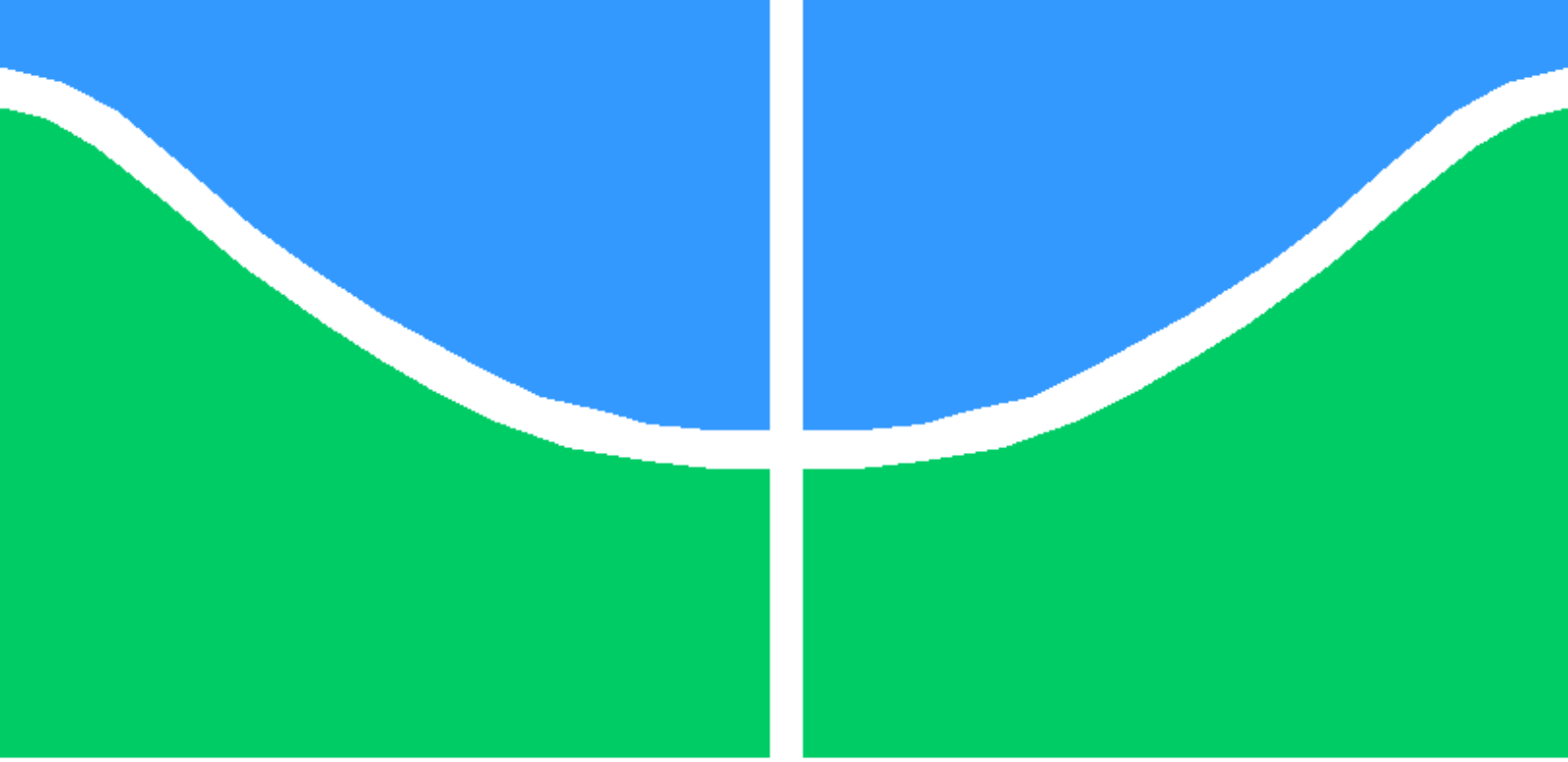


TIMER COM DUPLA BUFERIZAÇÃO PARA HARDWARE DE ACESSO LIVRE: OPENMSP430

GIAN PIETRO TAVARES DE OLIVEIRA PELUCHETTI

TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

TIMER COM DUPLA BUFERIZAÇÃO PARA HARDWARE DE ACESSO LIVRE: OPENMSP430

GIAN PIETRO TAVARES DE OLIVEIRA PELUCHETTI

Orientador: PROF. DANIEL CHAVES CAFÉ, ENE/UNB

TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

BRASÍLIA-DF, 29 DE JUNHO DE 2018.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

TIMER COM DUPLA BUFERIZAÇÃO PARA HARDWARE DE ACESSO LIVRE: OPENMSP430

GIAN PIETRO TAVARES DE OLIVEIRA PELUCHETTI

TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO ACADÊMICO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM ENGENHARIA ELÉTRICA.

APROVADA POR:

Prof. Daniel Chaves Café, ENE/UnB
Orientador

Prof. Stefan Michael Blawid, ENE/UnB
Examinador interno

Prof. José Edil Guimarães de Medeiros, ENE/UnB
Examinador interno

BRASÍLIA, 29 DE JUNHO DE 2018.

FICHA CATALOGRÁFICA
GIAN PIETRO TAVARES DE OLIVEIRA PELUCHETTI
Timer com dupla buferização para hardware de acesso livre: openMSP430
2018xv, 147p., 201x297 mm
(ENE/FT/UnB, Bacharel, Engenharia Elétrica, 2018)
Trabalho de Conclusão de Curso de Graduação - Universidade de Brasília
Faculdade de Tecnologia - Departamento de Engenharia Elétrica

REFERÊNCIA BIBLIOGRÁFICA

GIAN PIETRO TAVARES DE OLIVEIRA PELUCHETTI (2018) Timer com dupla buferização para hardware de acesso livre: openMSP430. Trabalho de Conclusão de Curso de Graduação em Engenharia Elétrica, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 147p.

CESSÃO DE DIREITOS

AUTOR: Gian Pietro Tavares de Oliveira Peluchetti
TÍTULO: Timer com dupla buferização para hardware de acesso livre: openMSP430.

_____

Gian Pietro Tavares de Oliveira Peluchetti

O objetivo deste trabalho é escolher um microcontrolador de acesso aberto existente e apropriado para o LDCI. Foi escolhido o openMSP430 por possuir uma arquitetura bem estabelecida, testada e dominada pelos membros do laboratório. Além disso, é desenvolvido os códigos HDL e validação para um periférico compatível com as instruções do periférico Timer B da Texas Instruments para demonstrar a integração no barramento do sistema para que esse processo possa ser repetido para os outros IPs do laboratório, além de ser um periférico com características interessantes para criação de ondas PWM.

Palavras-chaves: Eletrônica, Design digital, Microcontroladores

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS FONTE

LISTA DE TERMOS E SIGLAS

ARM	Advanced RISC Machine
DMA	Direct-Memory-Access
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IP	Intellectual Property
LDCI	Laboratório de Disposivos e Circuitos Integrados
MIPS	Microprocessor without interlocked pipeline stages
PWM	Pulse Width Modulation
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
TI	Texas Instruments
UART	Universal asynchronous receiver/transmitter

Capítulo 1

Introdução

Segundo [Crisp 2003]:

> Em 1971 duas companhias, ambas nos EUA, introduziram o mundo ao seu futuro produzindo microprocessadores. Elas eram jovens companhias chamadas *Intel*, e sua rival, *Texas Instruments* (TI) . Desde então os microprocessadores, e sua "prole"os microcontroladores, invadiram todo o mundo nas mais diversas áreas desde indústrias, agropecuária e mesmo dentro de casas.

Simplificadamente o microprocessador pode ser entendido como um sistema caixa preta. O sistema é alimentado com uma entrada, essa entrada é processada dentro do microprocessador gerando uma saída.

Figura 1.1: Esquema de um sistema caixa preta(Fonte: [Taylor 2018])

O microcontrolador é um sistema que integra um processador e um conjunto de periféricos se tornando uma solução conveniente, compacta e eficiente para muitas aplicações. Em especial, na indústria de circuitos integrados, os blocos de propriedade intelectual (IPs) geralmente são vendidos acompanhados de um circuito digital de controle para permitir o uso e teste desses IPs. Conversores A/D, oxímetros, e diversos tipos de sensores são produzidos em conjunto com um circuito digital que fornece uma interface de comunicação permitindo a fácil manipulação dos IPs.

O Laboratório de Dispositivos e Circuitos Integrados (LDCI) da Universidade de Brasília (UnB) produz diversos tipos de IPs analógicos e digitais para fins acadêmicos. O IP mais recente projetado foi um conversor SAR de 12-bits e 1Msps, desenvolvido pelos alunos Iago Pereira e Pedro Tolentino [Pereira 2018] [Tolentino 2018]. O uso desse IP ainda é baseado no controle manual de seus terminais de controle, o que, de um ponto de vista de uso e teste deste dispositivo, é bastante inconveniente pois exige um número grande de conexões e equipamentos específicos. Seria mais interessante encapsular o dispositivo dentro de um framework digital de um microcontrolador, acoplando o IP no barramento e permitindo o controle por um processador digital ou uma máquina de estados dedicada.

Por esse motivo, é interessante para o LDCI, possuir um microcontrolador próprio para os seus projetos, já que o LDCI trabalha principalmente com projetos de sensores e atuadores.

Assim, pelo site opencores.org, uma comunidade de desenvolvimento de IPs de acesso livre, foi possível encontrar opções de implementações de microcontroladores como Amber ARM, NEO430, Plasma (MIPS), OpenRISC, openMSP430 que se mostram como opções viáveis para o LDCI.

Objetivo Geral

O objetivo deste trabalho é escolher um microcontrolador apropriado para o LDCI, para fazer deste microcontrolador a base para a disponibilização dos demais IPs do laboratório.

Um periférico, integrável ao barramento do sistema deve ser desenvolvido para que esse processo possa ser repetido para os outros IPs do laboratório.

Objetivos específicos

Para alcançar os objetivos descritos acima, se faz necessário:

- A escolha de uma arquitetura de interesse
- Desenvolvimento de um periférico
- A elaboração da idéia um periférico inspirado nas funções do *Timer B*
- Elaboração de um código HDL em *Verilog*
- Escrita dos códigos de validação
- Validação do código um periférico capaz de gerar ondas PWM sem *glitches*

Organização do documento

No capítulo 2, é apresentada a revisão dos microcontroladores de acesso livre e a escolha e alguns ponto importantes para o entendimento do trabalho.

No capítulo 3, é explicada a metodologia usada para o fluxo digital e os testes automáticos feitos para validar o comportamento do periférico

No capítulo 4, são mostrados os resultados obtidos no trabalho.

E por fim, no capítulo 5 é apresentada as conclusões e ideias de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Revisão dos microcontroladores de acesso aberto

Existem várias implementações disponíveis de microcontroladores. Foi feita uma revisão das implementações destacadas do site opencores.org. São elas: Amber ARM, NEO430, Plasma (MIPS), openMSP430.

O Amber ARM, [Santifort 2010], é um processador RISC de 32 bits, compatível com as instruções v2a ARM da empresa britânica ARM Holdings. Esse projeto fornece os códigos completos para implementar o processador e alguns periféricos como, um timer, ponto de acesso ethernet e módulo de comunicação serial UART. O projeto foi desenvolvido em verilog e otimizados para síntese em FPGAs. É usada a velha versão v2a pelo fato de ela ser de livre acesso, assim o projeto pode ser implementado sem uma licença da ARM Holdings.

O NEO430, [Nolting 2015], é um processador de 16 bits, compatível com as instruções da MSP430 da TI. Esse projeto fornece os códigos para implementação do processador junto de alguns periféricos como, um timer, watchdog, um módulo de comunicação serial UART, sortas de saída e entrada digitais. Esse processador apresenta como principais diferenças para o MSP430 a presença de apenas um modo de baixo consumo, número reduzido de canais de interrupção e módulos de processamento com funcionalidades diferentes.

O Plasma (MIPS), [Rhoads 2010], é um processador de 32 bits de arquitetura RISC compatível com as instruções MIPS desenvolvida pela MIPS Computer Systems. Seu código é feito em VHDL, possui alguns periféricos como interface flash, ponto de acesso ethernet e timer.

Foi escolhida, após revisão dos microcontroladores de acesso aberto, a implementação do *openMSP430*. O *openMSP430*, [Girard 2009], é um IP desenvolvido por Olivier Girard, que apresenta um microcontrolador sintetizável de 16 bits de arquitetura quase-RISC totalmente compatível com o conjunto de instruções do MSP430 da TI. Deste modo, o *openMSP430* é uma solução interessante, pois possui uma arquitetura bem estabelecida, testada e dominada pelos membros do laboratório.

Uma outra grande vantagem é o fato de Olivier ter disponibilizado o *openMSP430* com a licença *GNU Lesser General Public License*, que possibilita que o código seja modificado, ou usado de forma irrestrita para esse trabalho.

Observando, o código do openMSP430 foi observada a falta do Timer B. Este periférico é um contador programável de 8 ,10, 12 ou 16-bits com múltiplos registros de captura/comparação. A grande vantagem que ele apresenta em relação ao Timer A (que já está implementado no código de Olivier) é a dupla buferização do comparador que o torna ideal para implementações que precisem de sinais modulados por largura de onda (PWM). Aproveitando a ausência desse periférico, iremos implementar o Timer B com o intúito de demonstrar o acoplamento de um periférico qualquer no barramento do microcontrolador.

openMSP430

O *openMSP430* é um microcontrolador de 16 *bits* sintetizável compatível com a família MSP430 da TI podendo executar códigos gerados para o microcontrolador da TI de forma quase idêntica.

As principais características do *openMSP430* são sua pequena área, baixo consumo, alta densidade de código, boa performance, opções de gerenciamento de *clock* e alimentação integradas. A documentação oficial da Texas Instrumentes, família MSP430x1xx é aplicável quase que por inteira ao openMSP430, tendo apenas pequenas mudanças no tamanho e ciclos das instruções do processador. Ele apresenta a limitação de não poder executar códigos direto da RAM.

O microcontrolador também apresenta uma interface DMA que possibilita a cópia de dados de memória sem a interferência do CPU e uma interface de debug serial por 2 fios podendo usar tanto os protocolos UART quanto I^2C.

