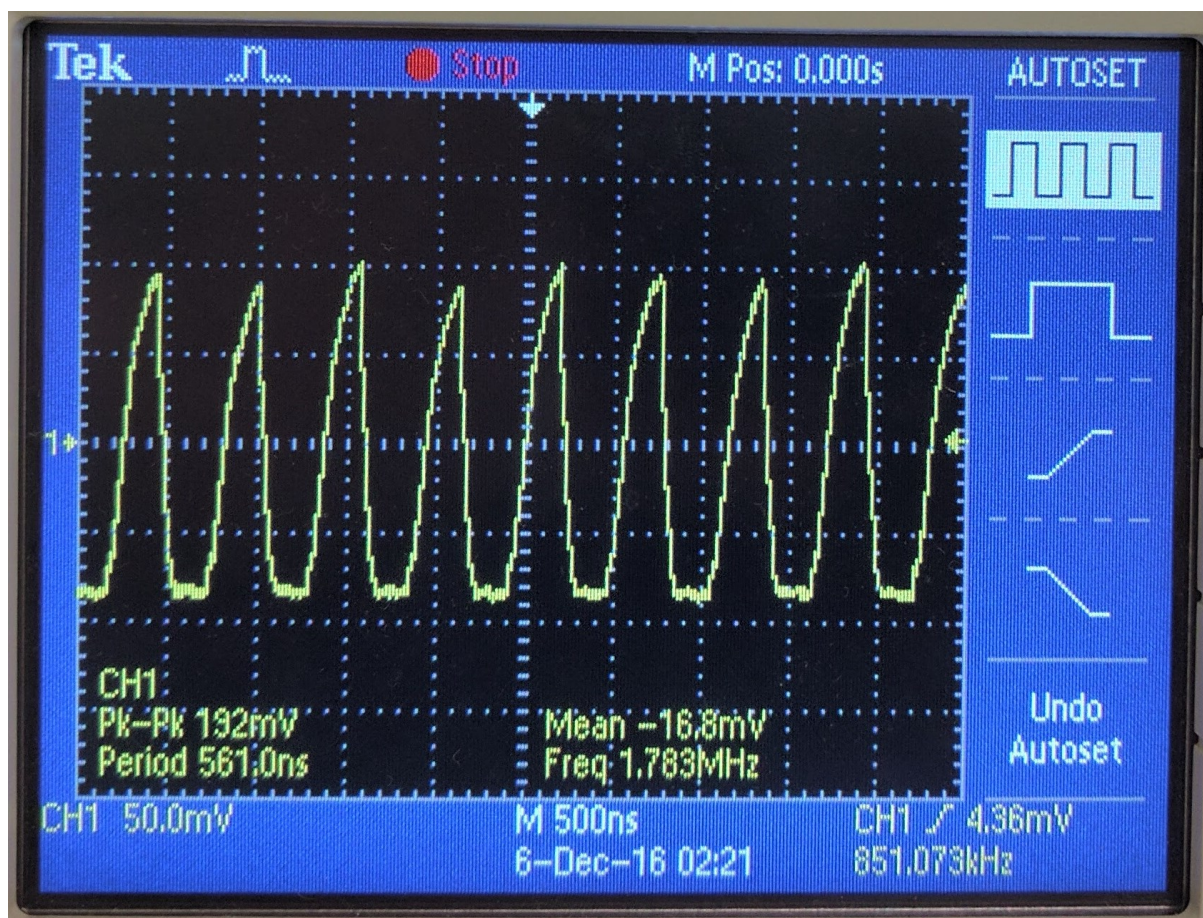


Figura 52 – Experimento da memória do dispositivo no modo de economia de energia

Ambos tiveram reduções significativas no tempo de resposta e frequência de trabalho, como esperado. O experimento de C padrão reduziu de 162kHz para 40kHz, enquanto o que utiliza a memória do dispositivo foi de 2.8MHz para 1.8MHz.

Como esperado, os dois não reduziram na mesma razão, uma vez que o perfil não estabelece uma frequência fixa sempre. No lugar disso, ele analisa o trabalho realizado e o consumo de energia e traça a frequência de operação sempre que um código é executado, de acordo com a necessidade de cada caso.

2.4 Discussão

Uma vez que os resultados foram expostos e explicados individualmente sob cada subseção deste capítulo, nesta etapa serão comparados, avaliados e observados, de forma mais crítica, cada um deles. Para tal, cada metodologia será encaixada em uma tabela, para obter-se uma análise mais objetiva.

Cada tabela contém os dados mais relevantes obtidos no estudo: a frequência máxima obtida e o tempo de resposta absoluto, equivalente a meio período, uma vez que cada período reflete o

tempo de duas instruções (alternando entre nível lógico alto e baixo). Além disso, são também listadas as vantagens e desvantagens observadas ao longo do processo de testes e coleta de dados. Esta lista pode refletir, em alguns casos, características relativas, ou seja, pontos que positivos ou negativos apenas se comparadas às demais metodologias.

No fim, o leitor deve ser capaz de observar e concluir, através desta sumarização, presente nas tabelas de 1 a 6, qual dos métodos são aplicáveis para um possível projeto, tendo em vista os objetivos e requerimentos que o mesmo deve cumprir. Por exemplo, um projetista iniciante provavelmente irá optar por uma abordagem com programação simplificada ou, caso trabalhe com processos mais críticos, uma mais veloz.

Visão Geral - Python

Frequência 7.6KHz	Tempo de Resposta 65 μ s
Vantagens Configuração simples; Fácil programação; Metodologia prática.	
Desvantagens Baixa estabilidade ao operar em velocidade máxima;	

Tabela 1 - Python

Visão Geral - C: Biblioteca de uso específico

Frequência 8.6KHz	Tempo de Resposta 58 μ s
Vantagens Configuração simples; Fácil programação.	
Desvantagens Baixa estabilidade ao operar em velocidade máxima; Necessário instalar biblioteca GPIO.	

Tabela 2 – C: Biblioteca de uso específico

Visão Geral - C: Biblioteca padrão

Frequência 162.3KHz	Tempo de Resposta 3 μ s
Vantagens Sistema estável; Configuração simples.	
Desvantagens Conhecimento intermediário de programação necessário.	

Tabela 3 – C: Biblioteca padrão

Visão Geral - Metal Puro

Frequência 639KHz	Tempo de Resposta 782 ns
Vantagens Sistema estável; Ausência de sistema operacional; Ferramentas disponíveis do CCS.	
Desvantagens Conhecimento intermediário de programação necessário; Configuração de hardware trabalhosa; Custo extra com cabo JTAG.	

Tabela 4 – Metal Puro

Visão Geral - Memória do dispositivo

Frequência 2.8 MHz	Tempo de Resposta 178 ns
Vantagens Sistema estável; Maior velocidade registrada.	
Desvantagens Conhecimento avançado de programação necessário; Conhecimento avançado do hardware; Acesso às permissões de administrador da placa.	

Tabela 5 – Memória do dispositivo

Visão Geral - Frequência da CPU

Frequência Variável	Tempo de Resposta Variável
Vantagens Aplicável em conjunto com qualquer método; Capacidade de otimizar performance ou economia.	
Desvantagens Conhecimento acerca do sistema operacional	

Tabela 6 – Frequência da CPU

3 CONCLUSÕES

O presente documento, através de uma análise extensiva acerca do tempo de resposta de um dispositivo em especial, agrega conteúdo de alta relevância ao tópico de desenvolvimento de sistemas embarcados. Este assunto, cada vez mais presente no cotidiano da tecnologia moderna, está, constantemente, em expansão, ajudando a otimizar projetos de automação, robótica e que envolvem algum tipo de controle em geral, reduzindo tamanho, recursos computacionais e custos de um determinado produto.

Foi cumprido também, como objetivo, a geração de materiais, em português, que auxiliem um projetista a desenvolver um sistema qualquer, através de esforços, por parte do autor, em expor o conhecimento de forma gradual, caminhando do básico ao avançado. Além disso, foram escritos manuais de como configurar a BeagleBone Black de maneiras distintas, cada qual para um determinado uso, presentes nos anexos.

Por mais que todos os objetivos tenham sido cumpridos, trabalhos futuros podem ser feitos para estender este conteúdo. Seria possível abordar, por exemplo, novas técnicas de uso da placa, outras linguagens de programação ou também compiladores distintos disponíveis. Outra alternativa interessante seria também realizar processos análogos com outro microcontrolador, que utilize o Linux como sistema operacional, afim de comparar os resultados entre eles. E, por fim, há a possibilidade de reproduzir os mesmos processos utilizando sensores acoplados aos periféricos da placa, com o intuito de trazer o estudo teórico mais próximo à prática.

4 ANEXOS

4.1 Anexo I: Cloud9

Para utilizar o Cloud9, é primeiro necessário conectar a BBB através da porta Mini USB ao computador. Alguns programas serão instalados automaticamente para que seja possível acessar os arquivos da placa. Esses arquivos constituem de uma cópia da documentação da BBB e alguns drivers.

Agora, para permitir ao sistema operacional do computador usado acesso de rede através do USB para a BeagleBone estes drivers presentes na placa precisam ser executados e instalados. A Figura 53 mostra o que o desenvolvedor encontrará dentro da pasta da placa.

BeagleBone Getting Started (G:)			
Name	Date modified	Type	
App	14-May-14 9:16 PM	File folder	
debug	15-May-14 1:19 AM	File folder	
Docs	14-May-14 9:16 PM	File folder	
Drivers	14-May-14 9:16 PM	File folder	
dtbs	14-May-14 9:16 PM	File folder	
scripts	14-May-14 9:16 PM	File folder	
autorun	14-May-14 9:16 PM	Setup Information	
ID	14-May-14 9:16 PM	Text Document	
initrd	15-May-14 1:19 AM	Disc Image File	
LICENSE	14-May-14 9:16 PM	Text Document	
MLO	15-May-14 1:20 AM	File	
README	14-May-14 9:16 PM	HTM File	
README.md	14-May-14 9:16 PM	MD File	
SOC.sh	14-May-14 9:16 PM	SH File	
START	14-May-14 9:16 PM	HTM File	
u-boot	15-May-14 1:20 AM	Disc Image File	
uEnv	15-May-14 1:20 AM	Text Document	

Figura 53 – Pasta inicial da BeagleBone Black

Em destaque na Figura 53 há uma pasta chamada Drivers. Dentro dela há opções de sistemas operacionais distintas. O usuário deve escolher a opção que se encaixa em seu caso.

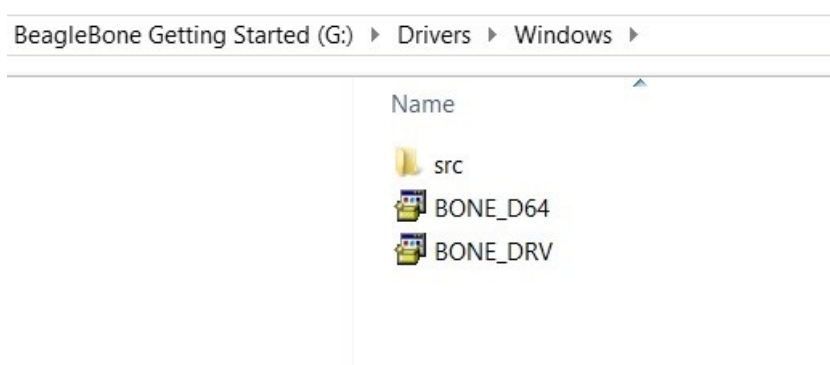


Figura 54 – Opções de drivers

Após fazer a escolha correta, basta instalar os drivers da maneira padrão. Caso ocorra algum erro, ou por algum motivo a pasta não esteja presente dentro dos arquivos contidos na BeagleBone Black, existe uma forma de obter o driver online em <http://beagleboard.org/getting-started>.

Step 2

▶

Install drivers

Install the drivers for your operating system to give you network-over-USB access to your Beagle. Additional drivers give you serial access to your board.

Operating System	USB Drivers	Comments
Windows (64-bit)	64-bit installer	If in doubt, try the 64-bit installer first. - Note #1: Windows Driver Certification warning may pop up two or three times. Click "Ignore", "Install" or "Run" - Note #2: To check if you're running 32 or 64-bit Windows see this: https://support.microsoft.com/kb/827218 - Note #3: On systems without the latest service release, you may get an error (0xc000007b). In that case, please install the following and retry: https://www.microsoft.com/en-us/download/confirmation.aspx?id=13523 - Note #4: You may need to reboot Windows. - Note #5: These drivers have been tested to work up to Windows 10
Windows (32-bit)	32-bit installer	
Mac OS X	Network Serial	Install both sets of drivers.
Linux	mkudevrule.sh	Driver installation isn't required, but you might find a few udev rules helpful.

Figura 55 – Opções de drivers online

A partir de agora, conexão com a Internet é imprescindível durante todo o tempo. Depois de instalar os drivers, existe uma plataforma completa para navegar em sua BeagleBone Black através do navegador de Internet. No entanto, vale ressaltar que há suporte apenas para os navegadores Chrome ou Firefox (o processo não irá funcionar para o Internet Explorer).

Dentro desta plataforma está o primeiro dois IDEs que serão exibidos neste documento: o Cloud9. Para acessá-lo basta digitar o endereço 192.168.7.2:3000 no navegador e submeter. É

importante ressaltar, no entanto, que a página não irá carregar caso os drivers não tenham sido instalados ou forem mal executados. Caso isso ocorra, basta repetir o processo de instalação de drivers. A Figura 56 mostra o como é o primeiro contato com a interface.

