UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RELÓGIO CMOS DE TEMPO REAL

EDUARDO NEUMANN MORUM SIMÃO
MARCELO REGAL RONZANI

ORIENTADOR: ADSON FERREIRA DA ROCHA
ORIENTADOR: ALEXANDRE ROMARIZ

PROJETO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

BRASÍLIA/DF: JULHO – 2005

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RELÓGIO CMOS DE TEMPO REAL

EDUARDO NEUMANN MORUM SIMÃO
MARCELO REGAL RONZANI

MONOGRAFIA SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA GRADUAÇÃO.

APROVADA POR:

FICHA CATALOGRÁFICA

RONZANI, MARCELO REGAL
SIMÃO, EDUARDO NEUMANN MORUM
Relógio CMOS de Tempo Real [Distrito Federal] 2005
Projeto de Graduação – Universidade de Brasília, Faculdade de Tecnologia,
Departamento de Engenharia Elétrica.
1. Projeto VLSI 2. Processadores RISC
3. VHDL 4. Testabilidade de Sistemas
I. ENE/FT/UNB II. Título (série)

REFERERÊNCIA BIBLIOGRÁFICA

RONZANI, M. R., SIMÃO, E. N. M. (2005). Relógio CMOS de Tempo Real, Projeto de Graduação, Publicação Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 71 p.

CESSÃO DE DIREITOS

AUTORES: Eduardo Neumann Morum Simão, Marcelo Regal Ronzani

TÍTULO: Relógio CMOS de Tempo Real

GRAU/ANO: Graduado/2005

Eduardo Neumann M. Simão

Marcelo Regal Ronzani

AGRADECIMENTOS

Agradeço aos professores Alexandre Ricardo Soares Romariz e Adson Ferreira da Rocha, por terem aceitado nos orientar neste trabalho. Ao professor José Camargo da Costa pelas sugestões e atenção. Ao colega Eduardo Neumann Morum Simão por ter aceitado minha parceria na realização deste projeto final de graduação. Ao colega Wagner Antunes Araújo pela ajuda prestada durante a elaboração do projeto.

Ao meu pai Pedro Ernesto Ronzani e minha mãe Cecília Regal Ronzani, pelo apoio pessoal e paciência. À Poliana Carvalho Cunha pelo incentivo, carinho e apoio. Aos meus irmãos que sempre me apoiaram.

Marcelo Regal Ronzani

Agradeço aos professores Alexandre Ricardo Soares Romariz, pela inestimável prestatividade, e Adson Ferreira da Rocha por nos orientar neste projeto. À decisiva ajuda do colega Edson Mintsu Hung. Aos amigos do CP pelo inestimável apoio ao longo de todo o curso. E a todos os colegas, da UnB e do CJF, e professores que contribuíram para o desenvolvimento deste trabalho.

Agradeço também ao constante apoio de toda minha família. Ao meu pai Caio Morum Simão e minha mãe Maria Celeste Neumann Simão, pelo constante exemplo de vida. Às minhas irmãs e cunhados, Letícia, Melissa e Ricardo, e Patrícia e Raphael, por estarem sempre prontos a ajudar. E à minha querida esposa, Aline, que está sempre ao meu lado me incentivando e me inspirando.

Eduardo Neumann Morum Simão

RESUMO

RELÓGIO CMOS DE TEMPO REAL

Neste trabalho foi projetado e implementado digitalmente um relógio de tempo real.

Tal estrutura integrará um Sistema em Chip (SoC) para comunicação sem fio em um sistema de controle de irrigação. Foi desenvolvido o projeto digital da unidade anteriormente citada utilizando técnicas de projeto orientado a testabilidade. Os módulos foram projetados e simulados utilizando ferramentas do Circuit Maker e simuladores VHDL. Os resultados atenderam às especificações previamente definidas. Ao final, os módulos foram integrados chegando ao resultado final satisfatório para que sejam implementados no projeto elétrico e posterior envio para fabricação.