Além disso, possui 6 periféricos ligados ao núcleo mostrados na tabela 2.1, e possui 2 moldes para criação de periféricos de 8 e 16 bits.

2.2 Timer B

Esta seção detalha as especificações principais do Timer B, retiradas do User's Guide da TI [Texas Instruments 2009]. Estas especificações serão usadas para a implementação detalhada no capítulo 3.

O timer B foi escolhido por gerar sinais PWM, sem riscos de *glitches*. Tais *glitches* podem acontecer no timer A pois o processador tem pouco tempo para atualizar o valor do comparador quando o mesmo chega próximo de zero. Quando o processador não consegue responder a tempo, ele perde um ciclo da PWM e gera um pulso 100% positivo num momento que o duty cycle deveria ficar próximo de 0%. Dependendo da aplicação isso pode

Periférico	Descrição
Módulo de *clock* básico	É o periférico que gerencia os *clocks* usados pelo microprocessador.
Special Function Registers (SFR)	É o periférico que implementa *flags* e interrupções.
Timer Watchdog	É um *timer* que tem como função principal reiniciar o sistema em caso de um problema de software.
Multiplicador de hardware 16x16	É um periférico especializado em multiplicações de palavras de até 16 bits totalmente separado do CPU.
GPIO	São o grupo de entradas e saídas digitais do microprocessador.
Timer A	Timer/contador de 16 bits com 3 canais de captura e comparação.

Tabela 2.1: Periféricos

ser um problema. O timer B resolve esse problema pois o processador sempre trabalha um ciclo atrasado, tendo bastante tempo para atualizar o valor do comparador, já que o latch de comparação só é atualizado no final da contagem (quando chega em CCR0).

Timer B é uma nomenclatura criada pela TI, para definir um temporizador com dupla buferização e tamanho do contador programável. O diagrama do Timer B pode ser visto na figura 2.1.

Registro do Timer B (TBR)

O TBR é um registro de contagem de 16 bits que incrementa ou decrementa o seu valor em 1, dependendo do modo de operação, a cada flanco de subida do sinal de *clock*. Ele pode ser lido e escrito por software e gera interrupções quando ocorre *overflow*

O tamanho de saída do contador é configurável para 16, 12, 10 ou 8 bits. Sendo o bit menos significativo sempre escrito na direita do registro. O seu *clock* pode ser fornecido pelo *ACLK*, *SMCLK*, que são clocks internos do microprocessador ou pelo *TBCLK* que pode ser fornecido externamente. Estas fontes de *clock* podem ser repassadas diretamente para o *timer* ou divididas em 2, 4 ou 8 vezes.

Modos de Contagem

Existem 4 tipos diferentes de modos de contagem apresentados na tabela , definidos pelos bits MCx

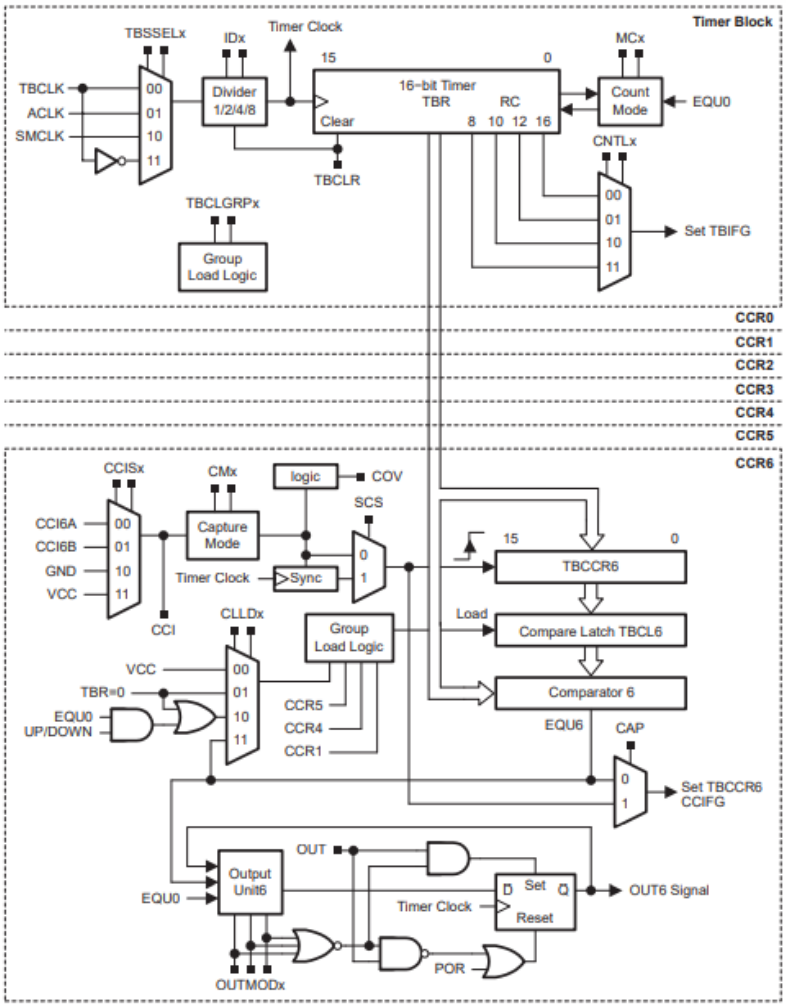


Figura 2.1: Diagrama Timer B(fonte : [Texas Instruments 2009])

MCx	Modo	Descrição
00	Stop	Timer parado
01	Up	Conta continuamente de 0 até o valor de TBCL0
10	Continous	Conta continuamente de 0 até o valor máximo de bits escolhido por CNTLx
11	Up-Down	Conta continuamente de 0 até o valor de TBCL0 e depois volta até 0

Tabela 2.2: Modos de Contagem

Modo *Continuous*

No modo *continuous* o *timer* conta repetidamente de 0 ao valor máximo de TBR, da forma apresentada na figura 2.2.

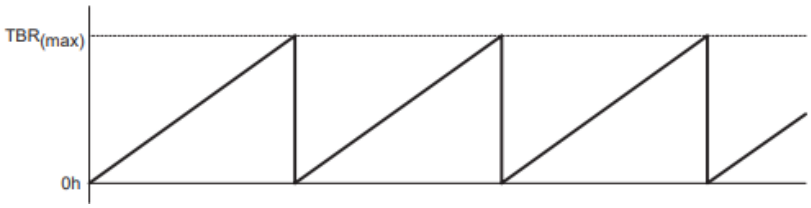


Figura 2.2: Modo *Continuous*(fonte : [Texas Instruments 2009])

Modo *Up*

O modo *up* é usado quando se quer definir um período diferente do valor máximo de TBR. O *timer* conta repetidamente de 0 até o valor do *latch* de comparação TBCL0, definindo assim o período como TBCL0+1 como mostrado na figura 2.3.

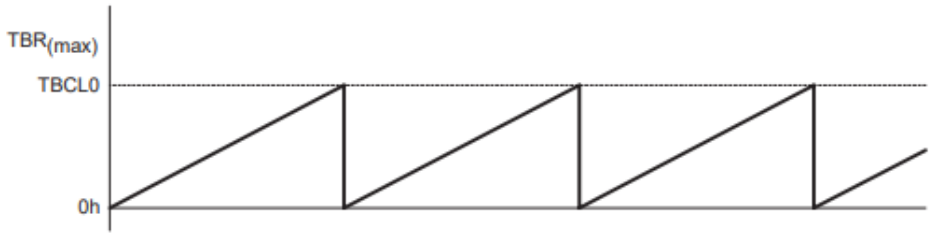


Figura 2.3: Modo *Up*(fonte : [Texas Instruments 2009])

Modo *Up/Down*

O modo *up/down* também é usado quando se quer definir um período diferente do valor máximo de TBR e uma geração de pulso simétrico é precisa. O *timer* conta repetidamente de 0 até o valor do *latch* de comparação TBCL0, e depois conta de forma contrária até 0, fazendo o período ser o dobro de TBCL0 mostrado na figura 2.4. A direção de contagem é salva, assim se o *timer* for parado e depois reiniciado ele contará na mesma direção que estava, é preciso zerar o *clock* ativando o bit TBCLR para zerar a direção.

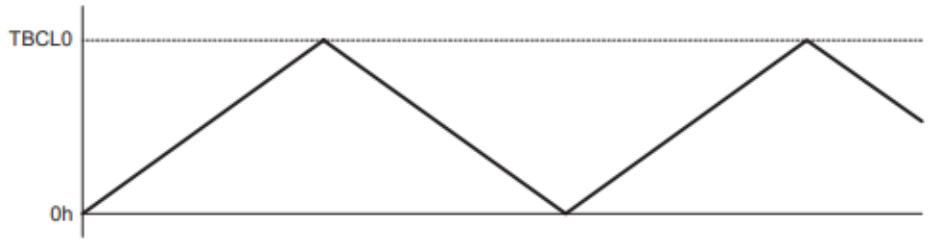


Figura 2.4: Modo *Up/Down*(fonte : [Texas Instruments 2009])

Modo de Captura/Comparação

O *timer* B, pode vir com 3 ou 6 blocos de captura/comparação. Qualquer uma desses blocos pode ser usado para capturar o tempo ou gerar intervalos.

Modo de Captura

O modo de captura é ativado quando o *bit* CAP = 1, no registro TBCCTLx. Este modo serve para guardar o tempo de eventos podendo ser usado para computações rápidas e medidas de tempo. Cada canal possui duas entradas conectadas a pinos externos CCIxA e CCIxB selecionáveis pela configuração do registro. Pode ser capturada o flanco de subida, descida ou ambos dos sinais. Quando a captura ocorre, o valor do tempo em TBR é copiado para o registro TBCCRx e a *flag* de interrupção CCIFG é ativada. Caso uma segunda captura ocorra antes que o valor da primeira captura seja lido o bit COV é ativado. O ciclo de captura é apresentado na figura 2.5.

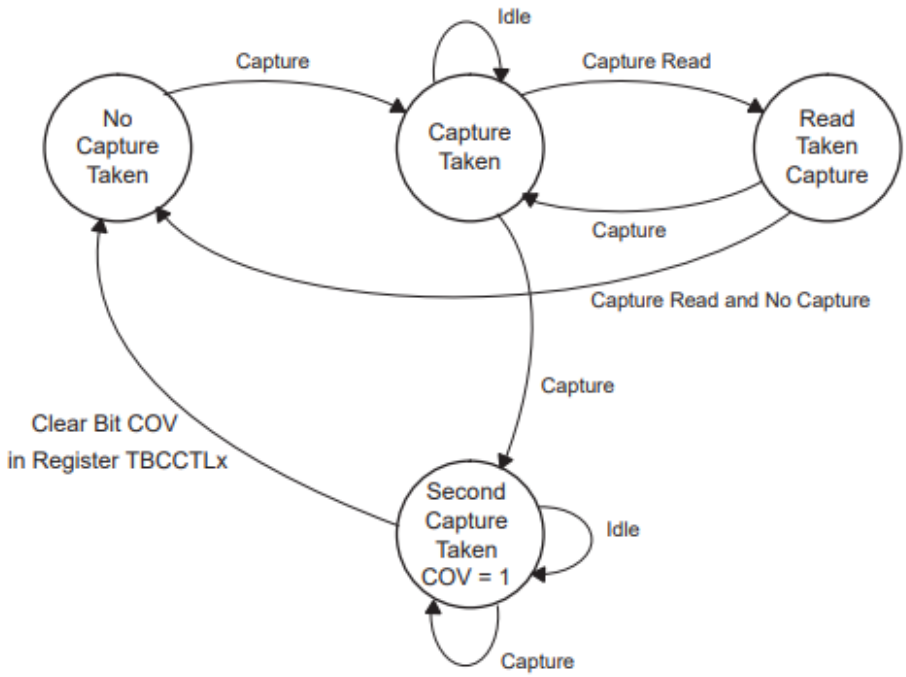


Figura 2.5: Ciclo de Captura (fonte : [Texas Instruments 2009])

Modo de Comparação

O modo de comparação é ativado quando o *bit* CAP = 0, no registro TBCCTLx. Este modo é usado para criar gerar saídas PWM, ou interrupções em tempos específicos. Quando TBR chega no valor de TBCLx, são ativadas a *flah* de interrupção CCIFG e o sinal interno EQUx, dependendo do modo de saída escolhido EQUx afeta a saída de forma diferente.

Latch de comparação TBCLx

O latch de comparação, TBCLx, guarda o valor de comparação no modo de comparação. Ele é alimentado pelo registrador TBCCRx, não é possível acessar diretamente o valor de TBCLx, o tempo em que o valor de TBCLx será atualizado para o valor de TBCCRx é escolhido pelos bits de controle CLLDx, o tempo da transferência é apresentado na tabela 2.3.

CLLDx	Descrição
00	O valor de TBCLx é atualizado imediatamente quando é escrito em TBCCRx
01	O valor de TBCLx é atualizado assim que TBR seja igual a 0
10	O valor de TBCLx é atualizado assim que TBR seja igual a 0 nos modos *up* e *continous* ou igual ao velho TBCL0 ou 0 no modo *up/down*
11	O valor de TBCLx é atualizado assim que TBR seja igual ao valor antigo de TBCLx

Tabela 2.3: Atualização de TBCLx

Unidade de saída

Cada bloco de captura/comparação possui uma unidade de saída. Essa unidade pode ser usada para gerar sinais de saída como sinais PWM. Cada unidade de saída possui 8 modos de operação diferentes baseados nos sinais internos EQU0 e EQUx. Os modos de saída são definidos na tabela 2.4.

OUTMODx	Modo	Descrição
000	Output	A saída é definida pelo bit OUTx.
001	Set	Quando TBR atinge o valor de TBCLx a saída fica no valor 1 e assim fica até o timer ser resetado ou o modo de saída trocado
010	Toggle/Reset	O valor da saída é alternado quando TBR atinge TBCLx e vai para 0 quando TBR chega a TBCL0.
011	Set/Reset	A saída vai para 1 quando TBR atinge TBCLx e 0 quando TBR atinge TBCL0.
100	Toggle	O valor da saída é alternado toda vez que TBR atinge TBCLx
101	Reset	Quando TBR atinge o valor de TBCLx a saída fica no valor 0 e assim fica até o timer modo de saída trocado
110	Toggle/Set	O valor da saída é alternado quando TBR atinge TBCLx e vai para 1 quando quando TBR chega a TBCL0.
111	Reset/Set	A saída vai para 0 quando TBR atinge TBCLx e 1 quando TBR atinge TBCL0.