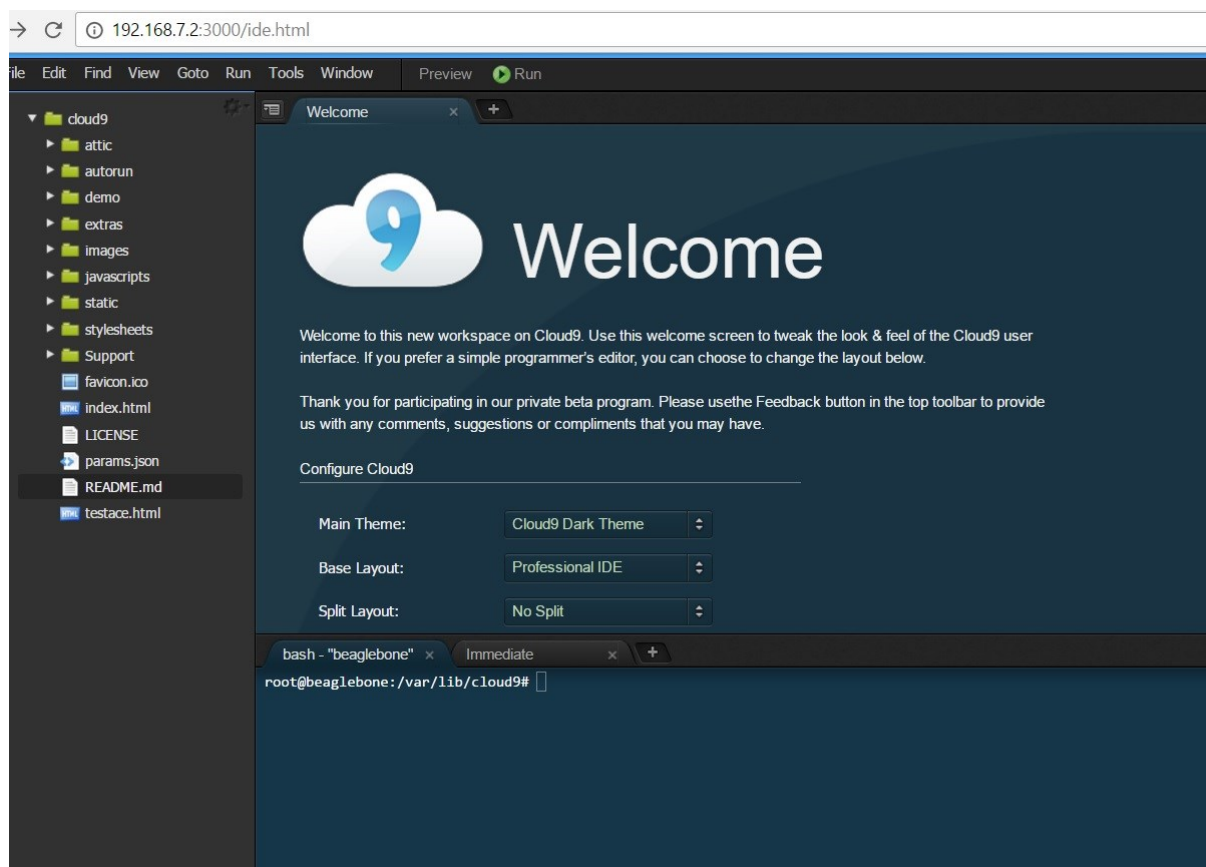


Figura 56 – Primeiro contato com Cloud9

Este ambiente de desenvolvimento integrado tem suporte para diversas linguagens de programação, incluindo C, C++, PHP, Ruby, Perl, Python, JavaScript com Node.js e Go. Sua praticidade permite que desenvolvedores e projetistas comecem a programar quase que imediatamente em áreas de trabalho pré-configuradas.

Além disso, há outras vantagens como o terminal embutido (que pode ser visto na parte inferior da Figura 56), que permite acessar a BeagleBone Black com os comandos básicos do Linux e executar programas na placa diretamente do Cloud9, assim como parar sua execução em apenas um clique.

À esquerda, existe uma pasta chamada “demo” já com exemplos de programas em .js e .py editáveis que podem ser executados para testar a eficiência do IDE. Ademais, para iniciar seus próprios projetos, o desenvolvedor já pode criar seus próprios arquivos programáveis. Para isto,

basta clicar com o botão direito na pasta onde deseja criar o arquivo e selecionar a opção “New File” como mostra a Figura 57.

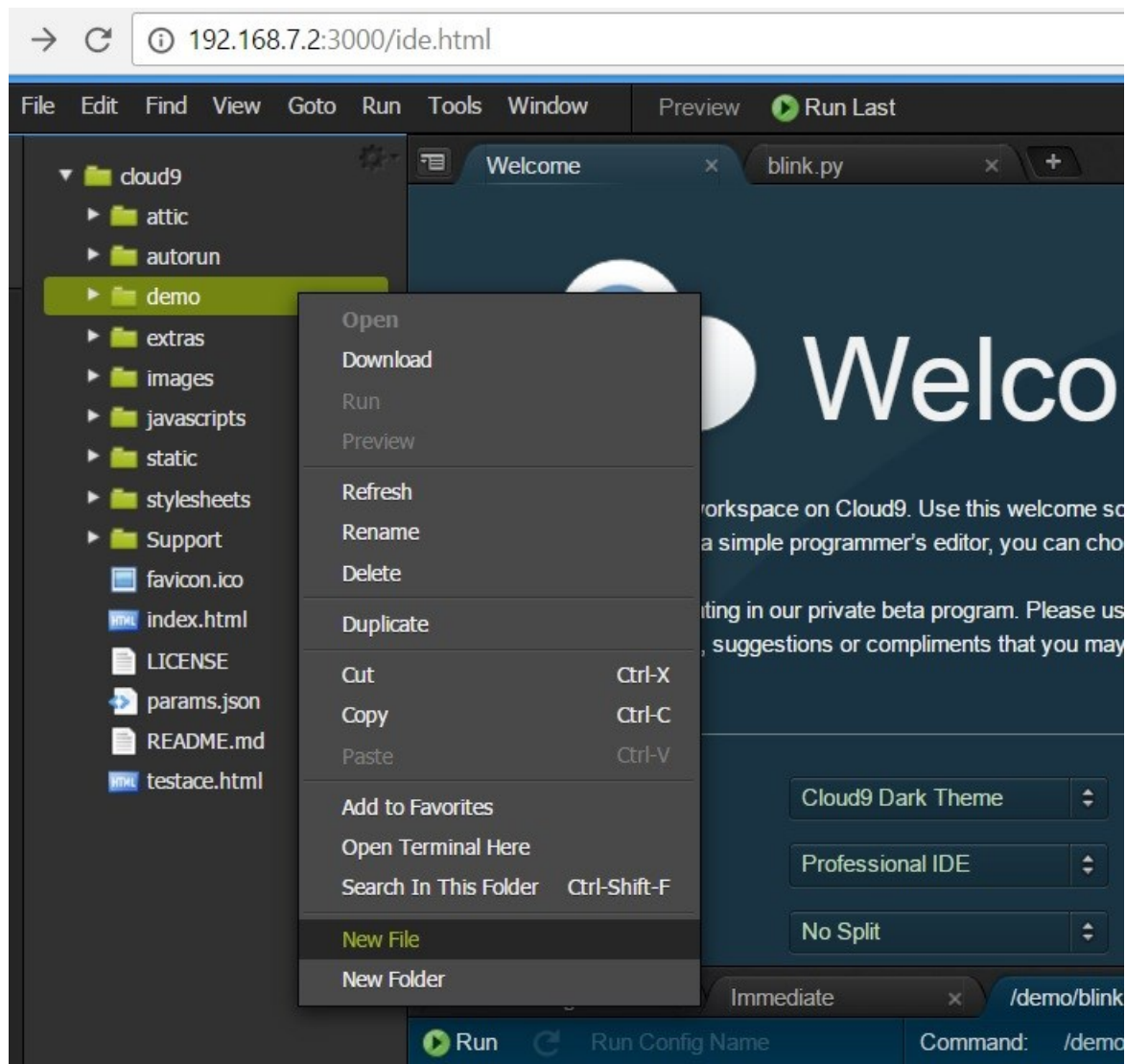


Figura 57 – Criando um arquivo no Cloud9

Então, para que este arquivo criado seja um programa executável pela BeagleBone através do Cloud9, é preciso nomeá-lo de acordo com a linguagem usada. Por exemplo, arquivos de linguagem C devem terminar com `.c`, linguagem Python devem terminar com `.py` e linguagem JavaScript devem terminar com `.js`.

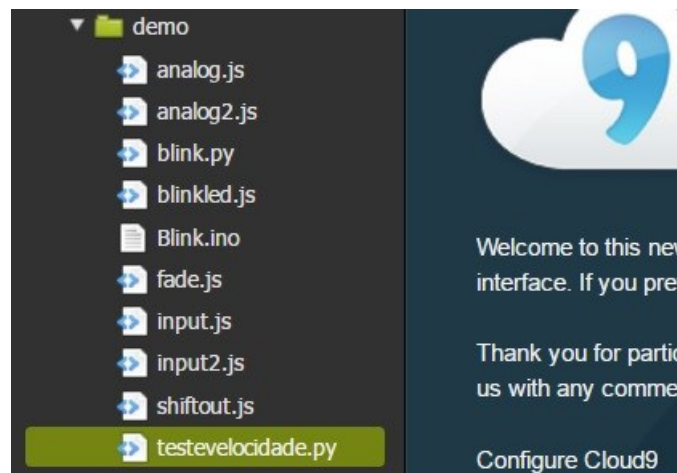


Figura 58 – Arquivo nomeado para programação em Python

A partir daqui o desenvolvedor já pode criar seus próprios programas e executá-los ao clicar no botão “Run” presente no topo do ambiente.

4.2 Anexo II: Eclipse IDE

Primeiramente, é necessário baixar e executar um software chamado Minimalist GNU for Windows. O mesmo está disponível em <https://sourceforge.net/projects/mingw/files/Installer/>.

mingw-get	2013-10-04
mingw-get-setup.exe	2013-10-04
readme.txt	2013-10-04
Totals: 3 Items	

Figura 59 – Executável do MinGW

Basta baixar o arquivo circulado em vermelho e executá-lo de forma padrão. Em seguida, uma nova janela chamada “MinGW Installation Manager” irá surgir. No canto superior esquerdo há uma aba chamada “Basic Setup” que deve ser aberta. Ao fazê-lo, o usuário deve então, com o botão direito do mouse, marcar as caixas “mingw32-base” e “mingw32-gcc-g++” com “mark for installation”.

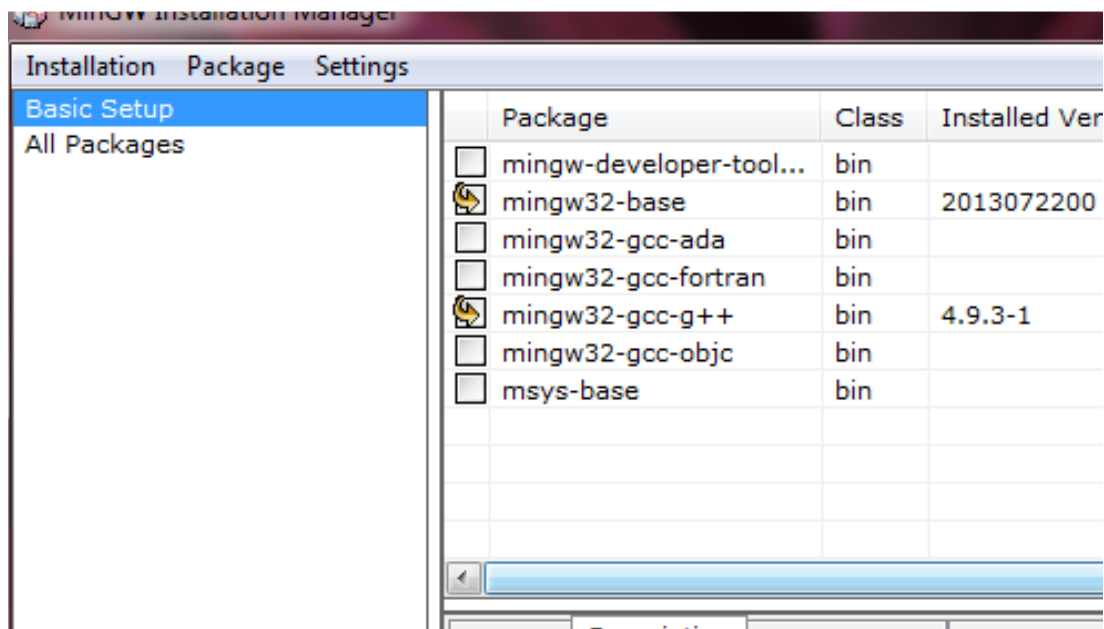


Figura 60 – Compiladores C/C++ do Eclipse

Agora no canto superior direito, na aba “Installation” deve-se clicar em “Apply Changes”

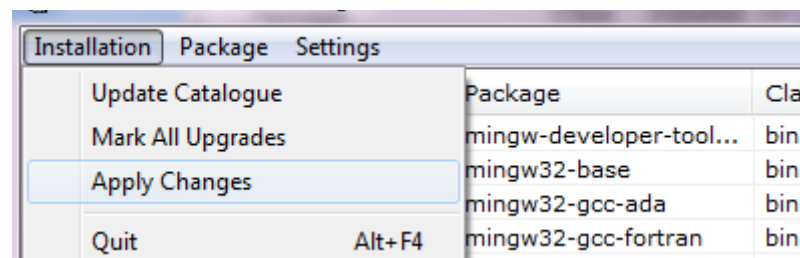


Figura 61 – Instalação dos compiladores

Para confirmar que a instalação obteve êxito, o usuário pode entrar em seu diretório C:/ e abrir a pasta bin dentro de MinGW. Caso os arquivos c++.exe e g++.exe estejam lá, os compiladores foram bem executados.