Foram ainda especificadas algumas rotinas que deverão estar no software para ajuste do relógio.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

INTRODUÇÃO

O constante avanço tecnológico dos semicondutores e o aumento do mercado de dispositivos eletrônicos tem impulsionado o desenvolvimento de sistemas computacionais em um único circuito integrado (C.I.). Tais sistemas são tipicamente compostos de milhões de transistores que englobam *hardware* digital e analógico e são conhecidos como SoC's (*Systems on Chip*). O projeto desse tipo de sistema é algo complexo uma vez que envolve questões como portabilidade, limite de consumo de potência, desempenho, confiabilidade e interferência eletromagnética, entre outras. Apesar da dificuldade inerente ao desenvolvimento de tais circuitos, esse tipo de projeto permite a implementação de novos sistemas e a formação de profissionais capacitados nessa área.

Com essa motivação, foi criado o Instituto do Milênio, que constitui um grande esforço por parte de pesquisadores e da comunidade de microeletrônica e áreas afins para o avanço em projetos de circuitos integrados (CIs). Dentre os objetivos do Instituto do Milênio, podem ser citados a formação de recursos humanos (alunos de iniciação científica, mestres, doutores, pós-doutores e outros), o domínio de processos de fabricação de CI's CMOS, contribuições em algoritmos para ferramentas de CAD avançadas e a experiência em realização de projetos de CI's nas aplicações de RF. Entre os projetos inseridos nesse programa está o "Sistemas em chip, Microssistemas e Nanoeletrônica" (SCMN), que é financiado pelo Ministério da Ciência e Tecnologia (MCT).

Neste projeto, um sistema de comunicação sem fio e de processamento de dados em um único *chip* está sendo desenvolvido através de uma parceria entre UnB, USP, UFSC, UFPE, UFRJ, Unicamp, UFRGS e EMBRAPA. O objetivo de tal trabalho é integrar um sistema de irrigação que controla o volume de água utilizado mantendo a umidade do solo em um nível ótimo. A necessidade hídrica das culturas é determinada por meio da medição da umidade do solo e de dados meteorológicos.

O sistema de irrigação é composto por uma estação-base, estações de campo, e por nós. A estação-base, única na propriedade rural, é a interface entre o usuário e as estações de campo, e concentra as informações oriundas das mesmas. A informação é enviada pelas estações de campo para a estação-base por um link sem fio (vide Figura 1.1), e então é disponibilizada para o usuário, que pode programar o comportamento do sistema após a análise dos dados recebidos.

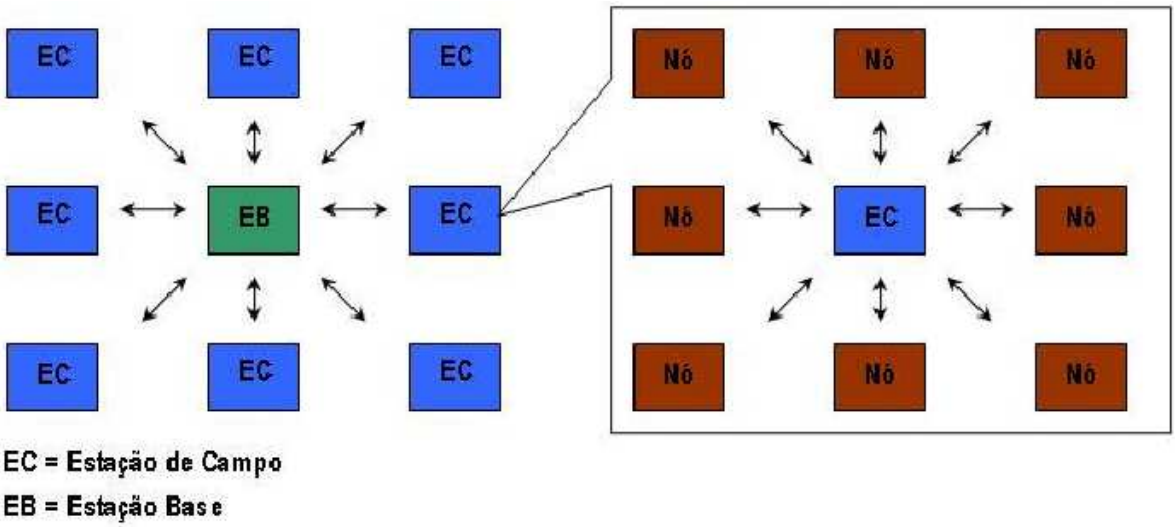


Figura 1.1 - Sistema de irrigação com nós, estações de campo e estação base [COS2003 – alterada]

Os nós monitoram a umidade do solo por meio de um sensor de pressão. Eles têm uma área de cobertura de 100 hectares e são compostos basicamente de um SoC CMOS 0,35 µm (microprocessador RISC, oscilador, interfaces serial e A/D e um transceptor de RF operando entre 915 e 927,75 MHz e consumindo 1 mW), sensor de umidade de solo (tensiômetro), antena, coletor solar, fonte de alimentação, bateria, atuador eletromecânico e programas computacionais. Eles mandam as informações coletadas pelo sensor para as estações de campo por um link sem fio. A Figura 1.2 mostra o arranjo físico do nó em detalhe.