Tabela 2.4: Modos de Saída

Dupla buferização

É chamada de dupla buferização o acréscimo de um latch de comparação que não existe no Timer A. Esse latch possibilita que sejam produzidos sinais PWM sem o risco de ocorrer glitches. No caso do timer A, o usuário tem acesso direto ao registro que controla quando o timer muda o valor do sinal para criar a PWM, esse controle pode levar a erros. Se por exemplo, o sinal sempre troca de posição (função de saída *TOGGLE*), no tempo 10 e o contador sempre conta de 0 a 20, a onda da forma será igual a da figura 2.6.

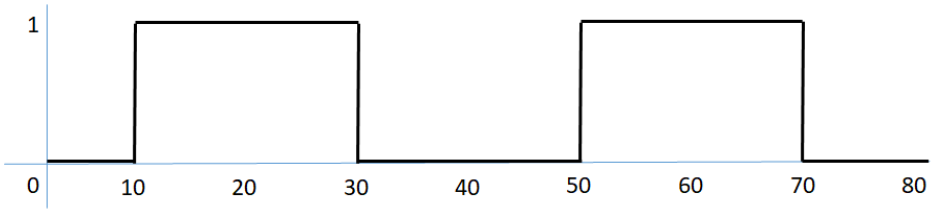


Figura 2.6: Exemplo Onda 1

Ao tentar mudar esse valor da troca para 5 existem duas situações, a primeira o valor da troca é feito no tempo 40, como o contador conta de 0 a 20 ele estaria no 0, e a outra o contador está em 48. As duas situações estão mostradas nas figuras 2.7 e 2.8

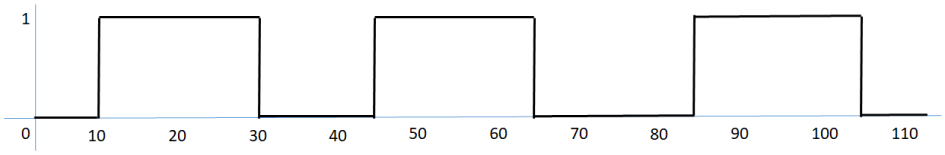


Figura 2.7: Exemplo Onda 2

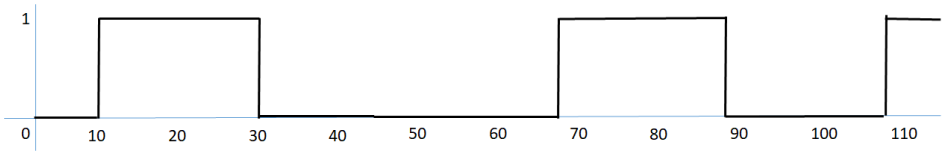


Figura 2.8: Exemplo Onda 3

Como é possível ver na figura 2.8, como o contador já passou pelo ponto de mudança e ainda não tinha chegado no próximo um glitch ocorre é perdido todo um período da onda, o que pode ser prejudicial dependendo a aplicação em que se está usando essa onda. Portanto, a dupla buferização conserta esse erro pelo fato de o usuário não possuir acesso direto a esse registro de troca e a atualização desse registro é feita por um latch. Assim, como explicado na seção anterior é possível escolher o momento em que esse registro é atualizado para que dessa forma não ocorra o glitch demonstrado aqui.

Capítulo 3

Metodologia

3.1 Desenvolvimento Top-Bottom

Os avanços tecnológicos das últimas décadas possibilitaram o aumento da densidade de transistores em circuitos integrados digitais. Esse aumento tornou praticamente impossível o desenvolvimento de projetos de circuitos digitais pelo método tradicional manual e baseado em esquemáticos. Uma forma mais automatizada se fez necessária e diante disso a indústria optou pelo uso de ferramentas de síntese automática de circuitos baseadas em descrições do hardware em linguagens de alto nível. Isso possibilitou o uso de abordagens Top-Bottom para a fabricação de circuitos integrados.

Um verdadeiro projeto Top-Bottom, num mundo ideal segundo [Smith 1998], seria poder descrever completamente um sistema de forma abstrata através de uma linguagem HDL e sintetizá-lo numa máscara pronta para fabricação de forma totalmente automática. Isso ainda não é possível, porém as ferramentas profissionais usadas para síntese de circuitos digitais estão melhorando cada vez mais em direção a esse objetivo.

Deste modo, a metodologia *Top-Bottom*, como apresentada na figura 3.1 consiste em elaborar primeiramente o conceito do sistema que deseja ser implementado, pensando em suas especificações e funcionamento esperado. Em seguida, detalha-se o comportamento de cada parte do circuito, e assim vai descendo a pirâmide cada vez mais aumentando o detalhamento do sistema até chegar no comportamento do transistor de cada porta lógica.

Essa metodologia não implica necessariamente num aumento de complexidade de desenvolvimento, já que as camadas de abstração são desenvolvidas separadamente e na indústria de semicondutores, essa separação chega a ser a nível de diferentes empresas. Enquanto o projetista digital se concentra na camada de abstração mais elevada, existem outros players no mercado que se concentram em desenvolver blocos elementares, como transistores de baixo consumo, portas lógicas de alta velocidade, etc. Uma outra parte do mercado se concentra na produção das ferramentas que traduzem o que foi desenvolvido em uma camada de abstração na outra. Essas ferramentas são chamadas de sintetizadores por juntar infor-

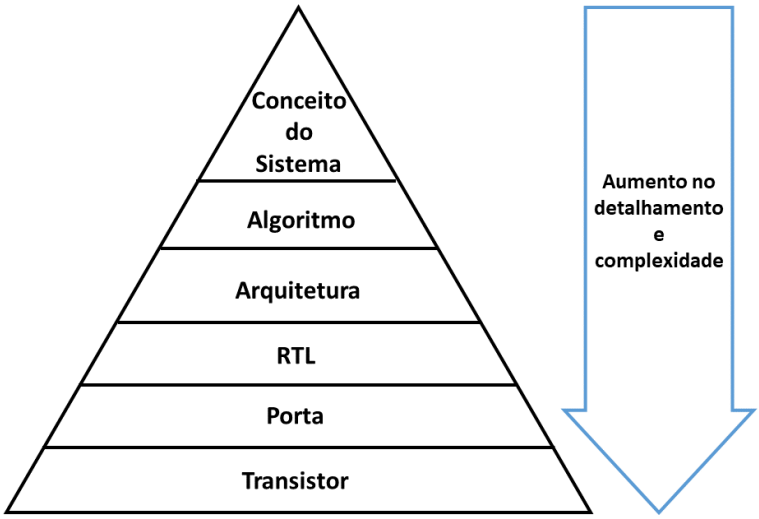


Figura 3.1: Estrutura Top-Down

mação de dois modelos diferentes (tecnologia + comportamento) produzindo um circuito manufaturável.

3.2 Fluxo Digital

A figura 3.2 apresenta um esquemático básico para criação de um circuito integrado digital. Neste trabalho o foco será a primeira parte que é a geração e simulação do código HDL.

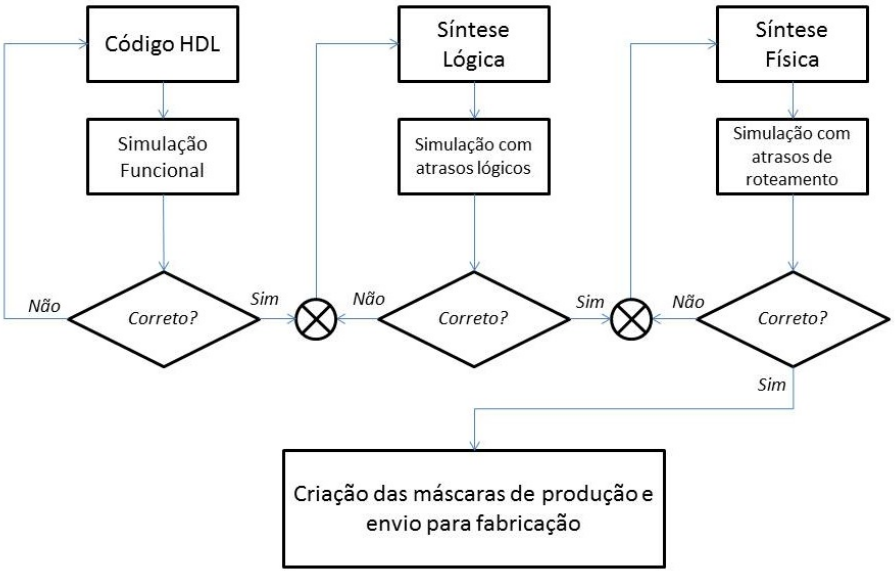


Figura 3.2: Fluxo Digital

3.2.1 Linguagem de descrição de hardware

Em seu livro [Palnitkar 2003] fala como as linguagens de descrição de hardware, do inglês *Hardware Description Language*(HDL), surgiram a partir da necessidade de desenvolvedores de circuitos eletrônicos necessitarem de uma linguagem de programação que possibilitasse modelar os processos paralelos que acontecem em circuitos.

A primeira parte no fluxo digital, é criar especificações e uma descrição do comportamento necessário para o circuito eletrônico que se quer criar. Depois disso essa descrição comportamental é manualmente sendo convertida para uma descrição RTL no formato da linguagem HDL. Após formado o seu modelo, é feita a primeira bateria de simulações que mostra se seu modelo está funcionando de acordo com o que havia sido especificado, essas primeiras são apenas comportamentais e não levam em conta atrasos de portas lógicas e roteamento.

Existem duas grandes linguagens HDL disponíveis nos dias atuais, VHDL e Verilog, ambas possuem suas peculiaridades de programação e é possível chegar em resultados iguais usando qualquer uma das duas, não havendo uma vantagem clara de nenhuma das duas linguagens depois de o profissional se aperfeiçoar. Assim, nesse trabalho foi escolhida a linguagem *Verilog* que, atualmente, é a mais utilizada no mercado, provavelmente por sua similaridade a linguagem C e também porque o código da omsp430 está em *verilog*.

3.3 Verificação Automatizada

Em seu livro, [Bushnell Michael and Agrawal Vishwani 2000] faz uma analogia interessante sobre as provas aplicadas por professores em sala de aula com os testes de circuitos eletrônicos. O professor faz perguntas e analisa as respostas, podendo ser, por exemplo, comparando-as com respostas já corretas disponíveis em um livro. A qualidade deste teste depende em quão bom o professor conseguiu fazer perguntas que abranjam todo o conteúdo ensinado. Assim, a verificação de circuitos eletrônicos pode ser visto da mesma maneira, é preciso saber o 'conteúdo' do dispositivo que é chamado de especificações e o objetivo é aplicar verificações que produzam respostas que são possíveis de analisar que garantam o funcionamento correto do dispositivo com suas especificações.

As verificações de dispositivos em *verilog* podem ser feito de forma manual, plotando as curvas de entradas e saídas e checando se elas batem com o esperado ou mesmo plotando tabelas verdades e checando se essas tabelas estão de acordo com as respostas esperadas pelo programador. Porém, em dispositivos grandes esses métodos se mostram ineficientes, pois com muitos dados para checar o tempo consumido é muito grande e as chances de erro são grandes. Assim, após algumas tentativas de fazer as verificações de forma manual foi escolhida a abordagem de verificações automatizadas.

Assim, na figura 3.3 está um esquema simplificado do modelo seguido neste trabalho.

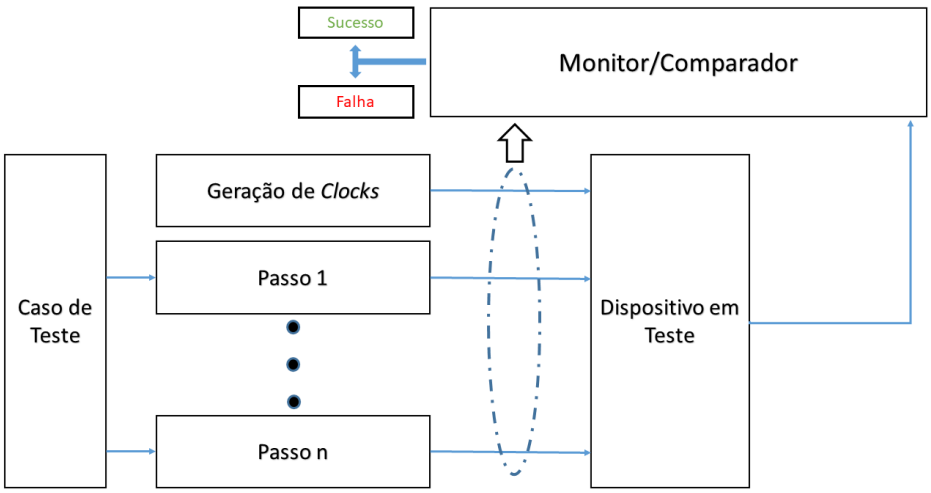


Figura 3.3: Esquema de uma verificação automatizada

Primeiramente, é escolhido um caso de verificação, depois são mapeados os passos para se obter o objetivo. Deste modo, é inicializado o dispositivo em verificação e aplicado os passos escolhidos, paralelamente o monitor observa e compara as entradas e saídas apresentadas, automaticamente, com o que seria esperado para essa verificação, parando a verificação a qualquer momento se ele apresentar um erro ou informando que não houve erros ao final.

Para a verificação do *Timer B* foram feitos 15 casos de verificação para checar suas funcionalidades:

1. Verificação de Reset:

Testa se a função *reset* do dispositivo está funcionando corretamente. Nesta verificação, é aplicado um *reset* inicial para observar se o dispositivo inicializa corretamente, e após isso é aplicado *resets* em tempos aleatórios e observa se os registradores voltam para os estados iniciais esperados.

2. Escrita no Dispositivo:

Verificação da escrita nos registrados do dispositivo. Nesta verificação, após inicializado o dispositivo é testado se o processo de escrita nos registros está funcionando corretamente, observando se os registradores estão realmente sendo escritos com o que foi colocado na entrada do dispositivo.

3. Leitura Registradores:

Verificação da leitura dos registros do dispositivo ligado no barramento. Nesta verificação é feito o contrário da anterior, como já está validado que realmente os registrados

estão recebendo o que é escrito pela verificação anterior essa verificação mostra se, quando o barramento, pede o valor de um registro o dispositivo mostra como saída o valor correto.