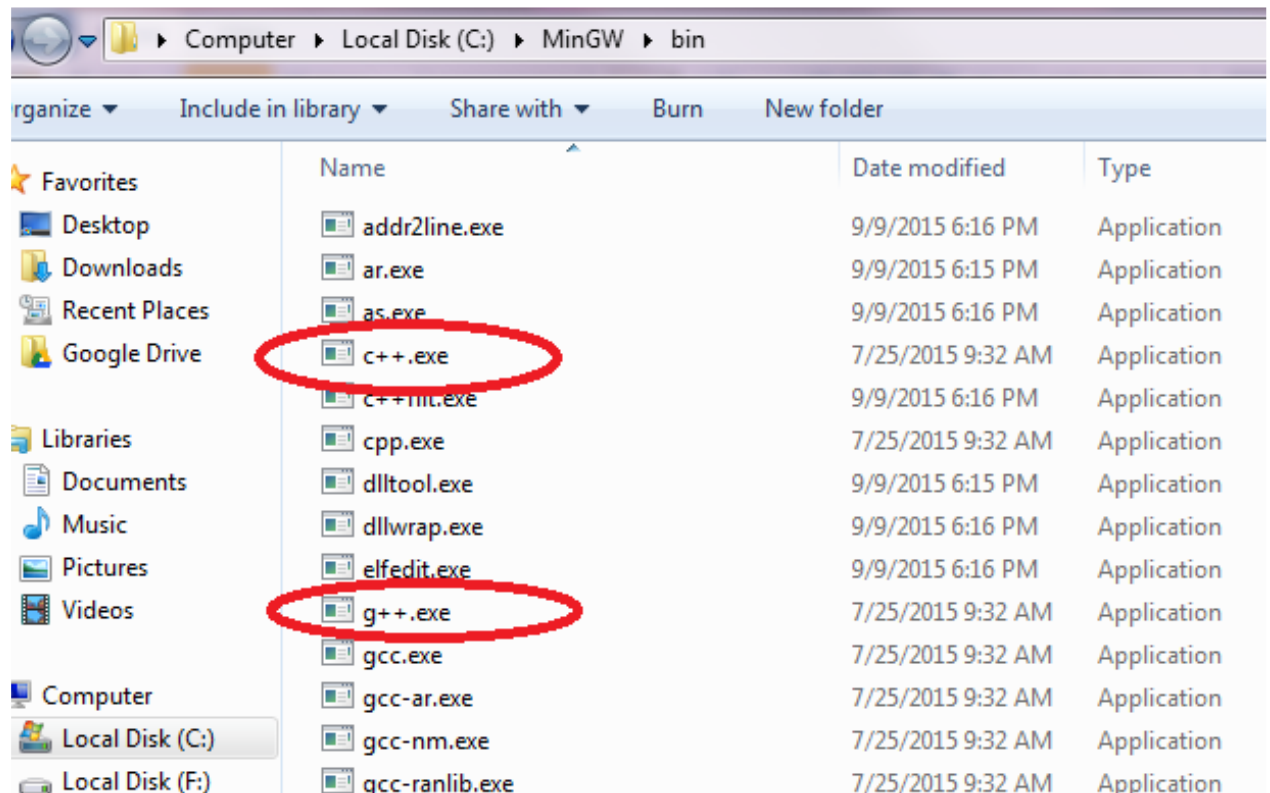


Figura 62 – Confirmando instalação

Agora que os compiladores estão em ordem, o próximo passo é obter o Eclipse. O mesmo está disponível para ser baixado para Windows e Mac OS em no site <http://www.eclipse.org/downloads/packages/eclipse-standard-431/keplersr1>.

Eclipse Standard 4.3.1

Package Description

The Eclipse Platform, and all the tools needed to develop and debug it: Java and Plug-in Development Tooling, Git and CVS support, including source and developer documentation. Click [here](#) to file a bug against Eclipse Platform.

Download Links

Windows 32-bit
Windows 64-bit
 Mac OS X (Cocoa) 32-bit
 Mac OS X (Cocoa) 64-bit

Figura 63 – Opções do Eclipse

Depois de baixar a versão compatível com seu sistema, o usuário precisa apenas extrair a pasta no diretório C:/, ou seja, não é necessário efetuar o processo de instalação.

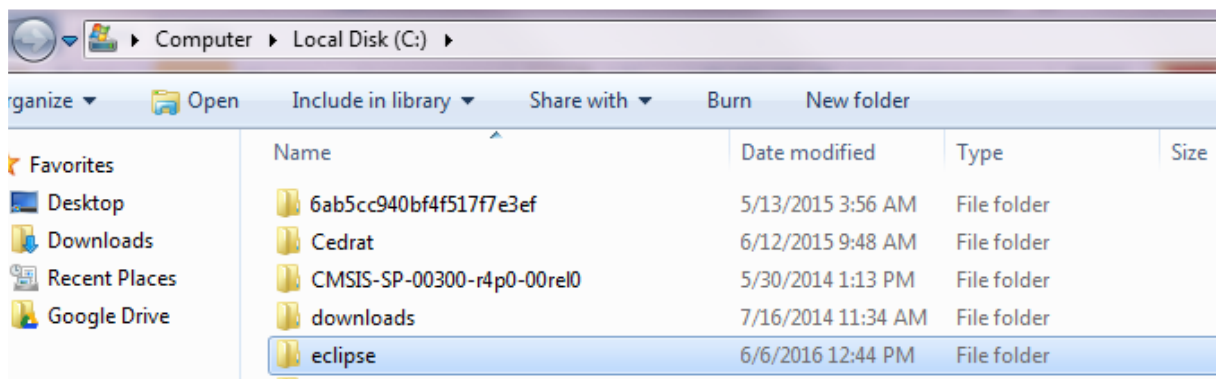


Figura 64 – Eclipse extraído

É recomendado apenas que o executável do programa seja adicionado a um atalho de fácil acesso. Ao abrir o Eclipse, é necessário configurá-lo corretamente. No topo da janela, o usuário deve clicar em “Help”, então “Install New Software” e, em seguida, “Add”. Em “Location” é necessário digitar o link <http://download.eclipse.org/tools/cdt/releases/kepler> e em “Name” digitar “Eclipse CDT”.

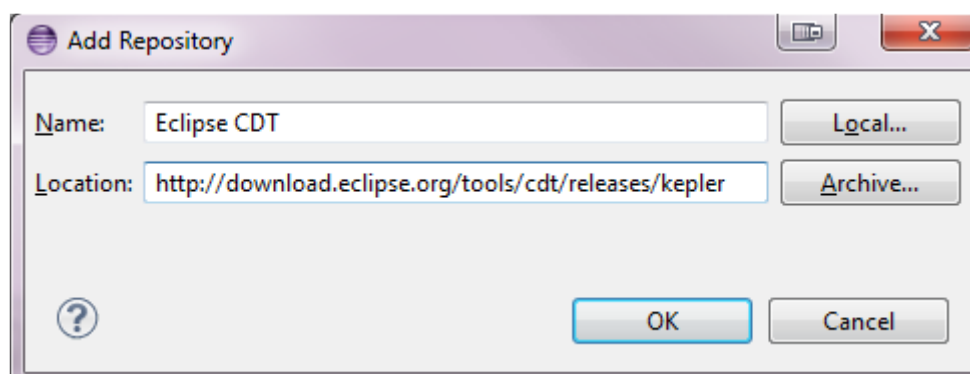


Figura 65 – Configurando o Eclipse

Depois de alguns momentos necessários para que o Eclipse se conecte à fonte e encontre as ferramentas necessárias, o usuário precisa marcar apenas a opção “CDT Main Features” e clicar em “Next”.

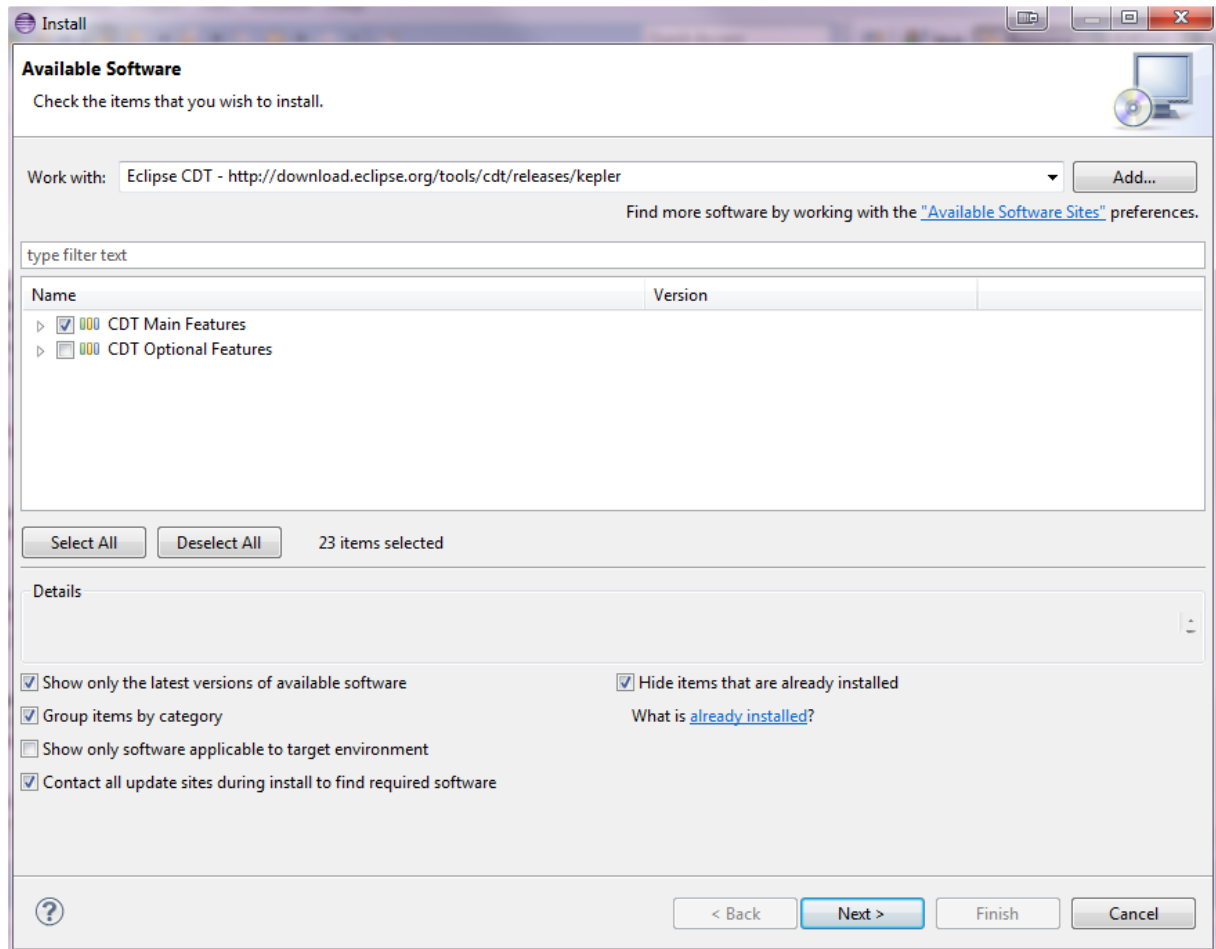


Figura 66 – Eclipse CDT

Depois de reiniciar o Eclipse, o resultado de todo esse processo é a capacidade agora de programar e compilar programas em C e C++ utilizando esse IDE. Para criar um projeto, basta clicar no canto superior esquerdo “File”, “New” e então “Project”. Na nova janela que surgir, deve-se selecionar em “C/C++” a opção “C Project” e então clicar em “Next”.

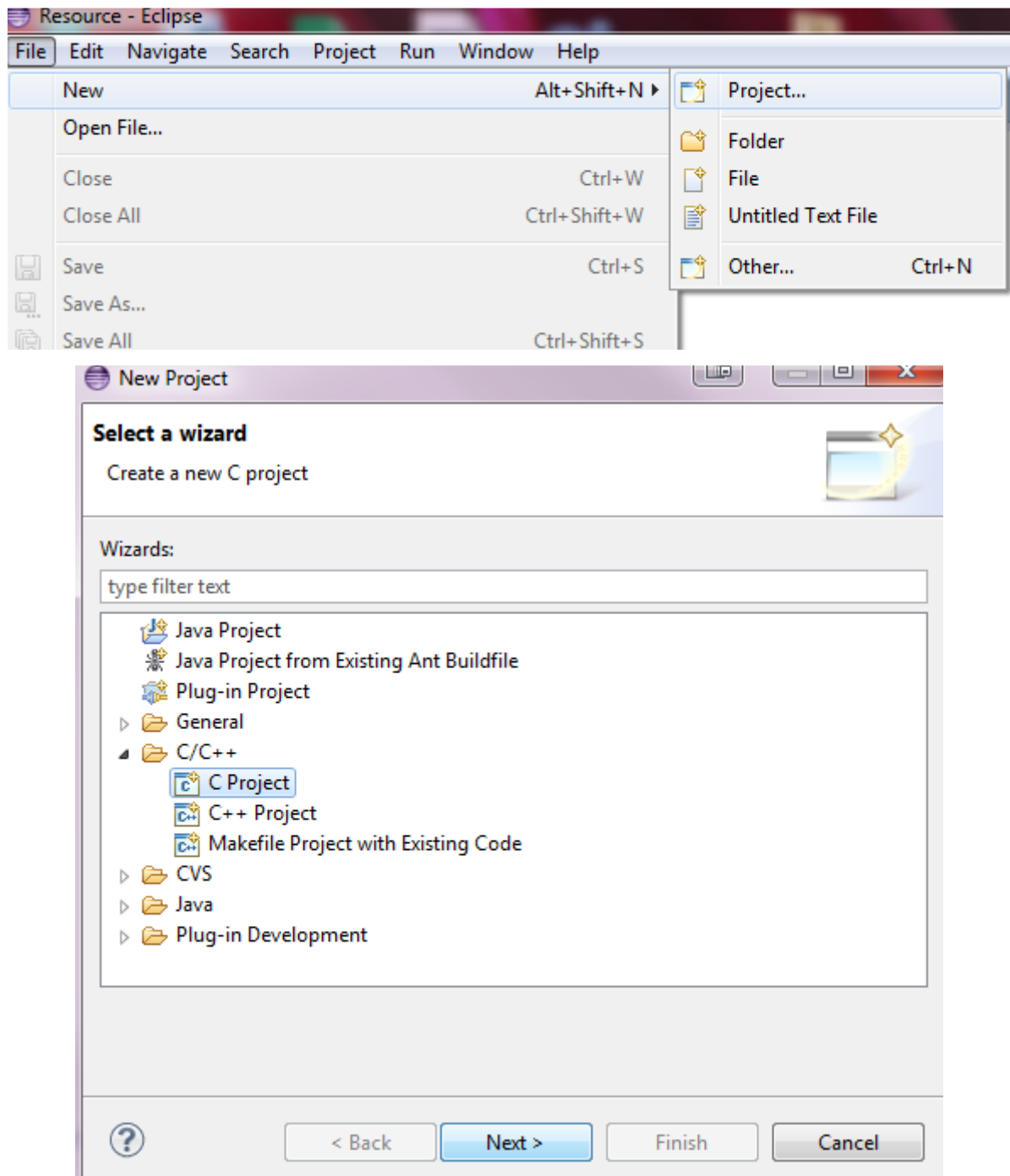


Figura 67 – Criando um projeto em C

Agora o usuário precisa dar um nome ao projeto. Em “Project Type” selecionar na pasta “Executable” a opção “Empty Project” e, ao lado, em “Toolchains” selecionar “MinGW GCC”. Depois de configurar estas opções, o botão “Finish” criará o projeto.