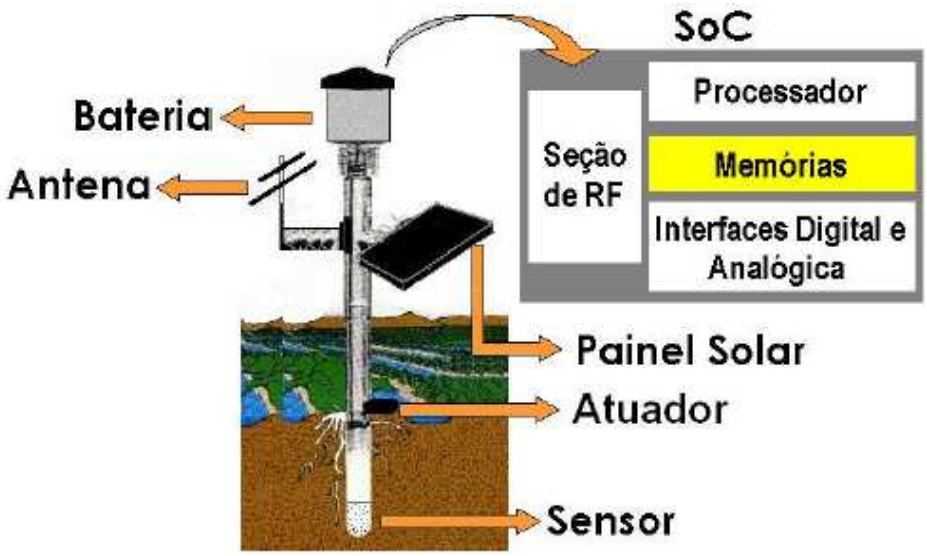


Figura 1.2 - Detalhes de um nó do sistema de irrigação

O sistema que se encontra na placa mostrada no esquema da Figura 1.2 é composto por um microprocessador, memórias RAM e ROM, interfaces analógica e digital, e um

transceptor RF. A arquitetura do microprocessador é a RISC de 16 bits, com *clock* de 10 MHz, alimentação de 3.3 V e consumo de potência de 100 mW. Ele possui uma unidade lógico-aritmética (ULA) operando em ponto fixo, 16 instruções e 16 registradores de 16 bits [BES2004].

A contribuição do presente projeto final está em projetar e implementar o relógio digital de tempo real que compõe o SoC em questão. Foi definido inicialmente que seriam utilizados Flip-Flop tipo D para a implementação dos contadores. Foi definido também que o relógio deveria ter uma precisão de segundos e contar de 00:00:00 até 23:59:59. Todo o projeto é voltado para orientar uma posterior implementação física e integração deste relógio ao Soc.

A importância do relógio para o projeto se dá devido ao fato que o processador ficará inativo durante grande parte de tempo, só sendo ativado nos momentos em que for realizar as medidas necessárias ou se comunicar com a estação base. Isso é necessário para se economizar energia da bateria que alimenta o chip. O relógio deverá então despertar o processador em momentos programados para as medições e também para sua comunicação com a estação base.

No Capítulo 2 foi feita uma revisão bibliográfica mostrando estruturas de sistemas digitais e programação em VHDL. O Capítulo 3 trata da descrição do sistema, com especificações do relógio e de seu funcionamento. Os contadores, que são as unidades básicas do relógio são descritos no Capítulo 4. O relógio propriamente dito está no Capítulo 5, onde são apresentadas também as simulações do mesmo. O Capítulo 6 contém as conclusões do trabalho e na seqüência são apresentados as referências bibliográficas e os apêndices.

REVISÃO BIBLIOGRÁFICA

SISTEMAS DIGITAIS

Flip-flop

A maior parte dos circuitos seqüenciais é composta por flip-flops. Os sistemas digitais típicos usam flip-flops pré-definidos e especificados em circuitos integrados. Esses circuitos contêm células de flip-flops que são na verdade um circuito seqüencial composto por portas lógicas e loops [WAK2000].