4. CNTLx 16 bits:

Verificação do modo de contagem com 16 bits e se a geração de *flag* de *overflow*. Nesta verificação, é criado um contador de 16 bits simples para simular o modos de contagem *continous*. E é visto se está ocorrendo a geração correta da *flag* de interrupção.

5. CNTLx 12 bits:

Verificação do modo de contagem com 12 bits e se a geração de *flag* de *overflow*. Nesta verificação, é criado um contador de 12 bits simples para simular o modos de contagem *continous*. E é visto se está ocorrendo a geração correta da *flag* de interrupção.

6. CNTLx 10 bits:

verificação do modo de contagem com 10 bits e se a geração de *flag* de *overflow*. Nesta verificação, é criado um contador de 10 bits simples para simular o modos de contagem *continous*. E é visto se está ocorrendo a geração correta da *flag* de interrupção.

7. CNTLx 8 bits:

Verificação do modo de contagem com 8 bits e se a geração de *flag* de *overflow*. Nesta verificação, é criado um contador de 8 bits simples para simular o modos de contagem *continous*. E é visto se está ocorrendo a geração correta da *flag* de interrupção.

8. TBSSELx:

Verificação da seleção de *clocks*. Nesta verificação, são gerados 3 *clocks* diferentes, um para cada *clock* diferente, *aclk*, *tbclk* e *smclk*. E depois é comparada a operação na função *continous* com a de um contador simples alimentado pelo mesmo *clock*.

9. IDx:

Verificação da divisão de *clock*. Nesta verificação, após inicializado o dispositivo, é colocado um *clock* para alimentar o dispositivo e outros *clocks* com múltiplos períodos em um contador simples. E assim, se compara a operação dos contadores.

10. MCx:

Verificação dos modos de contagem. Nesta verificação, após inicializado o dispositivo, são testados os 4 modos de contagem disponíveis e são comparadas com um contador simples. São testados seguidamente os modos *Up*, *Continous*, *Up/Down* e no final o *Stop*.

11. Capture Mode | Rising Edge:

Verificação do modo de captura quando usada a opção de detecção de flanco de subida. Nesta verificação é observada se o modo de captura está funcionando corretamente nos 3 canais do dispositivo para cada um das duas entradas de captura de cada canal no modo de captura de flanco de subida. São adicionados sinais nas entradas de captura e verificado se a geração das *flags* de interrupção e de *overflow*, COV, estão sendo geradas como esperado.

12. Capture Mode | Falling Edge:

Verificação do modo de captura quando usada a opção de detecção de flanco de descida. Nesta verificação é observada se o modo de captura está funcionando corretamente nos 3 canais do dispositivo para cada um das duas entradas de captura de cada canal no modo de captura de flanco de descida. São adicionados sinais nas entradas de captura e verificado se a geração das *flags* de interrupção e de *overflow*, COV, estão sendo geradas como esperado.

13. Catpure Mode | Rising/Falling Edge:

Verificação do modo de captura quando usada a opção de detecção de flanco de subida e descida. Nesta verificação é observada se o modo de captura está funcionando corretamente nos 3 canais do dispositivo para cada um das duas entradas de captura de cada canal no modo de captura de flanco de subida e descida. São adicionados sinais nas entradas de captura e verificado se a geração das *flags* de interrupção e de *overflow*, COV, estão sendo geradas como esperado.

14. CLLDx:

Verificação da atualização do latch *CCLx*. Nesta verificação, após inicializado o dispositivo, é visto se a atualização do sinal TBCLx está funcionando corretamente, dentro do dispositivo criado nesse trabalho ele possui o nome de TBCCRx. Assim, são testados todos os modos apresentados na tabela 2.3 e comparados com as respostas esperadas.

15. OUTMODx:

Verificação dos modos de saída. Nesta verificação é testado cada um dos sete modos de saída e o computador comparando as respostas em tempos definidos pelas respostas esperadas. São testados primeiro os modos 0,1,4 e 5 pelo fato de os 3 canais serem usados para esse modo, e os modos 2,3,6 e 7 são testados por último para os canais 1 e 2.

3.3.0.1 Passos no código

Será usado o código de validação 10 como exemplo para demonstrar como os códigos foram feitos em Verilog. Primeiramente, é necessário criar um módulo para verificação e instanciar esse módulo para o periférico.

Listagem 3.1: Exemplo de código 1

```
1  module omsp_timerB_testbench_10;
2
3          // Inputs
4          reg aclk_en;
5          reg dbg_freeze;
6          reg irq_tb0_acc;
7          reg mclk;
8          reg [13:0] per_addr;
9          reg [15:0] per_din;
10         reg per_en;
11         reg [1:0] per_we;
12         reg puc_rst;
13         reg smclk_en;
14         reg tb_cci0a;
15         reg tb_cci0b;
16         reg tb_cci1a;
17         reg tb_cci1b;
18         reg tb_cci2a;
19         reg tb_cci2b;
20         reg tbclk;
21
22         // Outputs
23         wire irq_tb0;
24         wire irq_tb1;
25         wire [15:0] per_dout;
26         wire tb_out0;
27         wire tb_out0_en;
28         wire tb_out1;
29         wire tb_out1_en;
30         wire tb_out2;
31         wire tb_out2_en;
32
33         // Instantiate the Unit Under Test (UUT)
```

```
34          omsp_timerB uut (
35                  .irq_tb0(irq_tb0),
36                  .irq_tb1(irq_tb1),
37                  .per_dout(per_dout),
38                  .tb_out0(tb_out0),
39                  .tb_out0_en(tb_out0_en),
40                  .tb_out1(tb_out1),
41                  .tb_out1_en(tb_out1_en),
42                  .tb_out2(tb_out2),
43                  .tb_out2_en(tb_out2_en),
44                  .aclk_en(aclk_en),
45                  .dbg_freeze(dbg_freeze),
46                  .irq_tb0_acc(irq_tb0_acc),
47                  .mclk(mclk),
48                  .per_addr(per_addr),
49                  .per_din(per_din),
50                  .per_en(per_en),
51                  .per_we(per_we),
52                  .puc_rst(puc_rst),
53                  .smclk_en(smclk_en),
54                  .tb_cci0a(tb_cci0a),
55                  .tb_cci0b(tb_cci0b),
56                  .tb_cci1a(tb_cci1a),
57                  .tb_cci1b(tb_cci1b),
58                  .tb_cci2a(tb_cci2a),
59                  .tb_cci2b(tb_cci2b),
60                  .tbclk(tbclk)
61          );
```

Após isso, são adicionados registros e fios que serão necessários para o teste, além disso são feitos eventos, que é uma funcionalidade do verilog que facilita na montagem das verificações.

Listagem 3.2: Exemplo de código 2

```
1  event configuracao1;
2  event terminate_sim;
3  reg dut_error;
4  reg [1:0] sel_clk;
5  reg up;
6  reg down;
7  reg fase;
8  reg [1:0] index;
9  reg [15:0] counter;
10 reg start_counter;
```

Esses registros funcionam como variáveis que podem ser usadas nos testes e os eventos como funções pensando em uma linguagem de programação mais conhecida como C. Após isso, são inicializados as entradas do dispositivo e dado um reset para que seus registros internos também possuam valores conhecidos.

Listagem 3.3: Exemplo de código 3

```
1          initial begin
2                  // Initialize Inputs
3                  aclk_en = 0;
4                  dbg_freeze = 0;
5                  irq_tb0_acc = 0;
6                  mclk = 0;
7                  per_addr = 0;
8                  per_din = 0;
9                  per_en = 0;
10                 per_we = 0;
11                 puc_rst = 1;
12                 smclk_en = 0;
13                 tb_cci0a = 0;
14                 tb_cci0b = 0;
15                 tb_cci1a = 0;
16                 tb_cci1b = 0;
17                 tb_cci2a = 0;
18                 tb_cci2b = 0;
19                 tbclk = 0;
20 #5               puc_rst=0;
21                 counter = 0;
22
23                 end
```

O *inicial* significa que o tempo inicial é considerado como 0, e o 5 indica que esse passo será feito depois de 5 unidades de tempo que pode ser definida pelo usuário, como essa verificação não é considerado atrasos de porta é usado o menor tempo possível para o simulador que é de 1ps. Após, são configurados alguns comandos que serão colocados na tela do console, é aberto um arquivo para despejar as formas de onda e são configurados os clocks usados na verificação.

Listagem 3.4: Exemplo de código 4

```
1  initial
2  begin
3   $display ("######################################################");
4   dut_error = 0;
5  end
6
7  always
8    #1 mclk = !mclk;
9
10 always
11   #5 tbclk = !tbclk;
12
13
14 initial
15 begin
```

```
16    $dumpfile ("testbench_10.vcd");
17    $dumpvars;
18  end
19
20
21  initial
22  @(terminate_sim)  begin
23   $display ("Simulacao Concluida");
24   if (dut_error == 0) begin
25     $display ("Resultado : SUCESSO");
26   end
27   else begin
28     $display ("Resultado: FALHA");
29   end
30   $display ("#####################################################");
31   #1 $finish;
32  end
```

No código 3.4 é possível ver que existe um sinal chamado de dut_error, o qual é configurado para 0, caso quando o evento de término da verificação for chamado e o seu valor estiver 1 o resultado será considerado falha, caso não mude de valor a verificação será considera sucedida.

As partes explicadas até são praticamente iguais em todas as verificações, agora o que realmente muda de verificação para verificação serão os eventos criados para se simular a situação que se deseja verificar. Deste modo, agora são mudadas as entradas e sabendo-se as saídas é necessário se fazer lógicas que verifiquem se a resposta do sistema está de acordo.

Listagem 3.5: Exemplo de código 5

```
1  always @(negedge tbclk)
2  if ((uut.tbr != counter) & ( start_counter == 1 ) & ( fase == 0 )) begin
3    $display ("Erro no tempo %d",$time);
4    dut_error = 1;
5    #1 -> terminate_sim;
6  end
```

Um exemplo de lógica pode ser visto no código 3.5, nele é usado a estrutura *always*, que como o nome em inglês mesmo diz, sempre que existir um flanco de descida no tbclk ele faz a verificação se a contagem do dispositivo está igual a contagem correta, se ela estiver não acontece nada, porém se estiver incorreta ele muda o valor de dut_error para 1 e termina a simulação além de plotar em que momento ocorreu o erro.

3.4 Especificações

As especificações obrigatórias para o periférico escolhido serão:

- Possuir entradas e saídas similares ao omps_timer A para fácil integralização do sistema
- Possuir 4 modos de contagem (Up, Down, Up/Down, Stop)
- Possuir os modos de saída do timer B da TI.
- Possuir a função de dupla buferização
- Possuir todas as demais funções do timer B que possam ser úteis.

Capítulo 4

Resultados

Como explicado no capítulo 3 o foco do trabalho foi na primeira parte do fluxo digital apresentado pela figura 3.2. Foi produzido o código HDL *omsp_timerB* e como esse código apresenta centenas de linhas ele ficará disponibilizado, junto com todos os arquivos de teste, na rede interna do LDCI.

Todos as funções mostradas pelo [Texas Instruments 2009] foram implementadas, exceta a lógica de agrupamento de *latchs*, que foi considerada dispensável pelo fato de ter sido escolhida o modelo de 3 canais. O *omsp_timerB* possui 17 entradas e 9 saídas apresentadas nas tabelas 4.1 e 4.2. Essas entradas e saídas foram escolhidas de forma a imitar o melhor possível o timer A já presente na openMSP430, para com essa escolha diminuir a complexidade de ligar o timer B no barramento do sistema.

Entradas	**Bits**	**Descrição**
mclk	1	Clock mestre do sistema
aclk_en	1	Clock ACLK
smclk_en	1	Clock SMCLK
tbclk	1	Clock TBCLK externo
dbg_freeze	1	congelar o dispositvo
irq_tb0_acc	1	Pedido de interrupção aceita
per_addr	14	Entrada de Endereço
per_din	16	Entrada de dados
per_en	1	Ativação de Leitura/Escrita
per_we	2	Ativação de Escrita
puc_rst	1	Reset mestre do sistema
tb_cci0a	1	Captura - Canal 0 - A
tb_cci0b	1	Captura - Canal 0 - B
tb_cci1a	1	Captura - Canal 1 - A
tb_cci1b	1	Captura - Canal 1 - B
tb_cci2a	1	Captura - Canal 2 - A
tb_cci2b	1	Captura - Canal 2 - B

Tabela 4.1: Entradas omsp_timerB

Saída	Bits	Descrição
irq_tb0	1	Interrupção : TBCCR0
irq_tb1	1	Interrupção : TBIV, TBCCR1, TBCCR2
per_dout_en	16	Saída do periférico
tb_out0	1	Saída - Canal 0
tb_out0_en	1	Saída - Canal 0 - Ativo
tb_out1	1	Saída - Canal 1
tb_out1_en	1	Saída - Canal 1 - Ativo
tb_out2	1	Saída - Canal 2
tb_out2_en	1	Saída - Canal 2 - Ativo

Tabela 4.2: Saídas omsp_timerB

Para a lógica de escrita e leitura no periférico foi usada a padrão disponibilizada no arquivo de molde de periféricos da *openMSP430*. Assim, os endereços dos registradores do periférico foram escolhidas igual aos endereços presentes nos registrados no apêndice.

Os registradores são codificados dentro do dispositivo pelo método *one-hot* que consiste em designar um número de 256 bits diferente para cada registrador, nesse número todos os bits são 0 exceto por apenas um que apresenta valor 1, a localização desse bit depende do endereço do registrador.

Agora, para a leitura ou escrita, é usada uma lógica que observa os sinais de entrada para determinar o que deverá ser feito. Primeiro, para haver tanto escrita quando leitura é necessário que a entrada per_en seja 1, após isso é visto se o endereço escrito em per_addr é válido, agora se per_we tiver algum bit 1 será feito o processo de escrita se ele for 0 será feito o processo de leitura. Escolhido o processo, é feito o processo de decodificação *one-hot* que determina em qual registrador os sinais em per_in entrarão ou de qual registrador os dados serão copiados a per_dout.