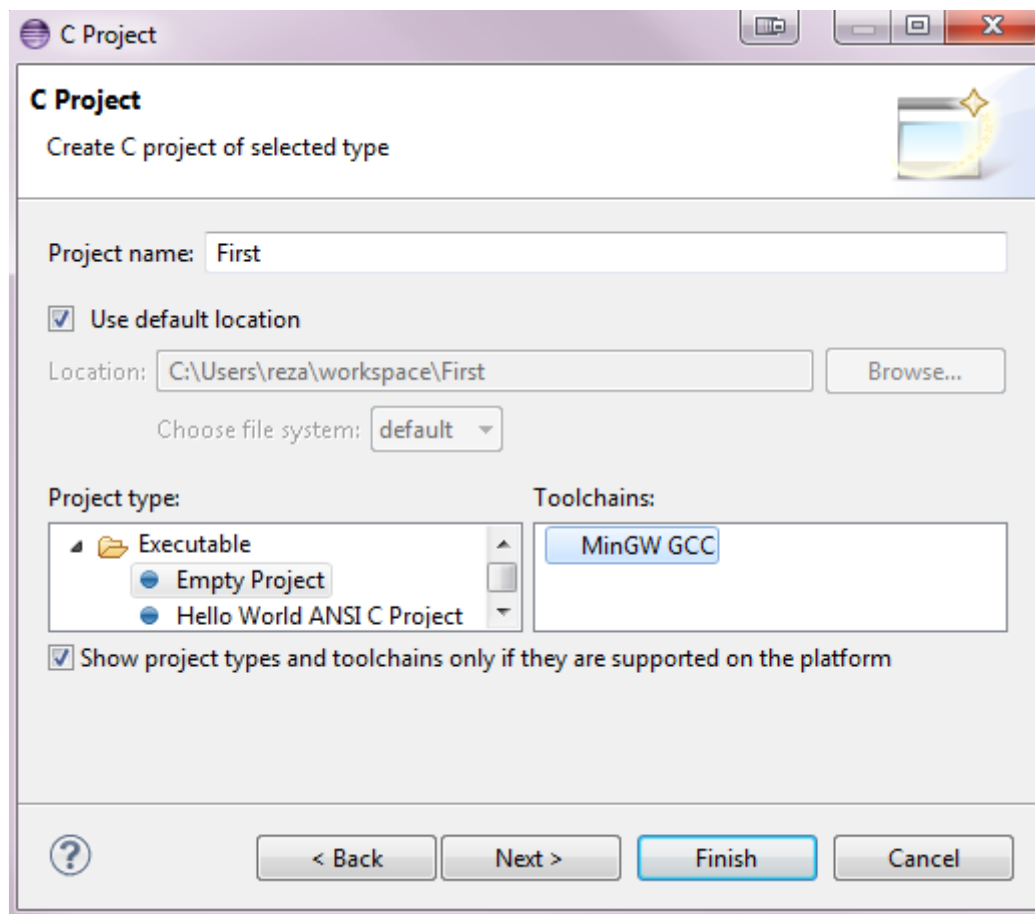


Figura 68 – Finalizando o projeto em C no Eclipse

Em seguida, para criar um arquivo para programar dentro do projeto, é preciso clicar no ícone de nova perspectiva no canto superior direito (com um pequeno símbolo de adição amarelo), então selecionar “C/C++” e confirmar. Assim que isto for feito, o Eclipse abrirá a perspectiva em C/C++, onde será possível visualizar todos os projetos criados, inclusive o que foi feito nos passos anteriores. Quando o usuário clicar com o botão direito do mouse no projeto, ele poderá selecionar “New” e “Source File” como na Figura 69.

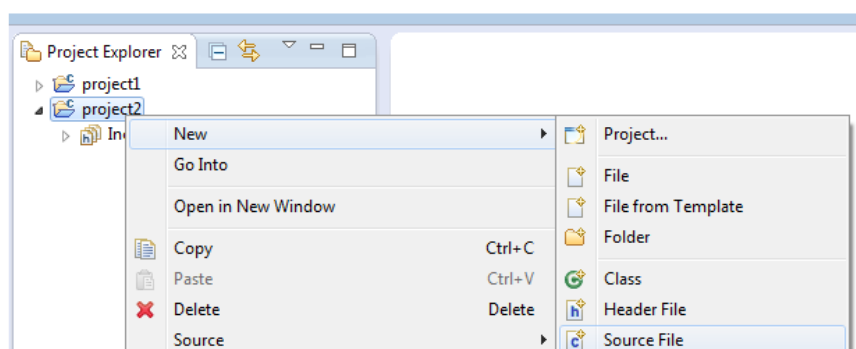


Figura 69 – Criar novo arquivo de fonte

A pasta em si será a pasta do projeto, já possui um nome dado anteriormente. Agora o arquivo fonte, onde é feito o programa, deve ser nomeado de acordo com a linguagem que será usada para programar, terminando em “.c” para linguagem C ou em “.cpp” para linguagem C++.

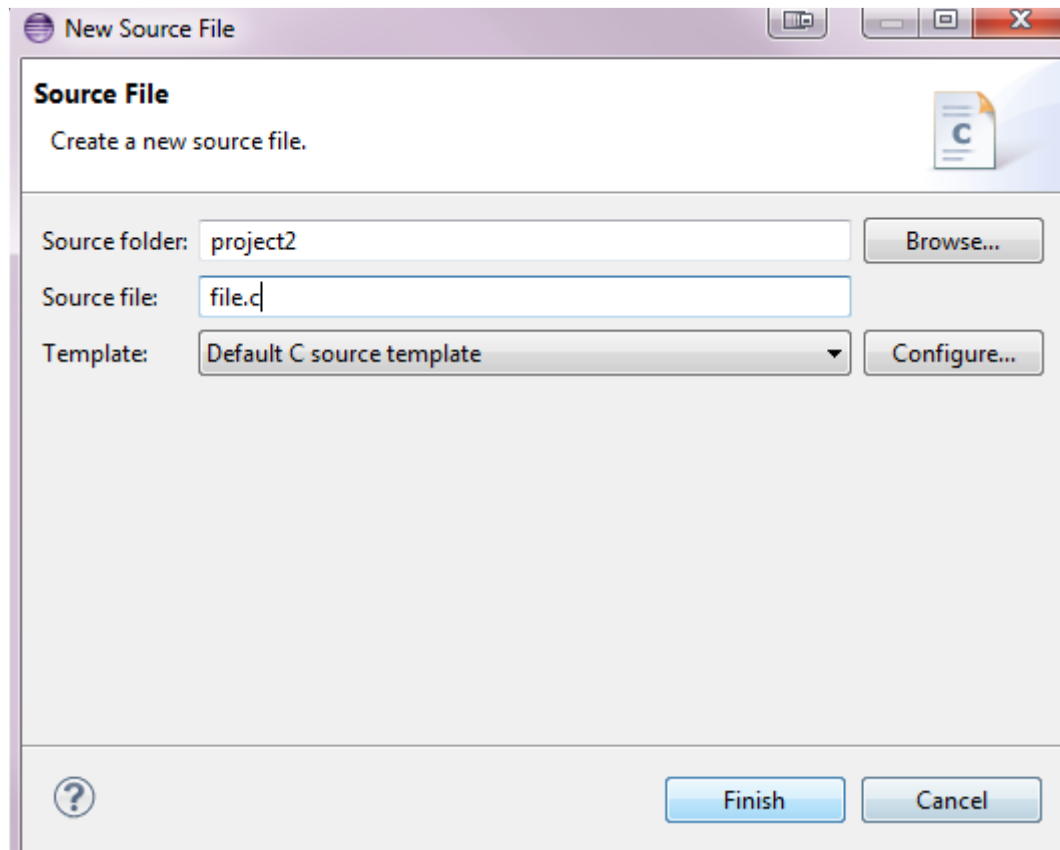


Figura 70 – Dando nome ao arquivo fonte

Ao finalizar, o arquivo estará presente dentro da pasta do projeto e pode ser aberto ao clicar duas vezes com o botão esquerdo do mouse sobre ele.

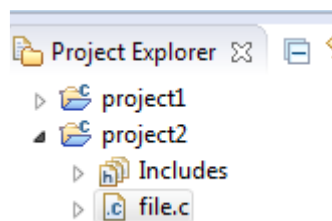


Figura 71 – Arquivo criado dentro do projeto

Finalmente, uma vez que já se sabe como criar um projeto em C, o último passo consiste em criar a comunicação entre Eclipse e BeagleBone Black. Para isso, é necessário acessar o site <http://archive.eclipse.org/tm/downloads/drops/R-3.3.1-201109141310/> e baixar o arquivo “RSE-

runtime-3.3.1.zip”, que significa Remote System inside Eclipse (sistema remoto dentro do Eclipse).

All-in-one (includes runtime and documentation for dstore, ftp, local, ssh, and telnet)		
For tool writers, the RSE SDK also includes the runtime as listed, plus programmer documentation and source.		
Status	Platform	Download
✓	All	RSE-runtime-3.3.1.zip (Recommended for most users)
✓	All	RSE-SDK-3.3.1.zip
✓	All	TM-repo-3.3.1.zip (Downloadable p2 repository including

Figura 72 – Sistema remoto dentro do Eclipse

Em seguida, é preciso extrair o conteúdo da pasta baixada na pasta “features”, dentro do endereço do Eclipse no computador do usuário.

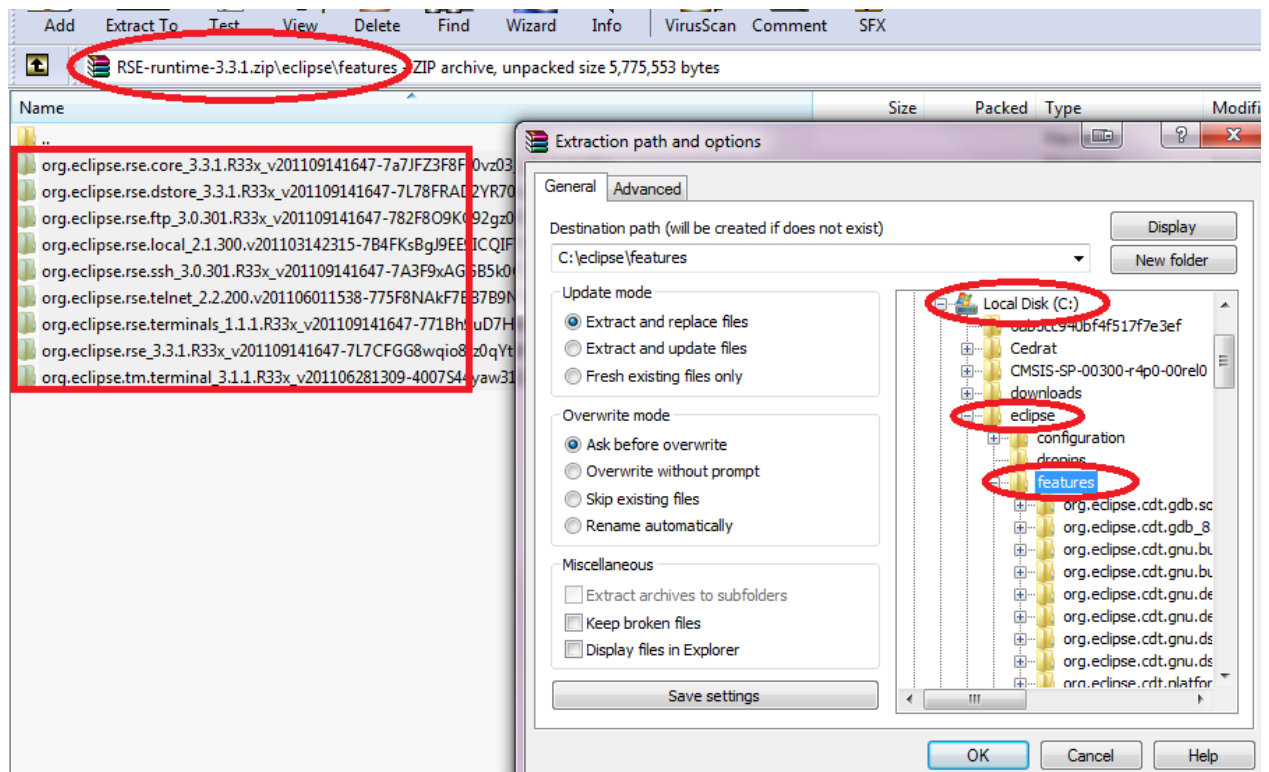


Figura 73 – Extraindo o RSE

Logo depois, deve-se extrair todos os arquivos com a terminação .jar na pasta eclipse/plugins/, exatamente como no passo anterior, mas agora dentro da pasta “plugins”.

No momento após isto ser feito já é possível estabelecer uma nova conexão entre a BeagleBone Black e o Eclipse. Assim que a placa estiver conectada via Mini USB, é possível abrir a aba de sistemas remoto como mostra a Figura 74.