Os flip-flops tipo D são largamente utilizados para a implementação de contadores digitais. O funcionamento desses dispositivos é bem simples, o tipo D contém quatro entradas e duas saídas. As entradas são: D, Clock, Set e Reset. As saídas são: Q e Q/. Veja a figura 2.1.

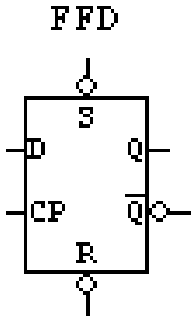


Figura 2.1 - Flip-Flop tipo D

A entrada Clock ou CP é a entrada para o pulso digital que excitará o dispositivo para que ele funcione como um contador. O CP ou CLK como é mais comumente chamado, é importante porque ele é quem permite que o estado das saídas seja alterado de acordo com o dado da entrada. O flip-flop só modifica a saída na borda de subida ou de descida do pulso do relógio, dependendo de seu projeto.

A entrada D é a entrada de dados que se quer manipular.

A entrada Set é assíncrona e, independente do CLK e da entrada D, faz com que a saída Q seja sempre 1 e Q/ sempre 0. A Reset, também de forma assíncrona, deixa Q sempre em 0 e Q/ em 1. Essas duas entradas nunca podem estar habilitadas ao mesmo tempo. Uma das duas deve estar habilitada sozinha ou ambas desabilitadas, com o risco de não funcionamento do dispositivo.

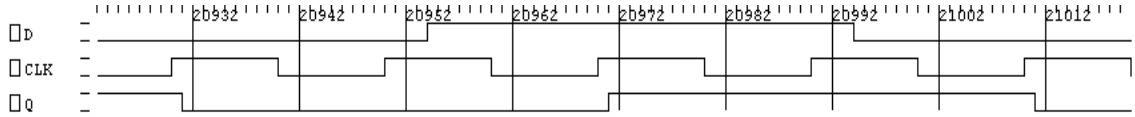


Figura 2.2 – Formas de onda do Flip-Flop D

Este Flip-Flop faz a transição do dado na borda de subida, como mostra a figura 2.2. Observe que a saída Q só muda quando o pulso do relógio passa pela borda de subida, caso contrário o FF mantém o nível anterior.

Contadores síncronos

Os contadores síncronos são circuitos digitais seqüenciais que podem ser utilizados como temporizadores. O módulo de um contador é o número de estados que ele apresenta em um ciclo. Um contador com *m* estados é chamado de contador de módulo-m. O contador mais utilizado é o contador binário de n-bit. Esse contador contém *n* flip-flops e 2^n estados [WAK2000].

Os contadores são formados por flip-flops em cascata, onde os níveis lógicos das entradas síncronas são funções Booleanas das saídas do flip-flop e das entradas de controle. As entradas podem conter portas lógicas para que funcione de acordo com a contagem estabelecida. Essas portas lógicas auxiliam para que, em um mesmo ciclo, todos os n-bits do contador sejam processados e apresentados à saída do flip-flop minimizando os atrasos dos flip-flops. Desta forma teremos uma máquina de estados, onde o estado seguinte dependerá do atual estado, e que ainda dependeu do estado anterior a ele.

A partir de contadores síncronos pode-se realizar funções lógicas e implementar diversos circuitos digitais como relógios digitais de tempo real.

A figura 1 mostra um contador implementado com Flip-Flop D que conta até 2.