Os testes apresentados no capítulo anterior foram executados e tiveram resultados positivos. Uma das limitações do dispositivo é que não é possível escrever no registro TBR, essa limitação foi deixada pelo fato de não ter muitas funções práticas a escrita em TBR, assim não é necessário mudança na lógica, que provavelmente faria um aumento em portas lógicas e registradores aumentando a complexidade e atrasos do periférico. Outra está no fato que usando o modo de saída 0, a mudança da saída não ocorre imediatamente e sim no primeiro flanco de subida do clock sendo usado pelo *timer*.

Todos os teste foram feitos utilizando o *software* gratuito *Icarus Verilog* gerando arquivos acontendo as formas de onda, que por sua vez podem ser lidas pelo programa, também gratuito, *GTKWAVE*. Como todos os testes são automáticos, eles voltam uma resposta imediata no final do teste sobre o sucesso ou falha do teste, como todos os testes obtiveram resultados positivos serão escolhidos alguns testes para se analisar as formas de onda e observar se, de fato, o teste automático mostrou um resultado coerente.

4.1 Teste 10 - MCx

Podemos ver o resultado do décimo teste na figura 4.1. Como foi apresentado no capítulo passado o teste 10 verifica se os modos de contagem do *Timer B* estão funcionando de maneira adequada.

```
PS D:\projetos\TimerB_n> iverilog -o omsp_timerB -f '.\file_list.txt'
PS D:\projetos\TimerB_n> vvp omsp_timerB
################################################
VCD info: dumpfile testbench_10.vcd opened for output.
Iniciando Simulacao 10 - MCx
Testando modo Up
Modo Up - Ok
Testando modo Continuous
Modo Continous - Ok
Testando modo Up-Down
Modo Up-Down - Ok
Testando modo Hold
Modo Hold - OK
Simulacao Concluida
Resultado : SUCESSO
################################################
```

Figura 4.1: Resposta console simulação 10

O resultado da simulação é positiva e é possível verificar a forma de onda colocada no arquivo *testbench_10.vcd* para validar se esse resultado é coerente. A primeira parte do teste como é possível ser observada na figura 4.1 é o teste do modo de contagem *up*. A figura 4.2 apresenta uma forma de onda que representa a parte *up* do teste 10.

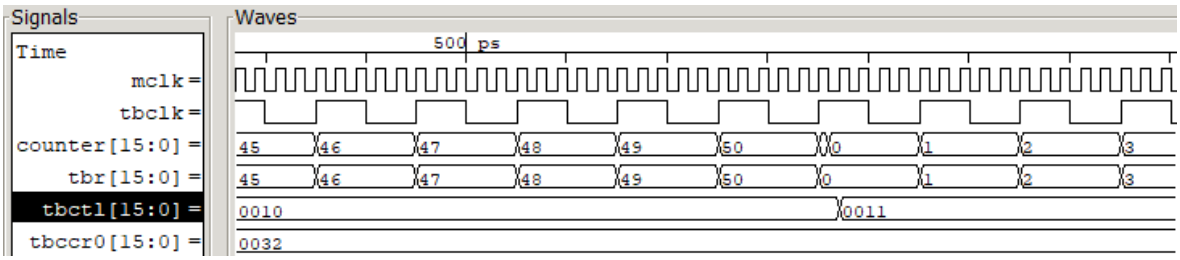


Figura 4.2: Forma de onda teste 10 - Up

Na figura são representados 6 sinais, *mclk* é o *clock* mestre do microcontrolador, *tbclk* é o clock que será utilizado no contador, *counter* é o contador feito para simular a contagem Up e ser comparado com o contador do *Timer B*, TBR, seus valores estão representados em decimais. O registrador TBCTL mostra a configuração do *timer*, seu valor está em hexadecimal junto com o registro chamado de TBCCR0 que na verdade é a nomenclatura usada para representar o valor do latch TBCL0.

Assim, o timer está configurado no modo de contagem *up* e com TBCL0 igual a 50, 50 em decimal é o mesmo que 32 em hexadecimal. O teste ocorre da forma em que todo flanco de descida do tbclk ele compara se *TBR* e *Counter* são iguais. *Counter* apresenta um pequeno erro ao mudar de 50 para 0 mas da forma como foi feito a comparação isso não é muito importante e não interfere em nada na simulação.

Nessa simulação, também é possível ver a flag de *overflow* quando o contador atinge 0 novamente pelo fato do bit menos significativo de TBCTL mudar de valor. E assim, parece que o teste do modo de contagem *up* está coerente. Após isso, é feito o teste para o modo continuous mostrado na figura 4.3.

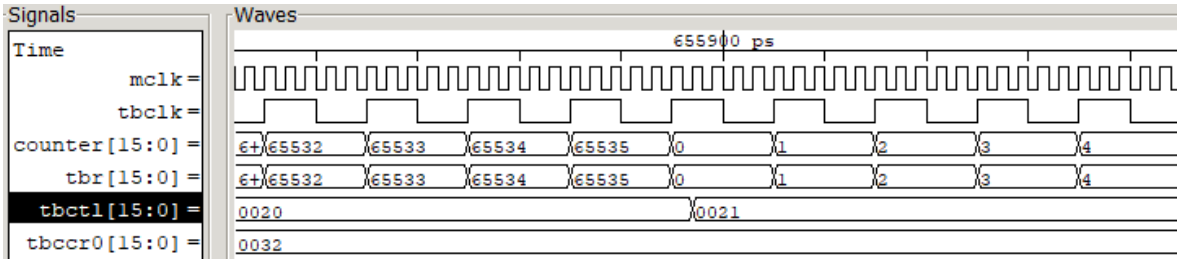


Figura 4.3: Forma de onda teste 10 - Continuous

O valor de TBCCR0 não foi mudado, já que de modo isso não deveria interferir no modo *continuous*. E de fato não interfere pelo que é possível ver na figura 4.3. Como o contador está configurado para o modo de 16 bits ele conta até 65535 antes de voltar a 0, e novamente é possível ver que a flag de *overflow* é criada, fazendo o resultado do teste da parte *continous* parecer coerente. Testada essa parte o teste segue para o modo *up/down*.

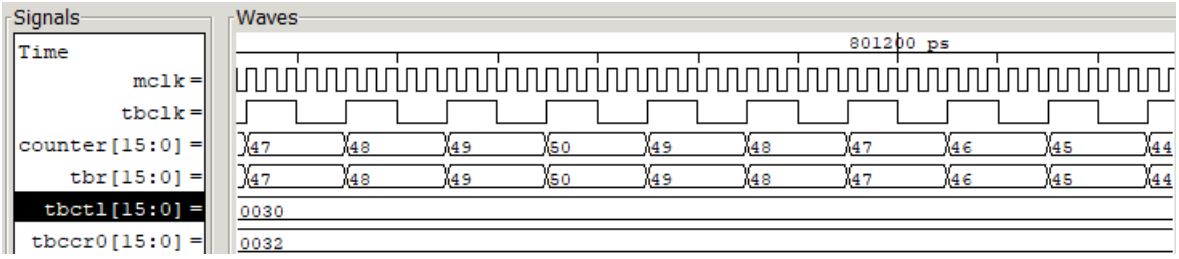


Figura 4.4: Forma de onda teste 10 - Up/Down

Novamente o valor de TBCCR0 não foi modificado, assim o contador deve contar a 50 e depois de volta a 0. Na figura 4.4 é possível ver a transição da contagem crescente para a decrescente mostrando que esse teste também parece estar certo. Deste modo, falta apenas testar o modo *stop*, apresentando na figura 4.5. quando o *timer* é colocado no modo *stop*, junto com o *counter*, é possível ver que o valor de contagem congela no 3, como deveria ocorrer.

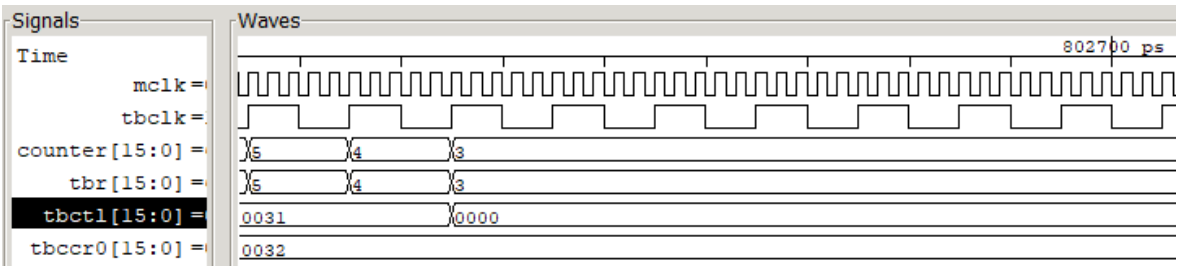


Figura 4.5: Forma de onda teste 10 - Stop

Portanto, a análise de algumas formas de onda conseguiram dar base ao resultado mostrado no console, mostrando que o teste automático está desempenhando seu papel corretamente.

4.2 Teste 14 - CLLDx

No teste 14 é testado a atualização do latch TBCL0, que nesse dispositivo é chamado de TBCCR0. Na figura 4.6 é apresentado a resposta do teste automático. Como no teste mostrado na sessão anterior o resultado é positivo e agora pela análise das formas de ondas será possível comprovar que de fato a atualização do latch está funcionando como descrito na tabela 2.3

```
PS D:\projetos\TimerB_n> iverilog -o omsp_timerB -f '.\file_list.txt'
PS D:\projetos\TimerB_n> vvp omsp_timerB
####################################################
VCD info: dumpfile testbench_14.vcd opened for output.
Iniciando Simulacao 14 - CLLDx
CLLDx 00
CLLDx 00 - OK
CLLDx 01
CLLDx 01 - OK
CLLDx 10
CLLDx 10 - Up e Continous OK
CLLDx 10 - Up/Down OK
CLLDx 11
CLLDx 11 - OK
Simulacao Concluida
Resultado : SUCESSO
####################################################
```

Figura 4.6: Resposta console simulação 14

A primeira parte do teste consiste em verificar o caso em que CLLDx é 00, ou seja, o registrador deve atualizar exatamente na mesma hora em que é escrito em seu registro. As figuras 4.7 e 4.8 mostram esse caso.

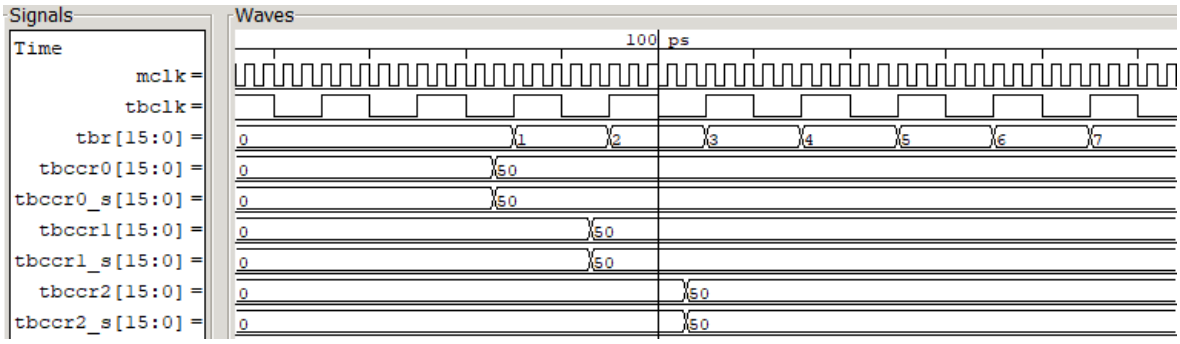


Figura 4.7: Forma de onda 1 - Teste 14 - CLLDx = 00

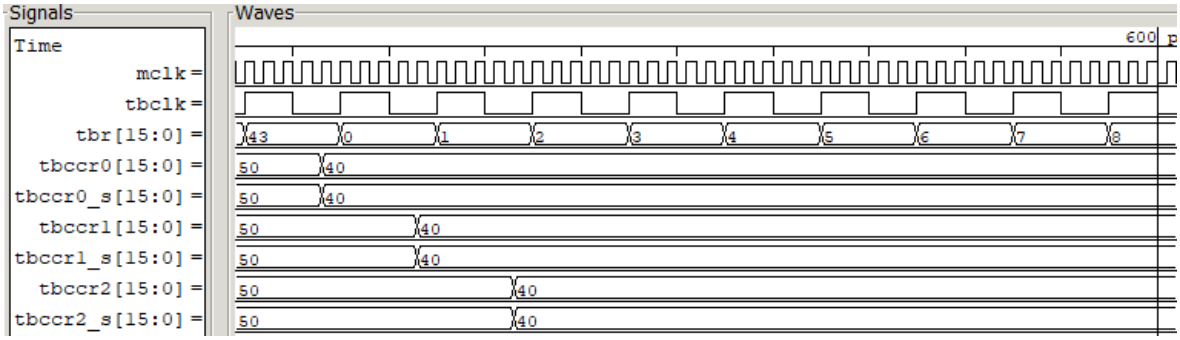


Figura 4.8: Forma de onda 2 - Teste 14 - CLLDx = 00

Os sinais de saída dos *latchs* são representados, como já dito antes, pelos sinais TBCCRx onde x indica o canal, 0, 1 ou 2. O sinal TBCCRx_s é que guarda o valor em que TBCCRx deve ser atualizado. Nas figuras é possível confirmar o comportamento esperado, pois no momento que se muda os sinais TBCCRx_s, os sinais TBCCRx são atualizados imediatamente. Seguindo o teste temos as figuras 4.9 e 4.10 que representam o caso em que CLLDx é 01 e assim a atualização de TBCCRx deve acontecer no momento que TBR chega em 0.