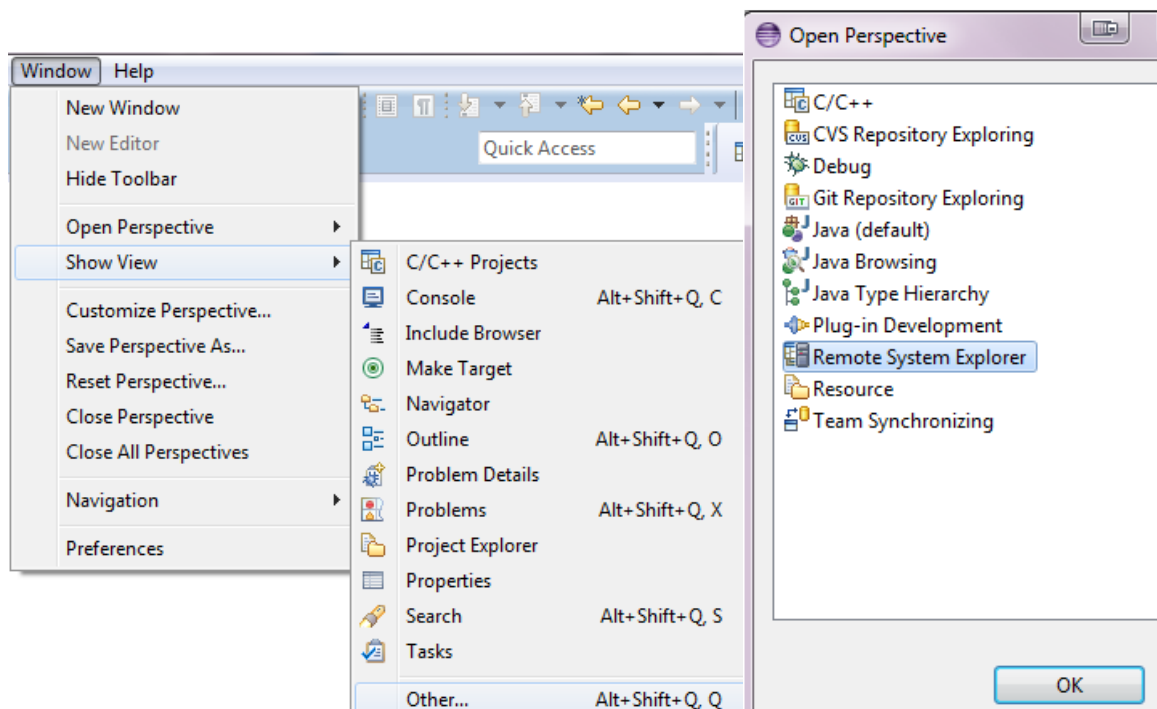


Figura 74 – Perspectiva de sistemas remotos

Em seguida, nesta nova aba, em “Local” o usuário deve clicar em “New”, “Connection”, “Linux” e então “Next”.

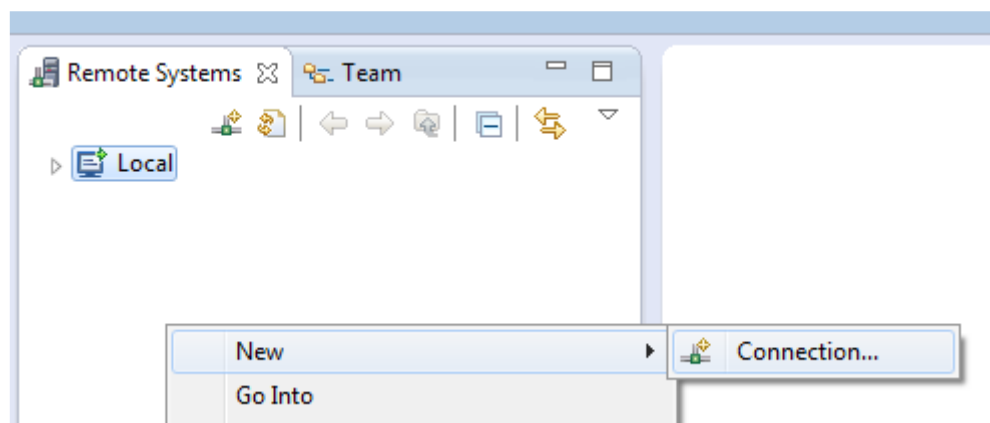


Figura 75 – Estabelecendo uma nova conexão

Por padrão, o “Host name” da BBB é o endereço de IP 192.168.7.2. Já o nome da conexão pode ser escolhido pelo usuário, por exemplo, BeagleBoneBlack.

Host name:	192.168.7.2
Connection name:	BeagleBoneBlack
Description:	

Figura 76 – Configurando a conexão

Uma das vantagens da BeagleBone Black citadas anteriormente acerca do sistema operacional Linux foi a presença de pacotes de segurança de conexão facilmente implementados, no caso o Secure Shell. Neste passo é possível implementá-los de forma rápida, logo depois de nomear a conexão.

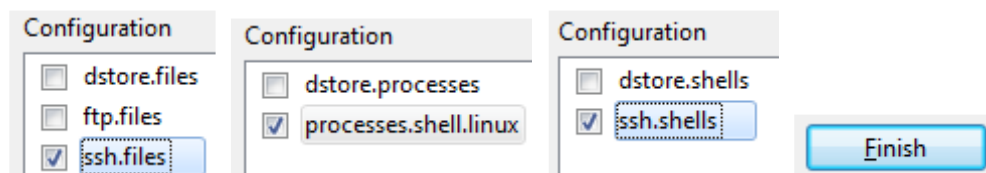


Figura 77 – Estabelecendo pacotes de segurança SSH

Agora uma nova conexão entre BBB e Eclipse pode ser estabelecida de forma segura. Caso o usuário não seja capaz de visualizar essa conexão no lado esquerdo da interface do programa, basta clicar no canto superior direito “Remote System Explorer” para que a aba referente a mesma seja aberta.

Nesta nova aba todos os sistemas estarão visíveis: local e todas as conexões remotas criadas, como a BeagleBoneBlack exemplificada. Ao clicar na conexão criada e selecionar “Connect” uma janela referente à placa irá abrir, pedindo ID e senha.

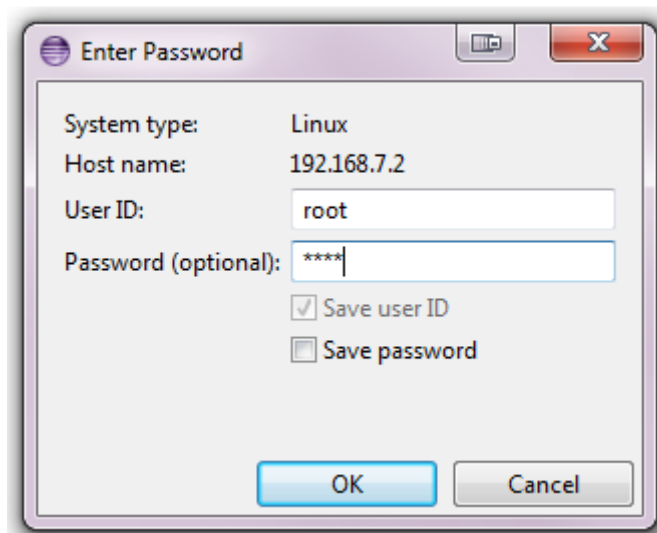
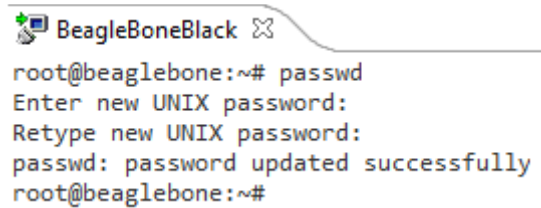


Figura 78 – Conectando à BBB

Por padrão, tanto ID quanto senha da BeagleBone são, respectivamente “root” e “root”. Agora a conexão foi realizada com sucesso e um terminal, igual ao do Cloud9, pode ser aberto no Eclipse. Para isso, dentro de BeagleBoneBlackConnection, é necessário clicar em “Ssh

Terminals” com botão direito e selecionar “Launch Terminal”. Neste terminal é possível compilar e rodar programas feito pelo desenvolvedor, assim como mudar a senha da placa.



```
BeagleBoneBlack
root@beaglebone:~# passwd
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@beaglebone:~#
```

Figura 79 – Escolhendo uma nova senha

Para isto, basta digitar “passwd” no terminal e submeter a entrada, como na Figura 79. Então o usuário deve digitar a nova senha duas vezes. Arquivos podem ser criados, nomeados, programados e executados através deste terminal utilizando os comandos do Linux.

4.3 Anexo III: Cabo JTAG e Code Composer Studio v6

Para usar o JTAG, o seu cabeçalho deve ser soldado na placa. Primeiramente, o bloco P2 deve ser identificado. Ele é uma sequência de 20 contatos organizados em 2 colunas de 10, situado na parte de trás do microcontrolador.

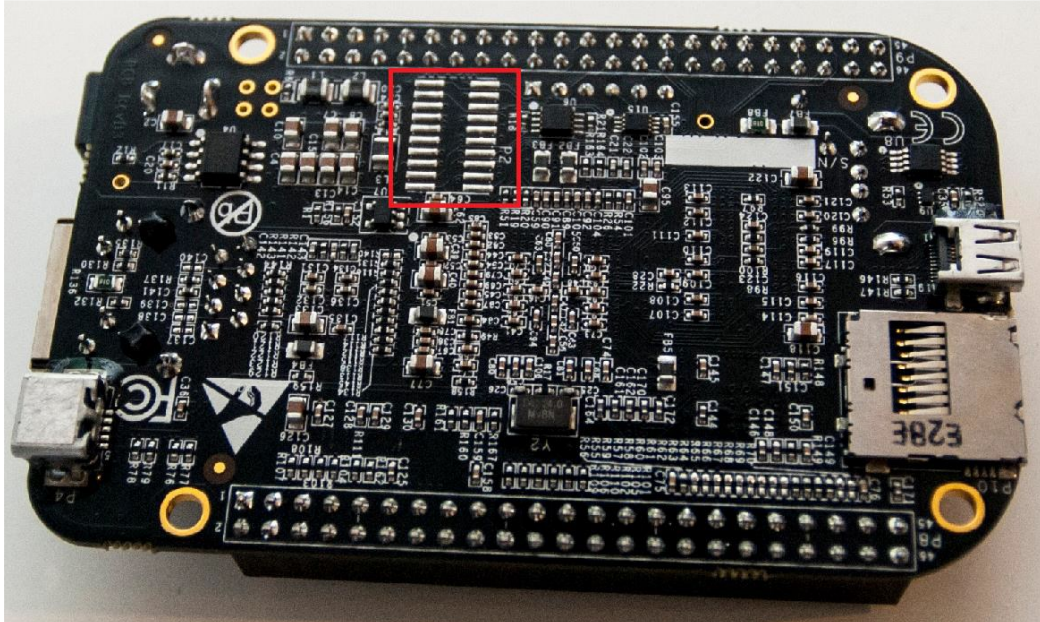


Figura 80 – Bloco P2 destacado na BeagleBone Black [6]

Em seguida, é preciso identificar o contato número 6 no bloco P2. Este contato é chamado de “chave” e servirá como referência

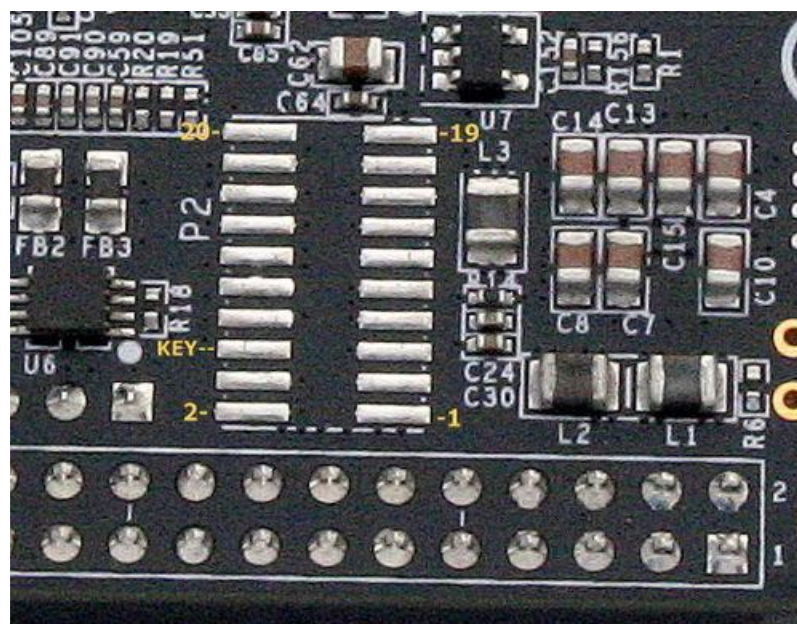


Figura 81 – Contato “chave” [6]

O cabeçalho deve então ser posicionado sobre o bloco P2 de sorte que o contato 6 (o terceiro de baixo para cima da coluna da esquerda, como mostrado na Figura 81) não possua uma conexão equivalente. Esta é posição correta para o cabeçalho.

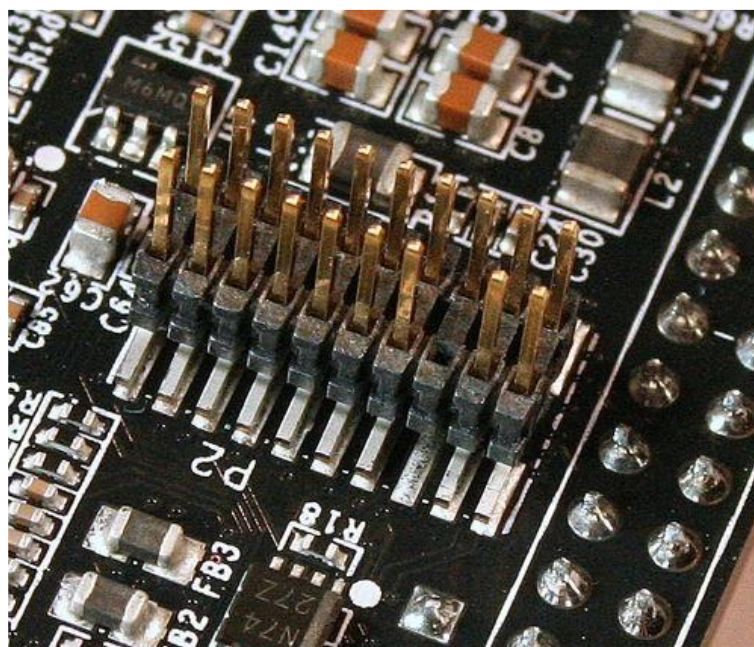


Figura 82 – Cabeçalho posicionado sobre o bloco P2 [6]

Depois de posicionar o cabeçalho, o usuário pode começar a soldá-lo. Este passo deve ser feito com extremo cuidado, para que não haja estanho extra entre os contatos, resultando em um curto-circuito entre as conexões. Além disso, é preciso certificar-se de que a placa está com o estanho completamente seco e afixado antes de conectar o JTAG ao cabeçalho. O resultado final deve se parecer com a Figura 83.