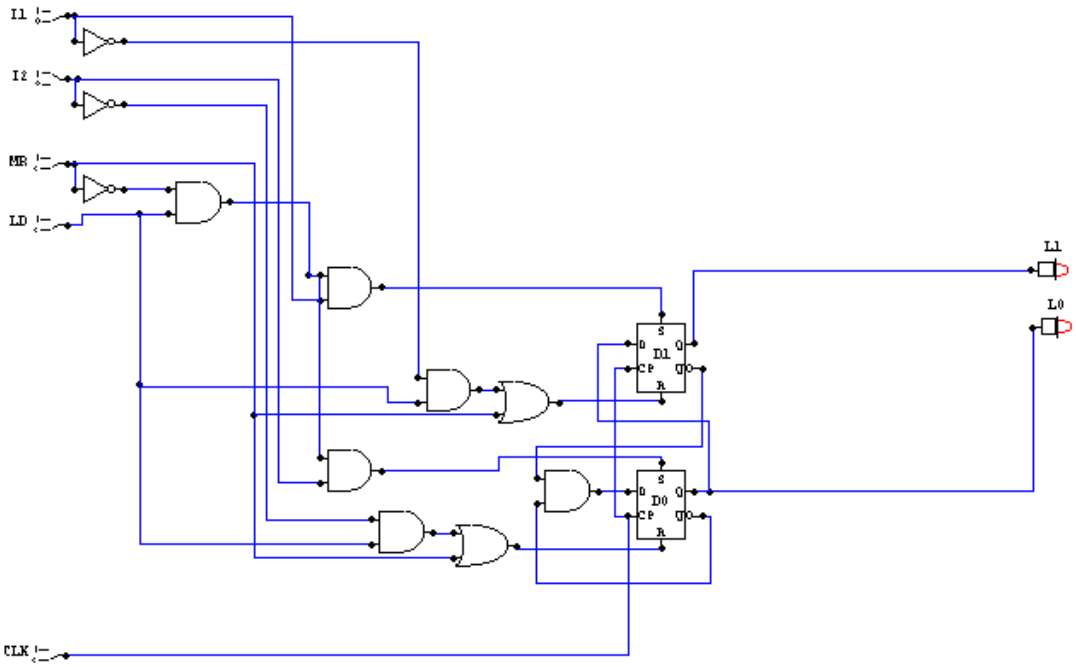


Figura 2.3 – Contador de 0 a 2

Este contador contém um *Master Reset* (MR) que zera os dois FF's e um bit LD que habilita o ajuste do contador por meio das entradas I1 e I2. Há ainda a entrada CLK onde será conectado o pulso do *clock*. L0 e L1 são os as saídas do contador.

1.1.1. Relógio de Tempo Real

Um relógio de tempo real é uma estrutura lógica que utiliza contadores. Esses contadores são dispostos de tal forma a comporem os segundos, os minutos e as horas do relógio.

Utilizando contadores decimais 74LS192, foi possível que um primeiro teste fosse feito a respeito da lógica do funcionamento em cascata do relógio. Portas lógicas também foram utilizadas a fim de interromper a contagem decimal de um contador para que o mesmo contasse até 5, por exemplo, no contador de 0 a 5 das dezenas dos minutos e segundos [REL2001].