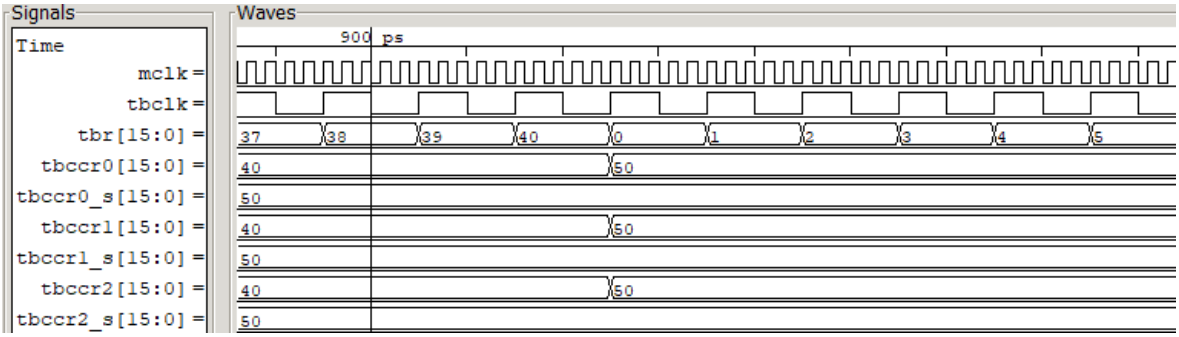


Figura 4.9: Forma de onda 1 - Teste 14 - CLLDx = 01

Nas figuras é possível ver que os sinais TBCCRx_s são diferentes dos sinais TBCCRx, e que eles só se tornam iguais, ou seja TBCCRx é atualizado no momento em que TBR

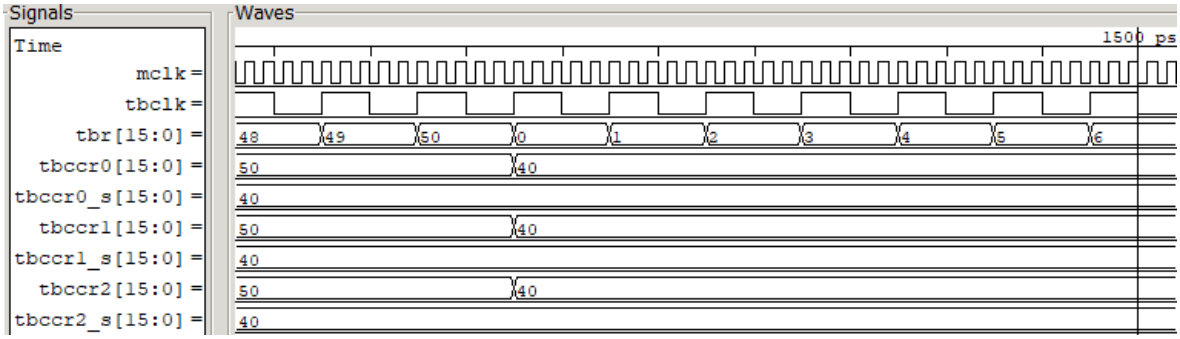


Figura 4.10: Forma de onda 2 - Teste 14 - CLLDx = 01

chega no valor de 0 como deveria ocorrer. A próxima parte do teste CLLDx igual a 10, nesse caso TBCCRx deverá ser atualizado quando TBR for 0 nos modos *up* e *continuous* e no modo *up/down* quando TBR for 0 ou o antigo TBCCR0. Assim, esse teste dividido em duas partes, uma parte que avalia se a atualização está ocorrendo de forma correta nos modos *up* e *continuous* mostrados nas figuras 4.11 e 4.12 outra que avalia se está ocorrendo de forma correta no modo *up/down*, figuras 4.13 e 4.14.

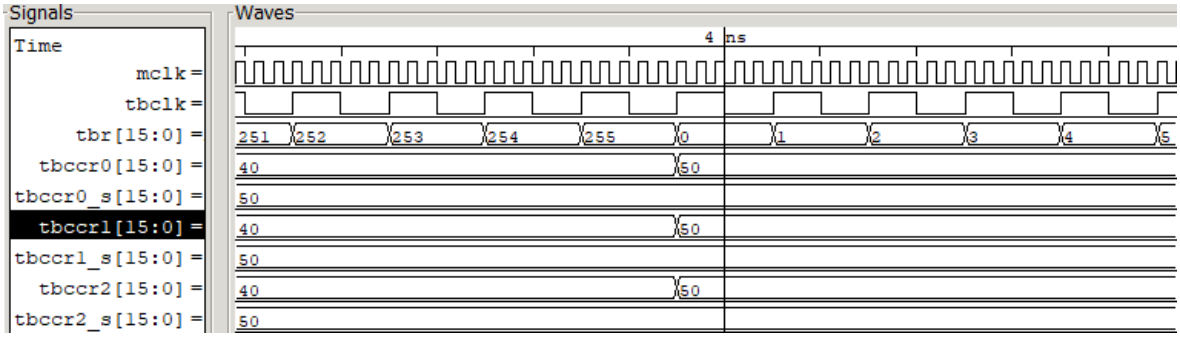


Figura 4.11: Forma de onda 1 - Teste 14 - CLLDx = 10

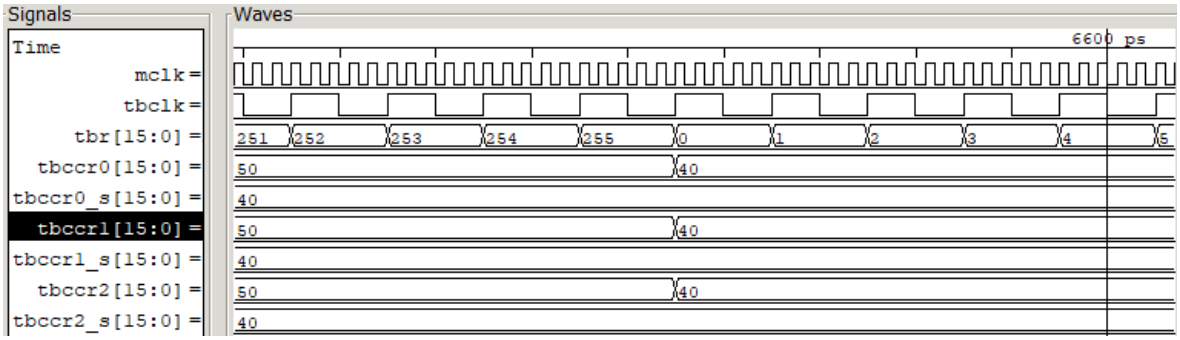


Figura 4.12: Forma de onda 2 - Teste 14 - CLLDx = 10

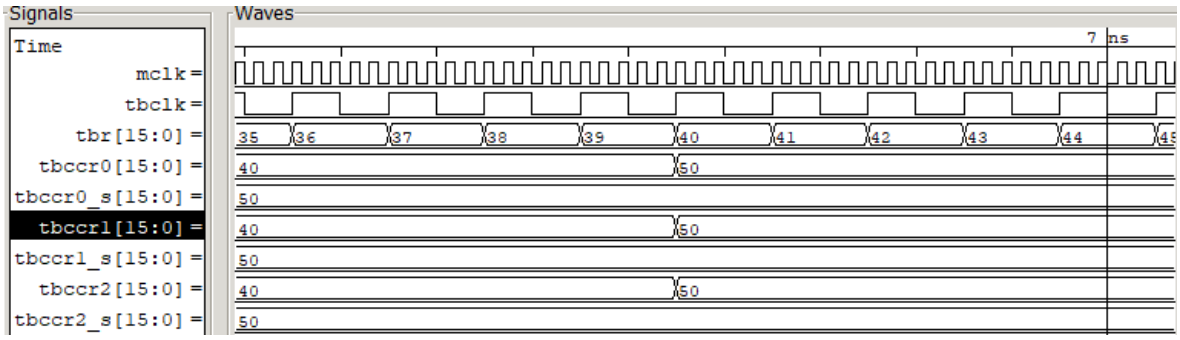


Figura 4.13: Forma de onda 3 - Teste 14 - CLLDx = 10

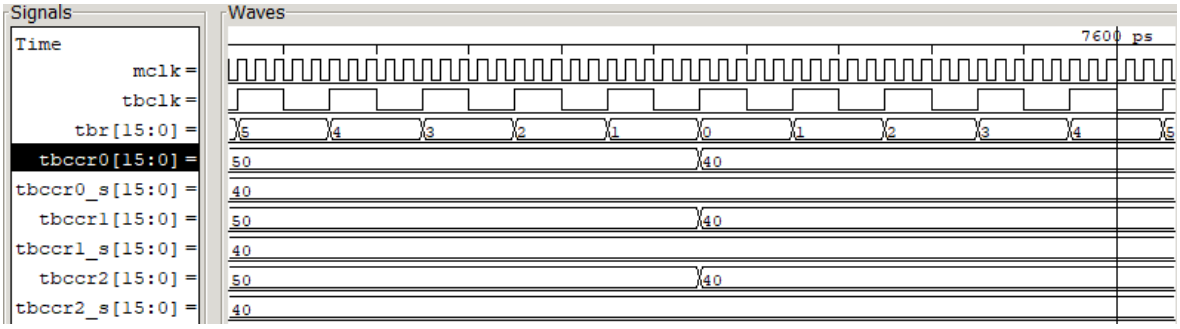


Figura 4.14: Forma de onda 4 - Teste 14 - CLLDx = 10

Observando as primeiras duas que estão representando o modo *continuous* com 8 bits, por isso a contagem até 255, para representar a primeira parte. Nesse modo, o funcionamento é igual ao caso anterior, CLLDx igual a 01, para esses dois modos é possível perceber que funciona de forma esperada pelos mesmos postos expostos no caso anterior. A diferença vem no caso *up/down*, nas figuras 4.13 e 4.14 é possível ver a atualização no caso de TBR igual a TBCCR0 antigo e e TBR igual a 0, respectivamente. E desse forma, só resta checar o caso em que CLLDx é 11, representado na figura 4.15.

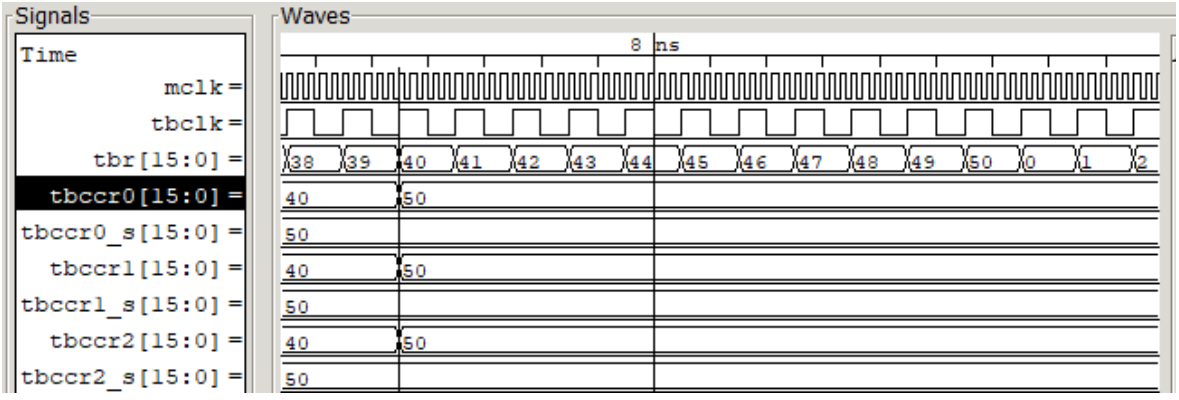


Figura 4.15: Forma de onda- Teste 14 - CLLDx = 11

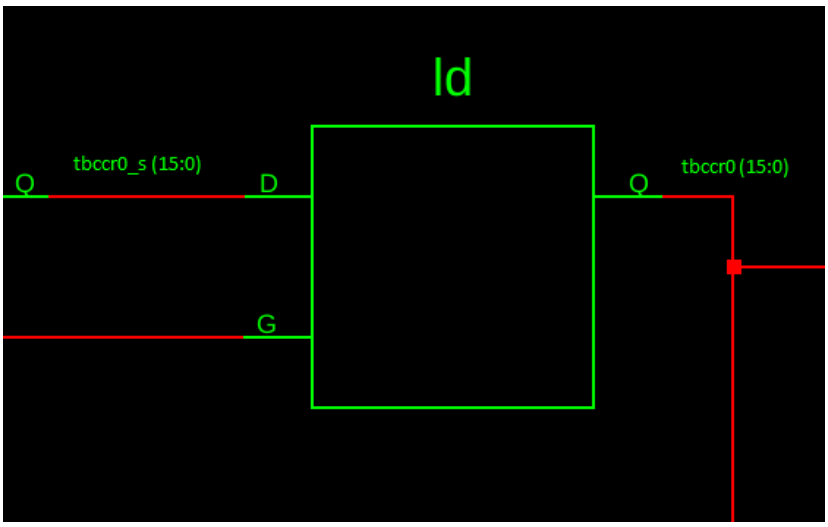


Figura 4.16: latch tipo D implementação ISE

Como é possível ser observado na figura, como era esperado os sinais TBCCRx foram atualizados no momento em que TBR atinge o valor de TBCCRx antigo. Mostrando assim que o teste automático realmente está de acordo com o que está acontecendo. A forma que o computador compara automaticamente se o programa está certo é por dois testes, o primeiro compara se TBCCRx é igual TBCCRx_s e o outro compara se os dois são diferentes, com uma análise dos tempos é indicado para o computador em quais horas cada teste deve ser positivo e se isso não ocorrer ele finaliza a simulação e indica o sinal de erro, como não houve erros, como foi possível observar pelas formas de onda, o computador não para a simulação e retorna o resultado como positivo para o teste.

Um resultado interessante foi, que houve realmente a formação de um *latch* tipo D, como é mostrado na figura 4.16 tirada de uma esquemático RTL feito pelo programa ISE da Xilinx, mesmo ele não ter sido programado explicitamente no código, mostrando mais uma razão que a função parece estar correta e de acordo com o proposto no manual da TI.

4.3 Teste 15 - OUTMODx

O teste 15 verifica os modos de saída do dispositivo apresentados na tabela 2.4. São 8 modos de saída ao todo, eles foram divididos em duas partes. Na primeira são testados os modos 0, 1, 2, 4 e 5, pelo fato desses modos terem sentido ser usados nos 3 canais, pois nos outros modos como TBCCRx é igual TBCCR0 para o canal 0 os modos não fazem sentido.