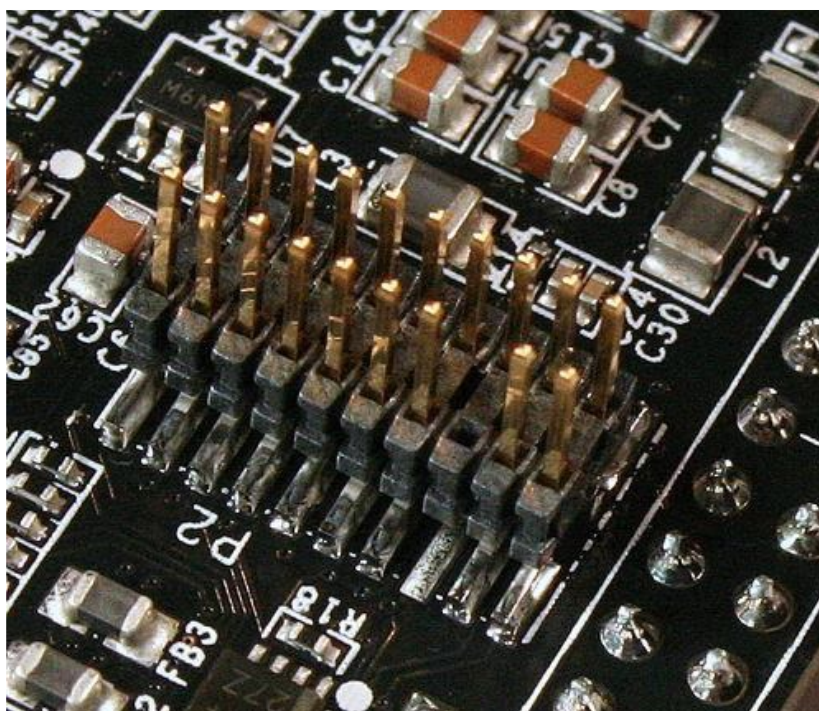


Figura 83 – Cabeçalho soldado à placa [6]

É importante citar que o JTAG não alimenta a BBB com os 5 V necessários para seu funcionamento. Logo, é preciso utilizar uma fonte juntamente com o JTAG para que quaisquer projetos funcionem. Além disso, o JTAG utilizado é, especificamente, o Blackhawk USB100v2 JTAG Emulator, e todas as instruções contidas neste documento são para o uso deste modelo ou semelhante.



Figura 84 – Blackhawk USB100v2 JTAG Emulator

Agora que o hardware está apropriadamente configurado, é necessário instalar o Code Composer Studio versão 6 (CCSv6, ou apenas CCS), um IDE da TI. Este ambiente suporta os microcontroladores da TI e seu portfólio de processadores embarcados. Ele contém um grande conjunto de ferramentas utilizadas para desenvolver e complicar aplicações embarcadas. O CCS

também inclui um compilador C/C++ otimizado, editor de código fonte, um ambiente de construção de projetos, depurador e diversas outras características.

O CCS foi escolhido para ser utilizado porque combina as vantagens do Eclipse em sua estrutura de programa com um depurador avançado de sistemas embarcados da TI, resultando em um ambiente de desenvolvimento atraente para desenvolvedores, graças às suas qualidades.

O IDE pode ser encontrado para baixar em diversas versões em http://processors.wiki.ti.com/index.php/Download_CCS#Code_Composer_Studio_Version_6_Downloads. É essencial, no entanto, que sejam baixadas a versão 6.1.2 ou 6.1.1, para que não haja nenhum problema de incompatibilidade futuro com os passos que ainda estão por vir.

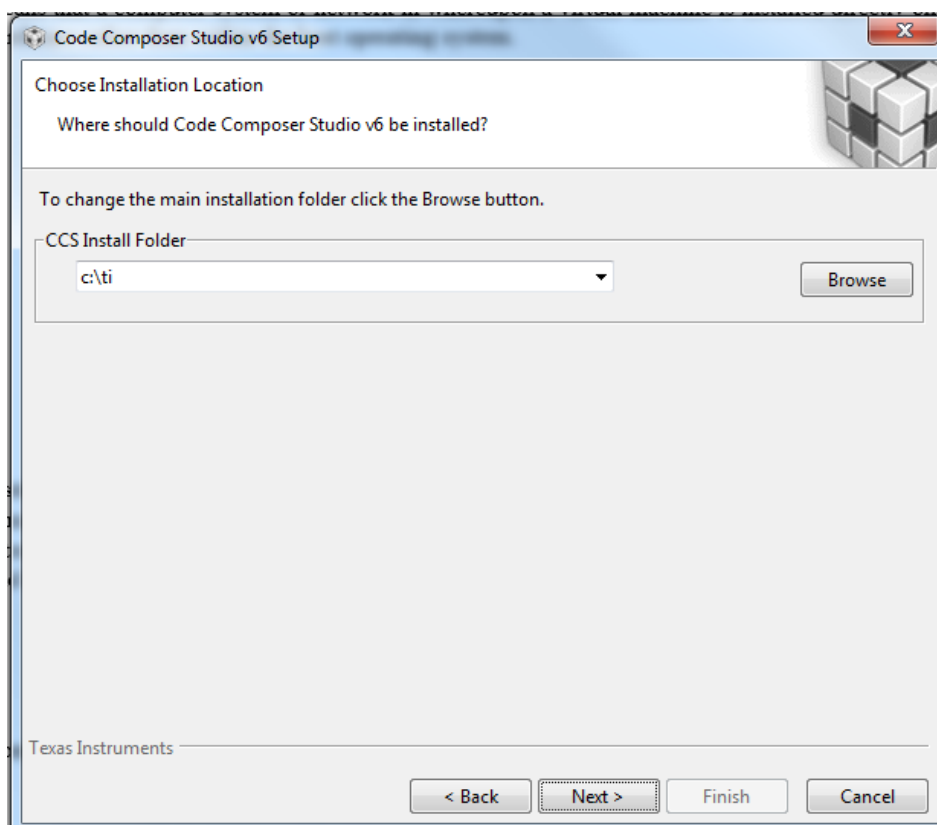


Figura 85 – Instalação do CCS, primeiro passo

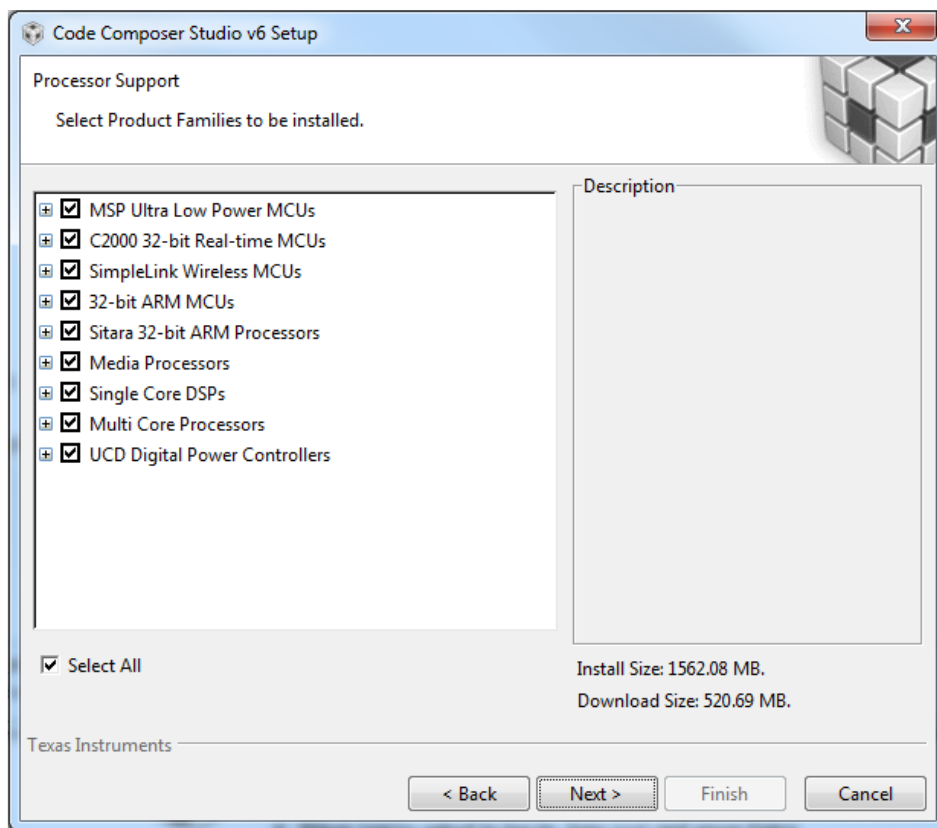


Figura 86 – Instalação do CCS, segundo passo

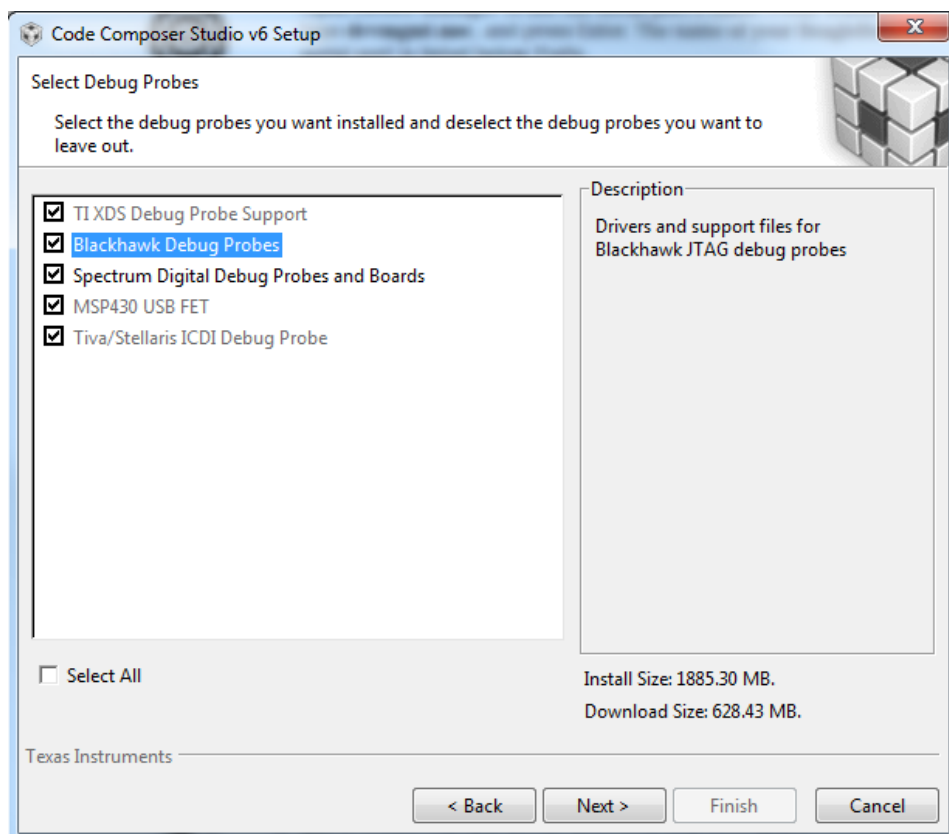


Figura 87 – Instalação do CCS, terceiro passo

Ao terminar a instalação destes recursos, é necessário baixar e instalar também o StarterWare para AM355x (o processador da BBB) também da TI, seguindo o processo de instalação regular. O StarterWare (ou SW) fornece uma plataforma de suporte baseada em C para dispositivos da TI equipados com ARM9 e ARM Cortex A8 (o qual está presente na BBB) sem sistema operacional. Além disso, ele provê bibliotecas de camadas de abstração ao dispositivo e exemplos de programas que fazem uso de periféricos, como Ethernet e manipulação de GPIOs. O endereço onde está disponível para ser baixado é http://software-dl.ti.com/dsps/dsps_public_sw/am_bu/starterware/latest/index_FDS.html. Ademais, neste mesmo endereço, há um arquivo chamado “BeagleBone Black patch”, que também precisa ser baixado. O patch adicionará ao pacote StarterWare a manutenção necessária para uso da BBB. Para aplicá-lo, basta mover o arquivo compactado para a pasta do SW (endereço padrão: C:\ti\AM335X_StarterWare_02_00_01_01) e extraí-lo, respondendo “Yes” sempre que solicitado se o programa deve combinar qualquer pasta ou substituir qualquer arquivo. Em seguida, após abrir o CCS, o usuário deve clicar em “App Center” e procurar pela extensão “TI ARM Compiler”. Ao encontrá-la, deve-se selecionar esta opção e instalar o programa. Ao terminar, o CCS irá reiniciar.

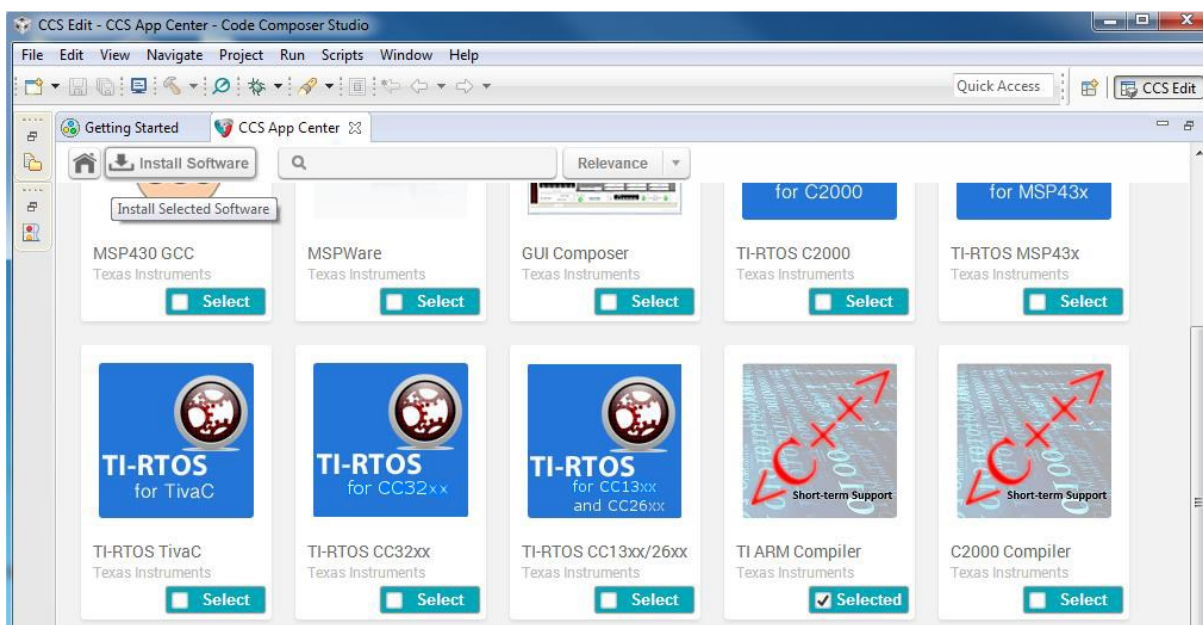


Figura 88 – Instalando extensão de compiladores ao CCS

Agora é essencial conferir se o JTAG está executando a conexão de forma correta depois de conectado ao cabeçalho recentemente soldado à placa. Assim que o CSS abrir novamente, na

aba “View” haverá a opção “Target Configurations”. Uma nova aba surgirá na parte direita da interface do programa.