O primeiro modo de saída consiste em a saída ser igual ao bit OUT, e podemos observar nas figuras 4.18 e 4.19, que realmente, a saída está seguindo o bit OUT (O bit out é terceiro bit menos significativo do sinal TBCCRx). Como é possível ver a limitação falada no inicio desse capítulo a saída só é atualizada no primeiro flanco de subida do clock usado, no caso tbclk.

```
################################################
PS D:\projetos\TimerB_n> iverilog -o omsp_timerB -f '.\file_list.txt'
PS D:\projetos\TimerB_n> vvp omsp_timerB
################################################
VCD info: dumpfile testbench_12.vcd opened for output.
Iniciando Simulacao 15 - OUTMODx
OUTMODx 0
OUTMODx 0 - OK
OUTMODx 1
OUTMODx 1 - OK
OUTMODx 5
OUTMODx 5 - OK
OUTMODx 4
OUTMODx 4 - OK
OUTMODx 2
OUTMODx 2 - OK
OUTMODx 3
OUTMODx 3 - OK
OUTMODx 6
OUTMODx 6 - OK
OUTMODx 7
OUTMODx 7 - OK
Simulacao Concluida
Resultado : SUCESSO
################################################
```

Figura 4.17: Resposta console simulação 15

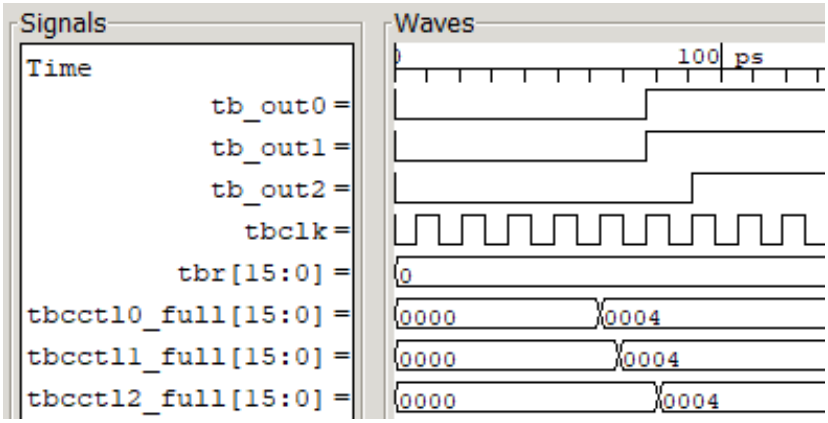


Figura 4.18: Bit OUT = 1 - Teste 15 - OUTMOD 0

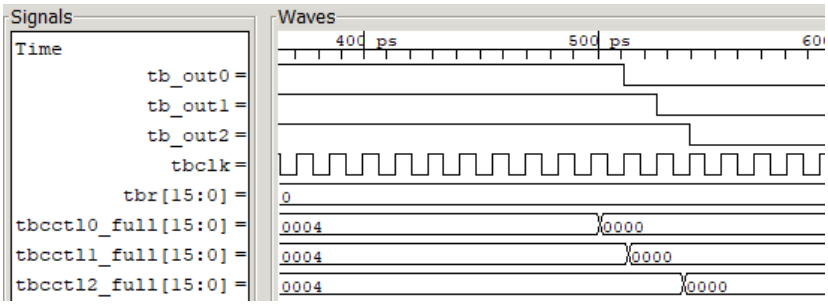


Figura 4.19: Bit OUT = 0 - Teste 15 - OUTMOD 0

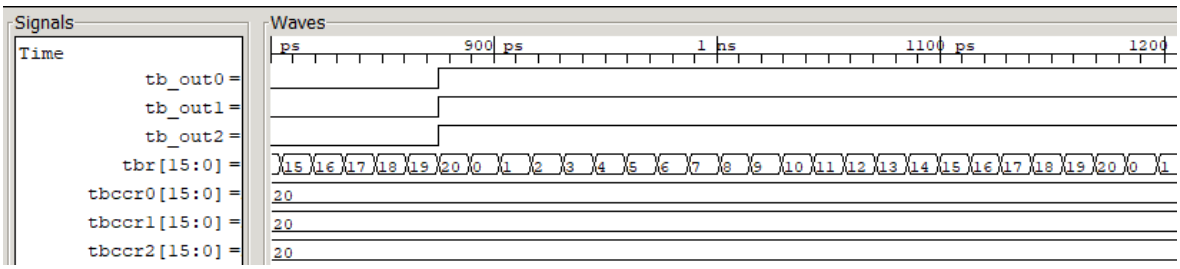


Figura 4.20: Teste 15 - OUTMOD 1

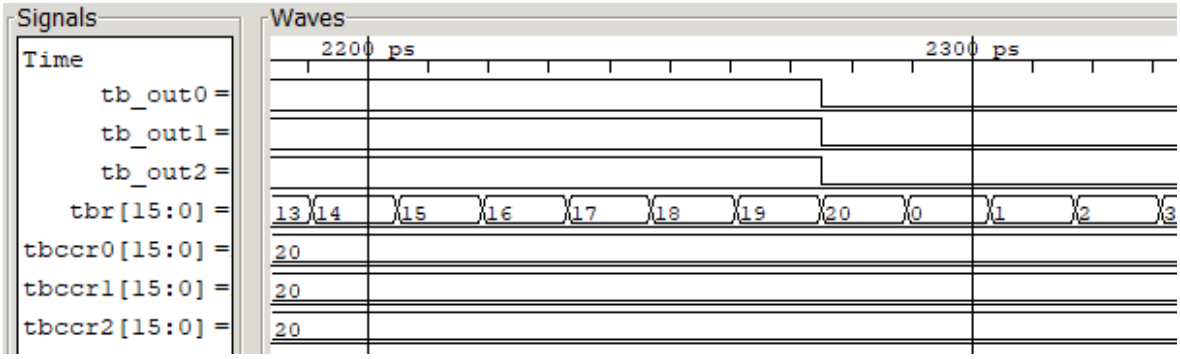


Figura 4.21: Teste 15 - OUTMOD 5

Agora foi escolhido fazer os testes dos modos 1 e 5 de forma complementar já que o modo 1 é o *SET*, ou seja quando, TBR é igual ao valor determinado por TBCCRx, nesse caso 20, a saída é mudada para 1 como é possível ser observado na figura 4.20, já na figura 4.21, o modo 5, *RESET*, faz exatamente o contrário e leva o valor das saídas a 0 quando TBR = TBCCRx.

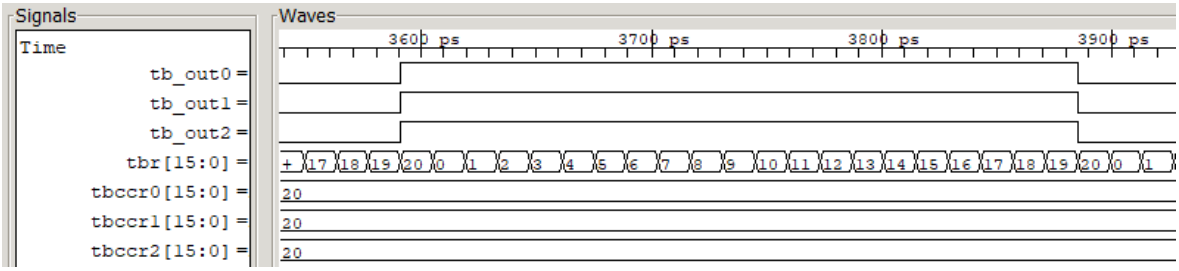


Figura 4.22: Teste 15 - OUTMOD 4

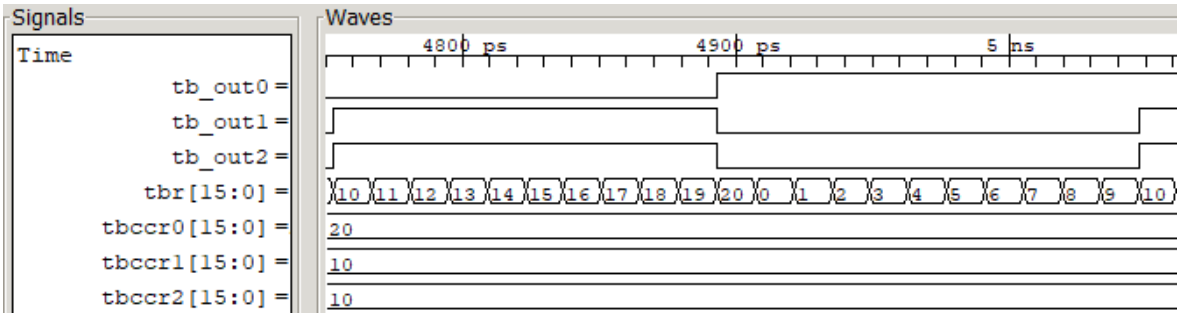


Figura 4.23: Teste 15 - OUTMOD 2

A figura 4.22 mostra o modo 4, *TOGGLE*, esse é o último modo em que se faz sentido usar o canal 0. Neste modo sempre que TBR atinge TBCCRx ele inverte o sinal da saída e isso pode ser observado na figura.

A partir deste parte, TBCCR1 e TBCCR2 foram mudados para 10 enquanto TBCCR0 continua 20. O primeiro modo a ser testado nessa parte é o modo 2, *TOGGLE/RESET*, apresentado na figura 4.23. Os sinais apresentados na saída do canal 0 serão ignorados nos próximos teste, assim nesse caso quando TBR é igual a 10 ocorre a inversão do sinal de saída, como é possível observar, e quando é igual a 20 ele faz com que as saídas virem 0.

Para as próximas curvas, escolheu o uso do modo *up/down*, pois parte das curvas ficaria igual no modo *up*. Assim, o modo de saída 3, *SET/RESET*, apresenta a seguinte configuração que pode ser observada na figura 4.24, quando TBR é igual a 20 ele torna as saídas 0 e quando é igual a 10 ele as torna 1.

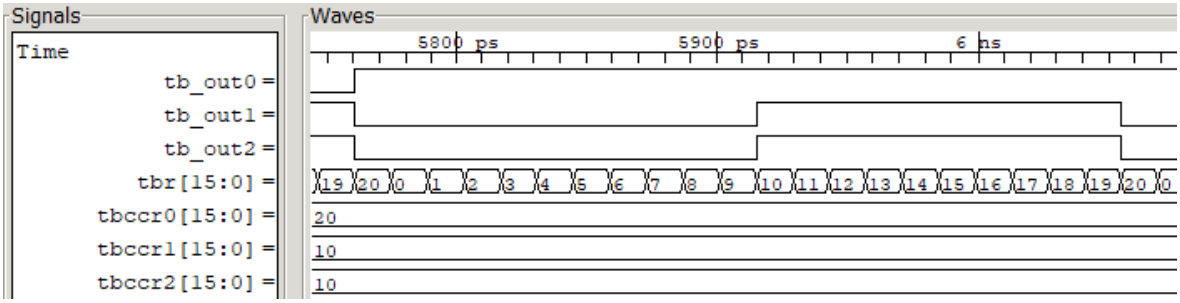


Figura 4.24: Teste 15 - OUTMOD 3

As últimas duas figuras podem ser analisadas de forma análoga as anteriores mostrando que o funcionamento dos modos de saída está funcionando, de fato, como o esperado. A

figura 4.25 representa o modo 6, *TOGGLE/SET*, TBR igual a 20 inverte o sinal, TBR igual a 10 torna os sinais igual a 1 binário e já na figura4.26 modo 7, *RESET/SET*, TBR igual a 20 torna os sinais igual a 0, TBR igual a 10 torna os sinais igual a 1.

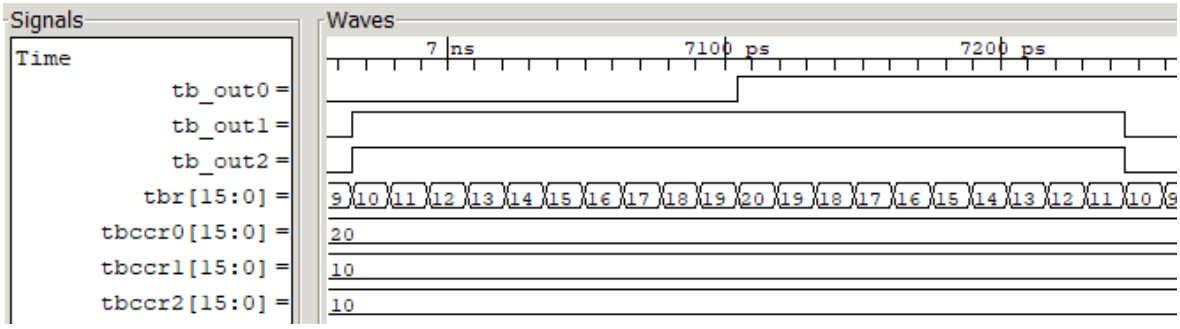


Figura 4.25: Teste 15 - OUTMOD 6

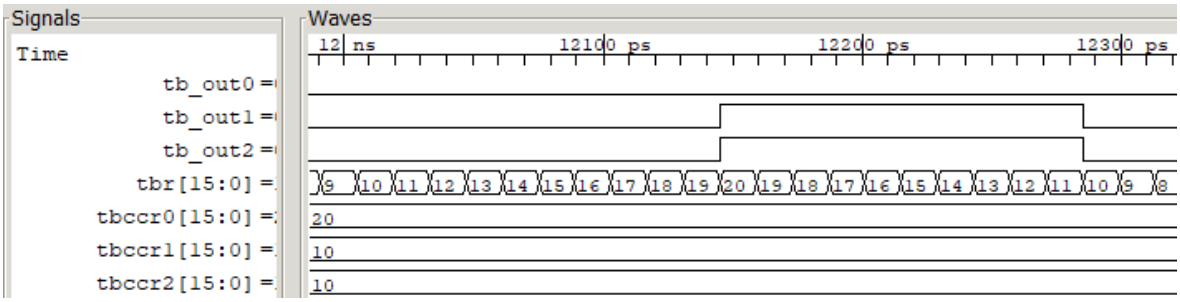


Figura 4.26: Teste 15 - OUTMOD 7

Como o modo de saída é usado para fazer ondas PWM, junto com o fato que foi demonstrado que a atualização do sinal TBCCR0, que é o diferencial principal pelo qual o timer B foi escolhido, é possível com esse dispositivo criar ondas PWM livres de *glitchs*.

Todos os testes podem ser analisados de forma análoga aos três anteriores, porém com esses três testes é possível demonstrar as funções básicas que eram necessitadas quando houve o pensamento desse periférico.

Capítulo 5

Conclusão

Este trabalho teve como objetivo principal a escolha de um microcontrolador que pudesse ser usado pelo LDCI para disponibilizar de maneira mais prática os seus blocos de propriedade intelectual (IPs). A escolha do *openMSP430* foi justificada principalmente pelo domínio que os membros do laboratório possuem em relação ao uso dessa arquitetura. Para demonstrar o acoplamento de um periférico ao sistema, foi implementado o periférico *Timer B*.

O periférico desenvolvido é compatível com a documentação da Texas Instruments, e implementa um temporizador com dupla buferização nos comparadores. O timer B foi escolhido por gerar sinais PWM, sem riscos de *glitches*.

O periférico desenvolvido apresentou algumas limitações, como a não possibilidade de escrita no registro de contagem TBR, e o fato de no modo de saída 0 é preciso esperar um flanco de subida de clock para a atualização da saída. Porém, nenhuma dessas limitações influencia nos comportamentos desejados desse periférico, que é a contagem e geração de ondas PWM.

Assim, com as demostrações dos testes apresentadas no capítulo 4 é possível dizer que o periférico omsp_timerB atende ao objetivo final de criar o código HDL de um periférico capaz de criar ondas PWM sem *glitches* e com possibilidade de ligamento no barramento da openMSP430.

5.1 Trabalhos Futuros

Agora que o periférico está com seu comportamento validado é possível acrescentá-lo na documentação da openMSP430 para ligá-lo em seu barramento. Assim, é possível fazer testes mais complexos com o omsp_timerB ligado diretamente na openMSP430 e após isso as sínteses lógica e física criando assim um IP para o LDCI criar seu próprio microprocessador. O código também pode ser usado de exemplo para criar outros periféricos que possam ser desejados.

Referências Bibliográficas

[Bushnell Michael and Agrawal Vishwani 2000] Bushnell Michael, L. and Agrawal Vishwani, D. (2000). Essentials of electronic testing for digital, memory and mixed-signal vlsi.

[Crisp 2003] Crisp, J. (2003). *Introduction to microprocessors and microcontrollers*. Elsevier.

[Girard 2009] Girard, O. (2009). openmsp430.

[Nolting 2015] Nolting, S. (2015). Neo430.

[Palnitkar 2003] Palnitkar, S. (2003). *Verilog HDL: a guide to digital design and synthesis*, volume 1. Prentice Hall Professional.

[Pereira 2018] Pereira, I. (2018). *Módulo Analógico Para Um Conversor A/d Por Aproximações Sucessivas*. UnB.

[Rhoads 2010] Rhoads, S. (2010). Plasma-most mips i (tm) opcodes. *Capturado em: http://opencores. org/project, plasma*.

[Santifort 2010] Santifort, C. (2010). Amber arm-compatible core. *OpenCores. org*.

[Smith 1998] Smith, D. J. (1998). *HDL Chip Design: A practical guide for designing, synthesizing and simulating ASICs and FPGAs using VHDL or Verilog*. Doone Publications.

[Taylor 2018] Taylor (2018). The black box system. `https://taylorpearson.me/blackbox/`. acessado em 18/03/2018.

[Texas Instruments 2009] Texas Instruments, T. (2009). Msp430 user guide. Technical report, Tech. Report.

[Tolentino 2018] Tolentino, P. (2018). *Projeto digital da lógica de controle de conversor A/D SAR em tecnologia CMOS*. UnB.

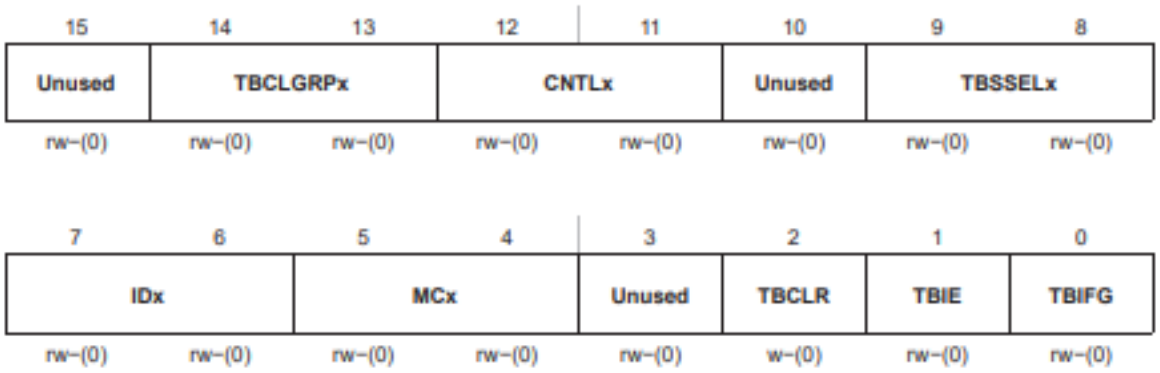
Registros Timer B

Aqui é apresentado os registros do timer B.

Fonte: [Texas Instruments 2009]

Register	Short Form	Register Type	Address	Initial State
Timer_B control	TBCTL	Read/write	0180h	Reset with POR
Timer_B counter	TBR	Read/write	0190h	Reset with POR
Timer_B capture/compare control 0	TBCCTL0	Read/write	0182h	Reset with POR
Timer_B capture/compare 0	TBCCR0	Read/write	0192h	Reset with POR
Timer_B capture/compare control 1	TBCCTL1	Read/write	0184h	Reset with POR
Timer_B capture/compare 1	TBCCR1	Read/write	0194h	Reset with POR
Timer_B capture/compare control 2	TBCCTL 2	Read/write	0186h	Reset with POR
Timer_B capture/compare 2	TBCCR2	Read/write	0196h	Reset with POR
Timer_B capture/compare control 3	TBCCTL3	Read/write	0188h	Reset with POR
Timer_B capture/compare 3	TBCCR3	Read/write	0198h	Reset with POR
Timer_B capture/compare control 4	TBCCTL4	Read/write	018Ah	Reset with POR
Timer_B capture/compare 4	TBCCR4	Read/write	019Ah	Reset with POR
Timer_B capture/compare control 5	TBCCTL5	Read/write	018Ch	Reset with POR
Timer_B capture/compare 5	TBCCR5	Read/write	019Ch	Reset with POR
Timer_B capture/compare control 6	TBCCTL6	Read/write	018Eh	Reset with POR
Timer_B capture/compare 6	TBCCR6	Read/write	019Eh	Reset with POR
Timer_B Interrupt Vector	TBIV	Read only	011Eh	Reset with POR

Timer_B Control Register TBCTL



15	14	13	12	11	10	9	8
Unused	TBCLGRPx		CNTLx		Unused	TBSSELx	
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
IDx		MCx		Unused	TBCLR	TBIE	TBIFG
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	w–(0)	rw–(0)	rw–(0)

Unused Bit 15 Unused

TBCLGRP Bit 14-13 TBCLx group
- 00 Each TBCLx latch loads independently
- 01 TBCL1+TBCL2 (TBCCR1 CLLDx bits control the update)
  TBCL3+TBCL4 (TBCCR3 CLLDx bits control the update)
  TBCL5+TBCL6 (TBCCR5 CLLDx bits control the update)
  TBCL0 independent
- 10 TBCL1+TBCL2+TBCL3 (TBCCR1 CLLDx bits control the update)
  TBCL4+TBCL5+TBCL6 (TBCCR4 CLLDx bits control the update)
  TBCL0 independent
- 11 TBCL0+TBCL1+TBCL2+TBCL3+TBCL4+TBCL5+TBCL6
  (TBCCR1 CLLDx bits control the update)

CNTLx Bits 12-11 Counter Length
- 00 16-bit, $TBR_{(max)}$ = 0FFFFh
- 01 12-bit, $TBR_{(max)}$ = 0FFFh
- 10 10-bit, $TBR_{(max)}$ = 03FFh
- 11 8-bit, $TBR_{(max)}$ = 0FFh

Unused Bit 10 Unused

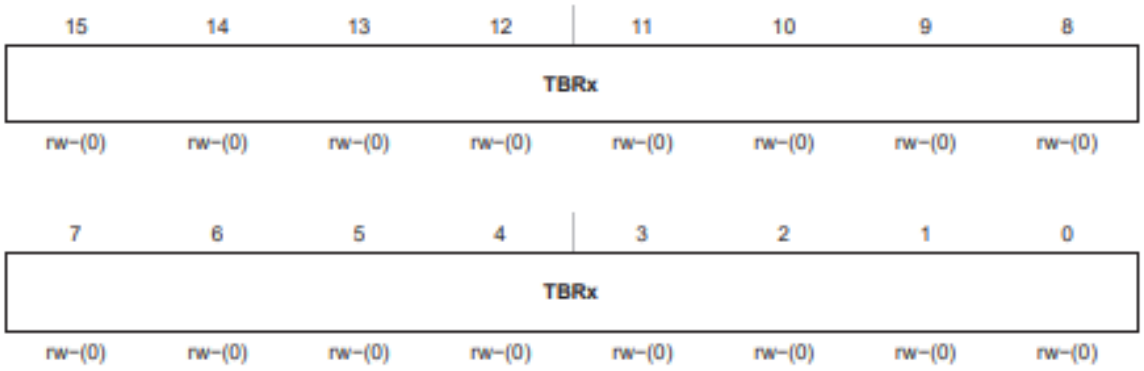
TBSSELx Bits 9-8 Timer_B clock source select.
- 00 TBCLK
- 01 ACLK
- 10 SMCLK
- 11 Inverted TBCLK

IDx Bits 7-6 Input divider. These bits select the divider for the input clock.
- 00 /1
- 01 /2
- 10 /4
- 11 /8

MCx Bits 5-4 Mode control. Setting MCx = 00h when Timer_B is not in use conserves power.
- 00 Stop mode: the timer is halted
- 01 Up mode: the timer counts up to TBCL0
- 10 Continuous mode: the timer counts up to the value set by TBCNTLx
- 11 Up/down mode: the timer counts up to TBCL0 and down to 0000h

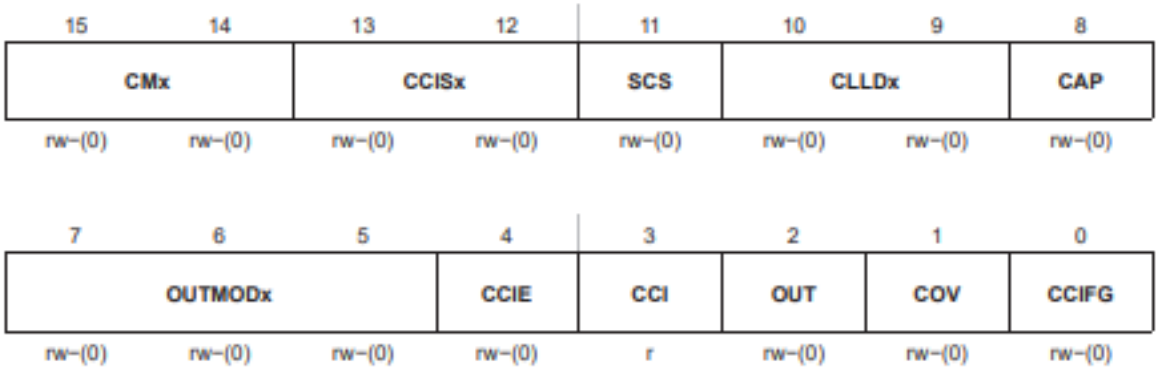
Unused	Bit 3	Unused
TBCLR	Bit 2	Timer_B clear. Setting this bit resets TBR, the clock divider, and the count direction. The TBCLR bit is automatically reset and is always read as zero.
TBIE	Bit 1	Timer_B interrupt enable. This bit enables the TBIFG interrupt request. 0 Interrupt disabled 1 Interrupt enabled
TBIFG	Bit 0	Timer_B interrupt flag. 0 No interrupt pending 1 Interrupt pending

TBR, Timer_B Register



TBRx	Bits 15-0	Timer_B register. The TBR register is the count of Timer_B.

TBCCTLx, Capture/Compare Control Register



CMx Bit 15-14 Capture mode
- 00 No capture
- 01 Capture on rising edge
- 10 Capture on falling edge
- 11 Capture on both rising and falling edges

CCISx Bit 13-12 Capture/compare input select. These bits select the TBCCRx input signal. See the device-specific datasheet for specific signal connections.
- 00 CCIxA
- 01 CCIxB
- 10 GND
- 11 V_{CC}

SCS Bit 11 Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock.
- 0 Asynchronous capture
- 1 Synchronous capture

CLLDx Bit 10-9 Compare latch load. These bits select the compare latch load event.
- 00 TBCLx loads on write to TBCCRx
- 01 TBCLx loads when TBR *counts* to 0
- 10 TBCLx loads when TBR *counts* to 0 (up or continuous mode)
  TBCLx loads when TBR *counts* to TBCL0 or to 0 (up/down mode)
- 11 TBCLx loads when TBR *counts* to TBCLx

CAP Bit 8 Capture mode
- 0 Compare mode
- 1 Capture mode

OUTMODx Bits 7-5 Output mode. Modes 2, 3, 6, and 7 are not useful for TBCL0 because EQUx = EQU0.
- 000 OUT bit value
- 001 Set
- 010 Toggle/reset
- 011 Set/reset
- 100 Toggle
- 101 Reset
- 110 Toggle/set
- 111 Reset/set

CCIE	Bit 4	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0 Interrupt disabled 1 Interrupt enabled
CCI	Bit 3	Capture/compare input. The selected input signal can be read by this bit.
OUT	Bit 2	Output. For output mode 0, this bit directly controls the state of the output. 0 Output low 1 Output high
COV	Bit 1	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0 No capture overflow occurred 1 Capture overflow occurred
CCIFG	Bit 0	Capture/compare interrupt flag 0 No interrupt pending 1 Interrupt pending

TBIV, Timer_B Interrupt Vector Register

15	14	13	12	11	10	9	8
0	0	0	0	0	0	0	0
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	0	0	0	TBIVx			0
r0	r0	r0	r0	r−(0)	r−(0)	r−(0)	r0

TBIVx	Bits 15-0	Timer_B interrupt vector value

TBIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	–	
02h	Capture/compare 1	TBCCR1 CCIFG	Highest
04h	Capture/compare 2	TBCCR2 CCIFG	
06h	Capture/compare 3†	TBCCR3 CCIFG	
08h	Capture/compare 4†	TBCCR4 CCIFG	
0Ah	Capture/compare 5†	TBCCR5 CCIFG	
0Ch	Capture/compare 6†	TBCCR6 CCIFG	
0Eh	Timer overflow	TBIFG	Lowest

† MSP430x14x, MSP430x16x devices only