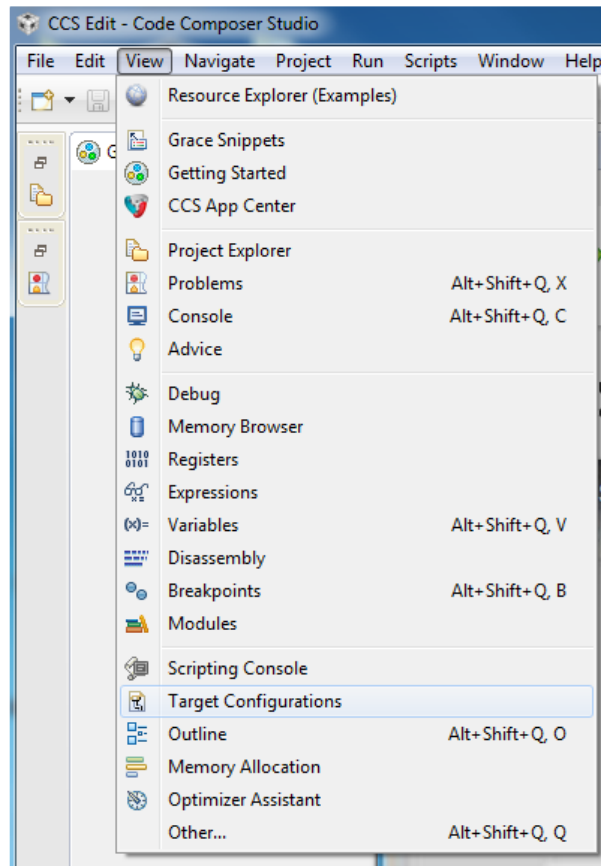


Figura 89 – Criando uma nova conexão, parte 1

Nesta aba é preciso selecionar a opção “New Target Configuration File”.

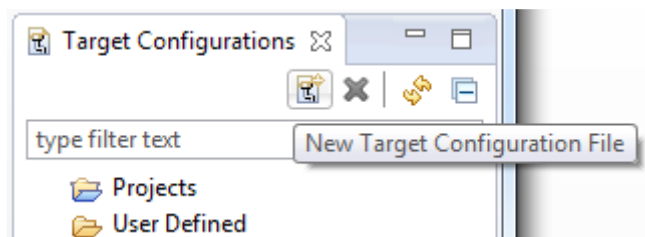


Figura 90 – Criando uma nova conexão, parte 2

O usuário pode escolher o nome que desejar para a conexão. Outra janela irá surgir. Nesta, é necessário encontrar e selecionar o modelo do conector JTAG compatível com o que está sendo usado pelo desenvolvedor. No caso, o XDS100v2 YSB Debug Probe é equivalente ao TMDSEMU100v2U-20T (utilizado neste experimento). Em seguida, no campo “Board or Device”,

deve-se selecionar a opção “BeagleBone_Black”. Assim que estes parâmetros forem estipulados, é preciso salvar a conexão através do botão “Save” à direita.

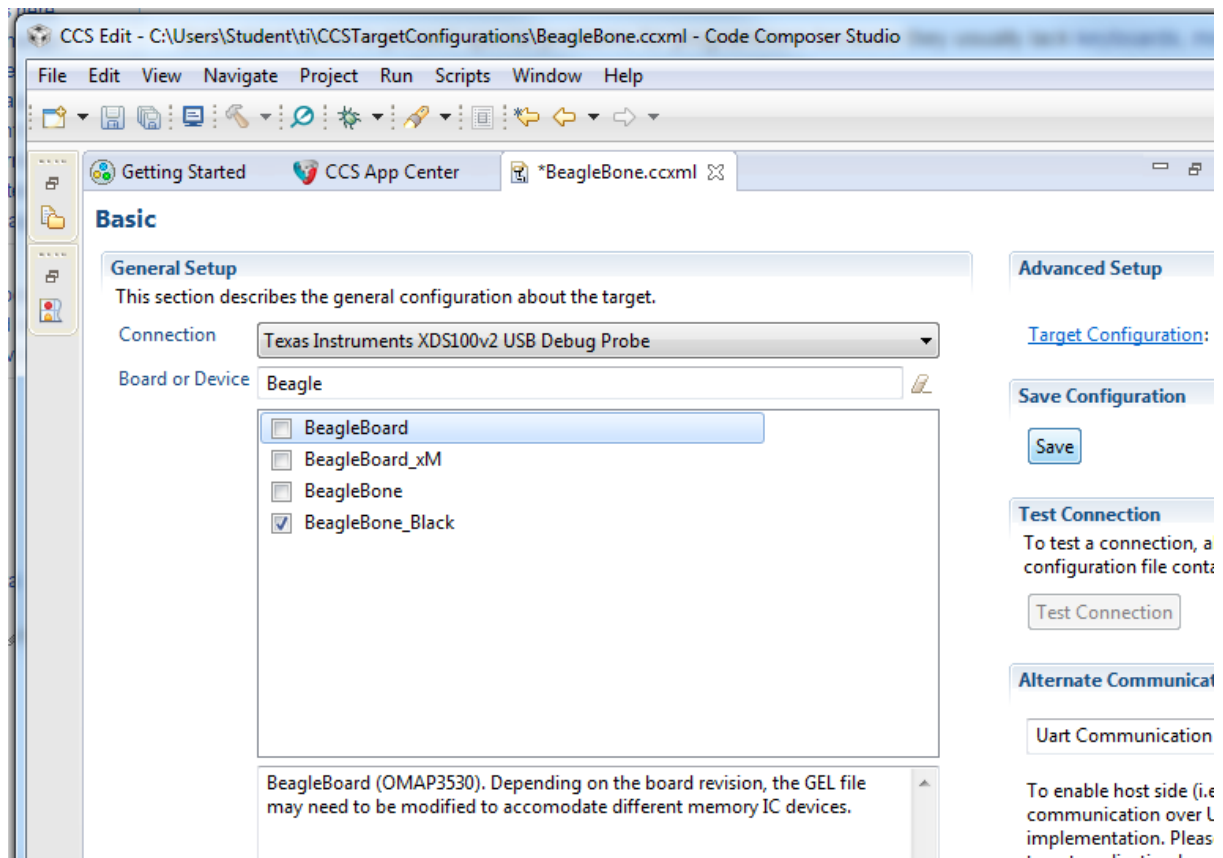


Figura 91 – Configurando a conexão

Para testar a conexão estabelecida, nesta mesma janela há a opção “Test Connection”. Depois de se certificar de que a BeagleBone foi ligada segurando o botão de inicialização (segurar o botão, ligar à fonte de alimentação e soltar o botão após 2 segundos) e o JTAG está encaixado, basta clicar na opção de teste de conexão.

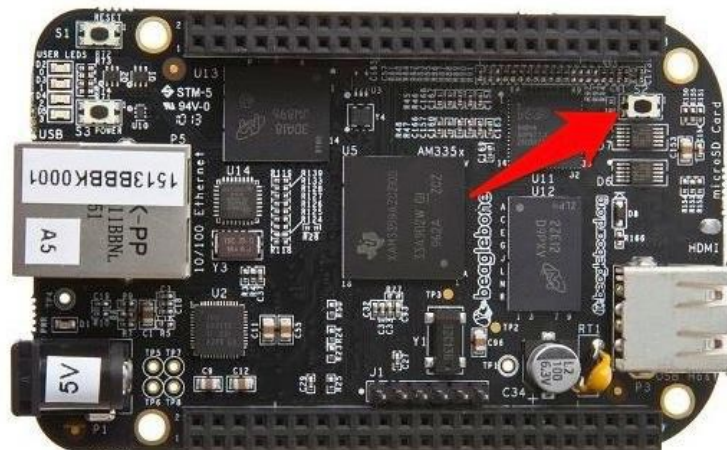


Figura 92 – Botão de inicialização

Caso tudo esteja correto, a mensagem contida na Figura 93 irá aparecer. Caso contrário, uma mensagem de erro irá surgir descrevendo que tipo de problema de conexão foi encontrado. Normalmente esta mensagem muda de 0xFFFFFFFF, que significa que existe algum curto entre um ou diversos conectores do JTAG e o Vcc, ou 0x00000000, que significa que houve uma falha de tensão (mal contanto entre o cabeçalho e o contato) ou um ou mais conectores estão em curto com o terra.

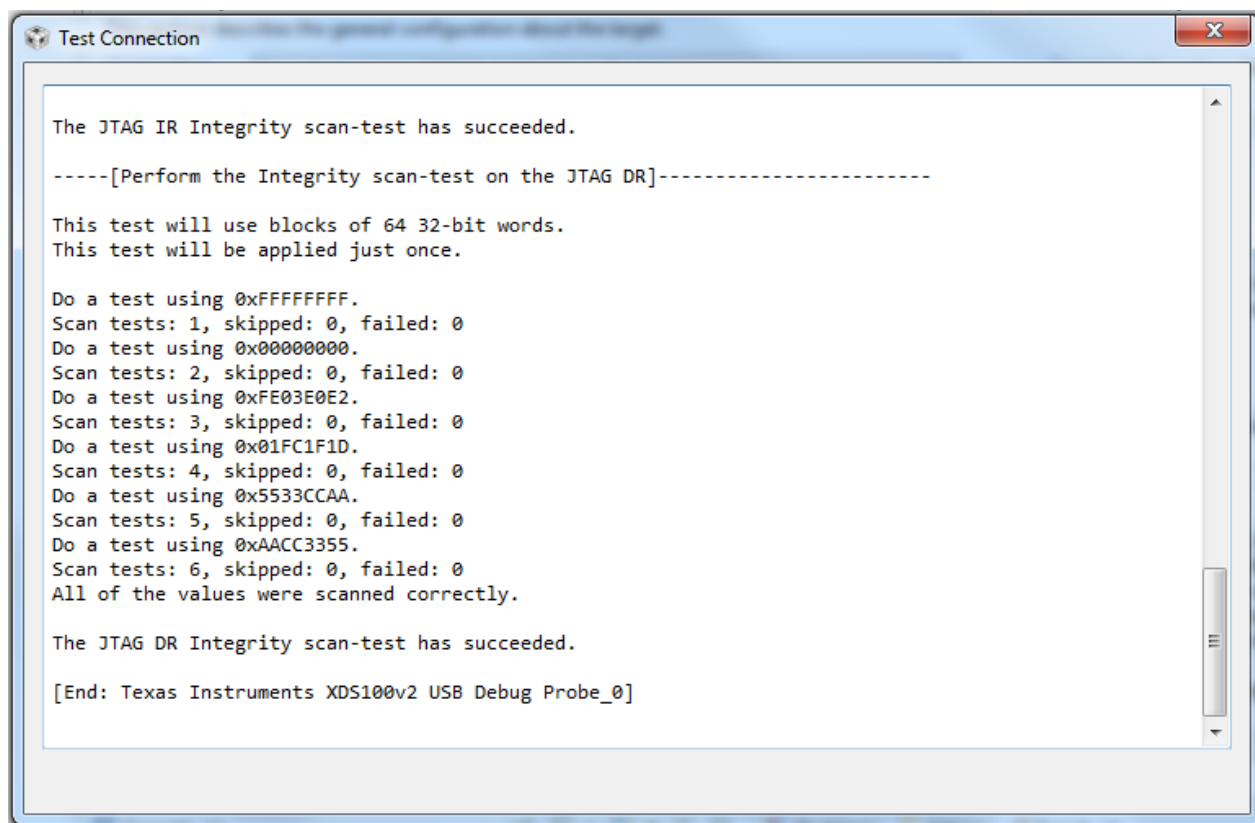


Figura 93 – Conexão realizada com sucesso

De qualquer forma, se o usuário encontrar qualquer um destes erros, a única solução possível é remover a solda do JTAG e repeti-la com mais cuidado e precisão. É uma tarefa trabalhosa e cansativa para pessoas menos experientes, por isso é importante tentar realiza-la com o máximo de calma possível.

Outras mensagens de diferentes erros podem também ser vistas. A maior parte delas está descrita em http://processors.wiki.ti.com/index.php/Debugging_JTAG_Connectivity_Problems.

Além da conectividade, outros possíveis erros devem ocorrer se a versão do CCS instalada difere da 6.1.1 ou 6.1.2 (sendo a última utilizada neste experimento).

Estando tudo correto até o momento, a BeagleBone Black está pronta para funcionar. No entanto, para programar do princípio no CSS é preciso um conhecimento muito avançado em C e certa experiência em como funcionam os processadores ARM9 e ARM Cortex A8. Porém, felizmente, há exemplos dentro do SW bem explicados, que auxiliam o desenvolvedor a obter certas informações essenciais para ser capaz de produzir projetos.

Além disso, começar a programar de um exemplo de base que utiliza as mesmas ferramentas, ou semelhantes, que o projetista pretende usar, poupa bastante tempo. Para acessar estes exemplos, é preciso clicar em “Browse Examples”, na tela principal. Na Figura 94, deve-se

escolher a opção “Configure Resource Explorer to discover examples, documentation and generates a resource package”.

 *Configure Resource Explorer to discover installed examples and documentation.*

- [Configure Resource Explorer to discover examples, documentation and generates a resource package](#) ➤
- [Re-scan user configured packages](#) ➤

Figura 94 – Explorando os exemplos, parte 1

Uma nova janela surgirá e irá pedir para escolher uma configuração de pacote. Ao clicar em “Add”, o usuário deve procurar pela pasta do StarterWare instalada no computador. Ao encontrá-la, basta selecioná-la como opção e clicar em “OK”.

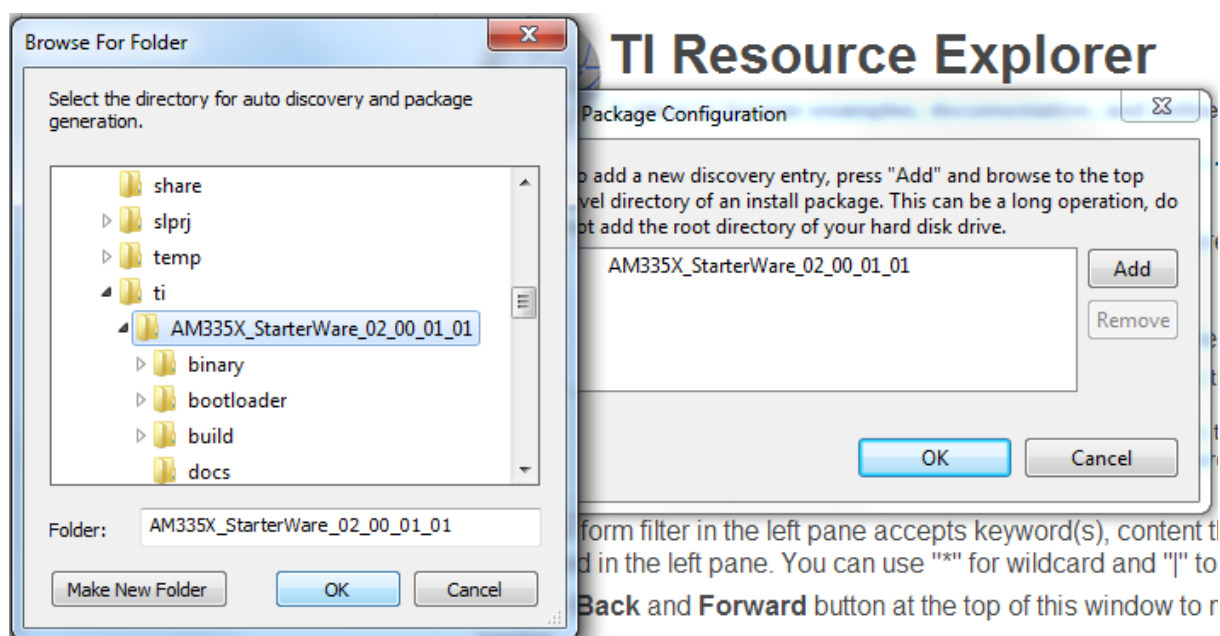


Figura 95 – Explorando os exemplos, parte 2

Os exemplos serão exibidos como opções possíveis de serem importadas. Caso todos sejam selecionados, o CCS ficará como na Figura 96. Os exemplos estão nomeados de forma bem objetiva, tratando-se ao que se referem.

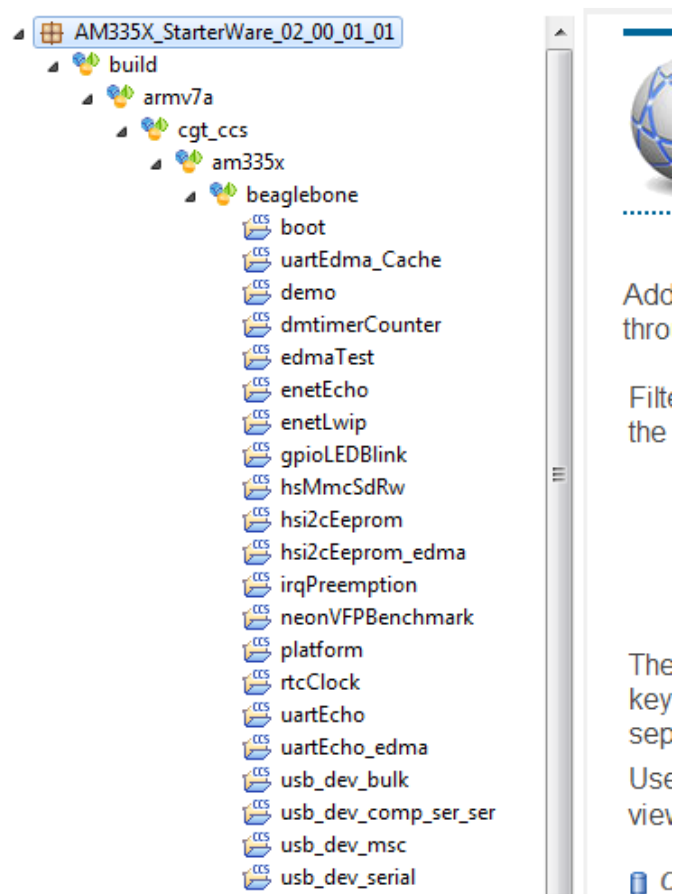


Figura 96 – Exemplos disponíveis

Ao carregar qualquer exemplo de sua escolha, o usuário agora está apto para compilar algum destes exemplos na BeagleBone. No entanto, alguns problemas com relações às bibliotecas podem vir a ocorrer nas primeiras vezes em que são compiladas. Para solucionar estes erros, basta compilar uma biblioteca de cada vez, certificando-se de que estão corretamente configuradas.

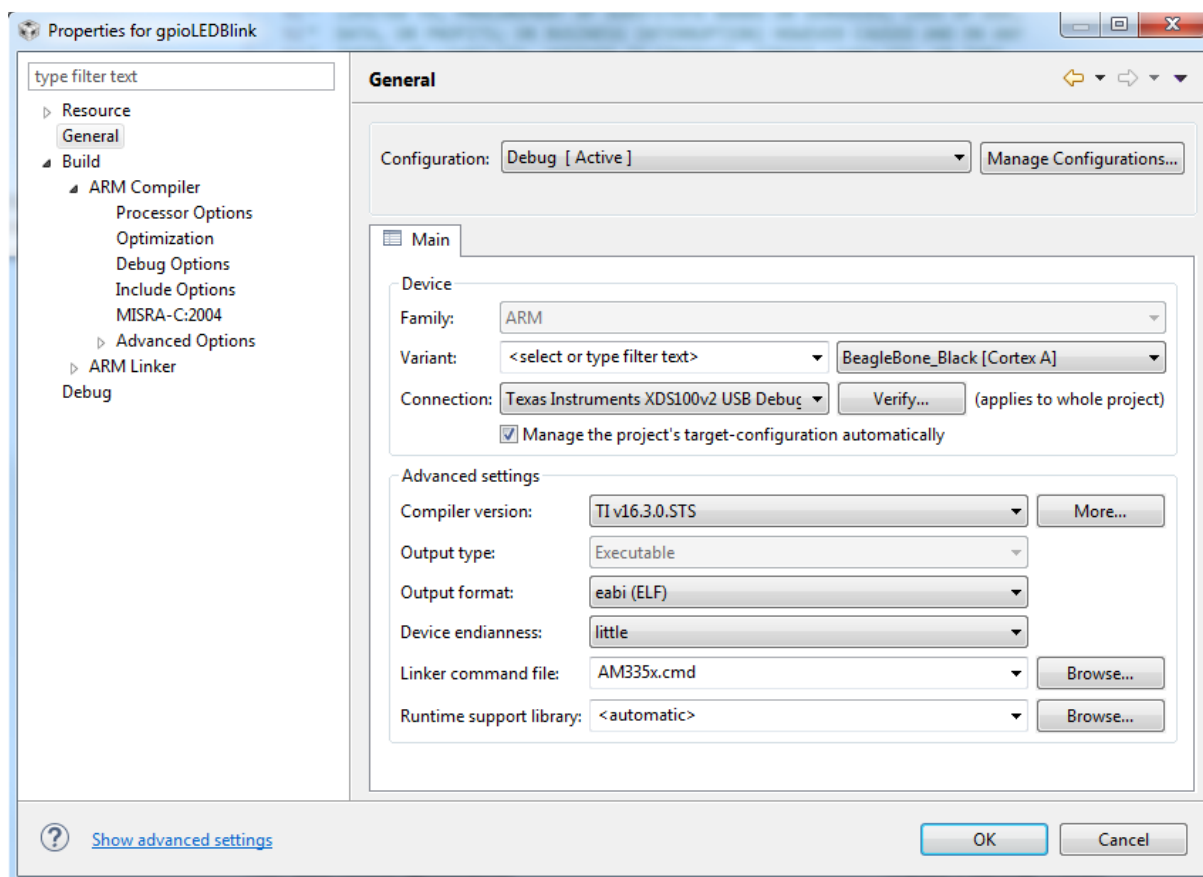


Figura 97 – Configurações para compilar as bibliotecas

Depois que o programa de exemplo for compilado sem problemas no CCS, está na hora de passá-lo para a BBB. Ela deve ser ligada no modo de inicialização citado anteriormente, segurando o botão de boot ao conectá-la na fonte e soltando apenas depois que a mesma for ligada. O cabo JTAG deve estar conectado antes de passar por este processo.

Agora, no CCS, o desenvolvedor deve clicar no botão de depurar (ou debug). Isto fará com que o programa seja carregado para a placa. Uma vez que o programa estiver na placa, o botão “Resume” irá executá-lo (atalho: F8). Para parar esta operação, basta clicar novamente no botão depurar.

Novamente, como o objetivo do trabalho é o estudo de unidades em tempo real, o próximo passo agora é testar a velocidade do BeagleBone Black utilizando a programação em metal puro.

Para isto, um exemplo será utilizado como base. Voltando à Figura 95, depois deste passo é possível escolher os exemplos que o usuário deseja importar, como mostra a seguir a Figura 98.

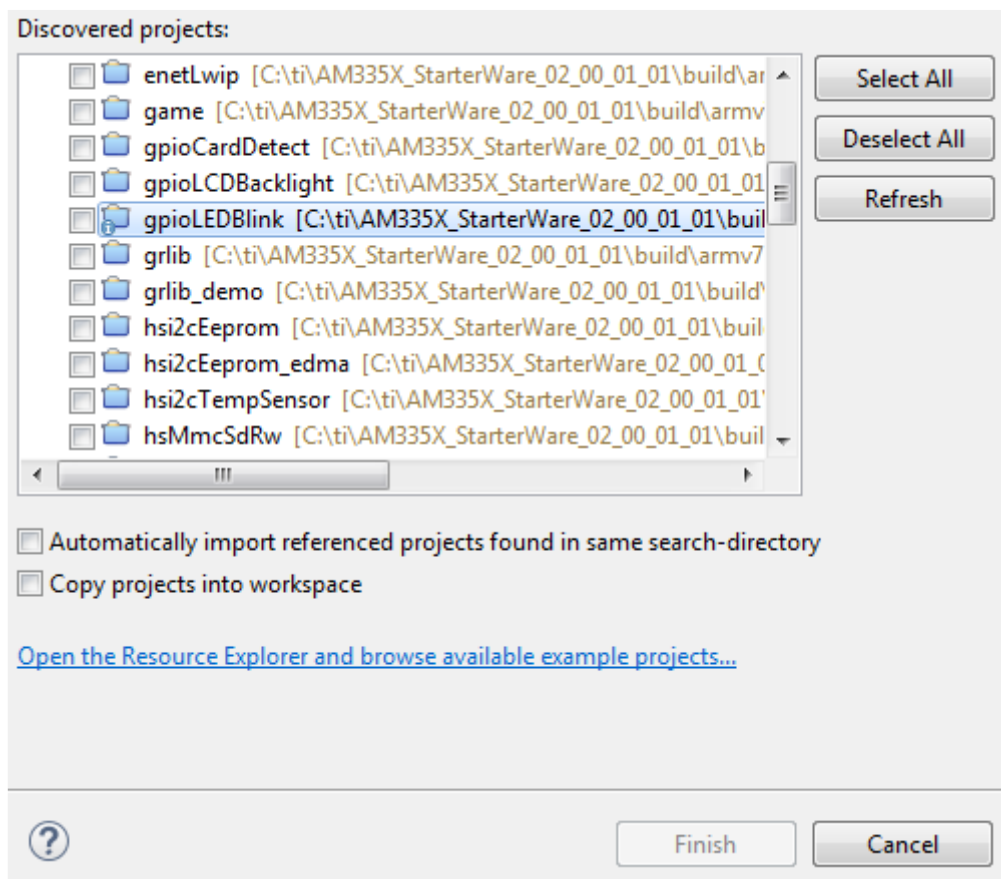


Figura 98 – Exemplo de base

5 REFERÊNCIAS

- [1] RICHARDSON, Matt. Getting Started with BeagleBone: Linux-powered Electronic Projects with Python and JavaScript. Maker Media, Inc., 2013.
- [2] MOLLOY, Derek. Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux. John Wiley & Sons, 2014.
- [3] GRIMMETT, Richard. **Mastering BeagleBone Robotics**. Packt Publishing Ltd, 2014.
- [4] INSTRUMENTS, Texas. AM335x ARM Cortex-A8 Microprocessors (MPUs). **Technical Reference Manual**, 2013.
- [5] BERGH, Paul van Den. **Preparing the Beaglebone Black**. Disponível em: <<http://paulvandenbergh.be/main/index.php/component/content/article?id=5:preparing-the-beaglebone-black>>. Acesso em: 15 maio 2017.
- [6] PARR, Terence. **BeagleBone Black Debugging via TI USB100v2 JTAG Emulator**. Disponível em: <<http://blog.parr.us/2014/11/14/beaglebone-black-debugging-via-ti-usb100v2-jtag-emulator/>>. Acesso em: 16 maio 2017.
- [7] <<https://github.com/derekmolloy/boneDeviceTree/tree/master/gpio>> Acesso em: 25 julho 2017.
- [8] <<https://github.com/derekmolloy/boneDeviceTree/blob/master/gpio/SimpleGPIO.cpp>> Acesso em 25 julho 2017.