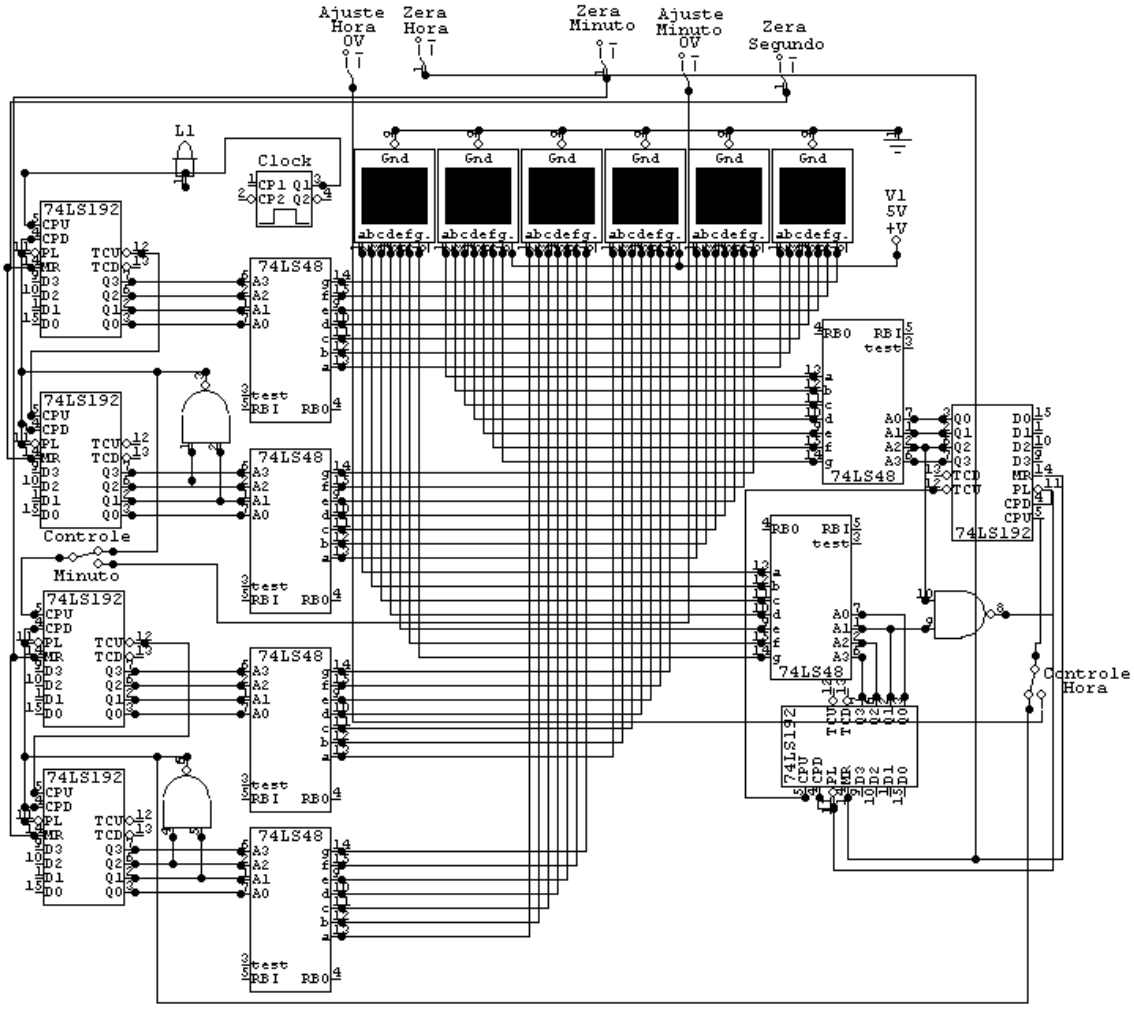


Figura 2.4 – Relógio implementado com contadores decimais 74LS192

PROGRAMAÇÃO EM VHDL

A capacidade cada vez maior de integração de circuitos e o conseqüente aumento da quantidade de transistores que é possível implementar em uma pastilha de silício é conseqüência do avanço tecnológico na fabricação de circuitos integrados e principalmente de técnicas de projeto cada vez mais poderosas [ASH2002].

Devido à grande complexidade dos circuitos implementados, sua compreensão e projeto só é possível usando técnicas de abstração e hierarquização do circuito. Isto é, o projetista concentra-se em uma etapa por vez. Ao aumentar o nível de abstração é possível pensar apenas na descrição do circuito, ou seja, focar-se na função do circuito protelando sua implementação. Já com uma visão hierárquica do circuito é possível analisar apenas um nível de cada vez, ocultando os outros.

Uma importante ferramenta para o desenvolvimento da topologia de um circuito utilizando estas técnica de projeto é a linguagem VHDL. Esta é a sigla de VHSIC *Hardware Description Language* (VHSIC linguagem de descrição em Hardware) onde VHSIC é outra sigla que significa *Very High Speed Integrated Circuits* (Circuitos Integrados de Velocidade Muito Alta). É uma linguagem utilizada para descrição de sistemas digitais de grande complexidade. Envolve diversos aspectos do projeto, tais como a documentação, verificação e síntese de sistemas digitais complexos.

A linguagem VHDL surgiu em meados dos anos 80 quando o departamento de defesa norte-americano e o IEEE patrocinaram o desenvolvimento de uma linguagem de descrição de Hardware que teve como objetivo o desenvolvimento de circuitos integrados de alta velocidade. Esta linguagem foi desenvolvida e tornou-se um padrão IEEE [ASH2002].

Embora esta seja uma linguagem parecida com outras linguagens de programação de software convencionais, existem algumas importantes diferenças a serem destacadas. Uma HDL (*Hardware Description Language*) é fundamentalmente paralela, ou seja, os comandos que correspondem a portas lógicas são executados paralelamente, assim que a entrada é alterada. Um programa em HDL simula o comportamento de um sistema físico, usualmente digital. Ele também considera especificações de tempo como atraso de porta assim como descreve um sistema com diferentes interconexões de componentes.

Níveis de Abstração

Um sistema digital pode ser representado em diferentes níveis de abstração. Podemos, por exemplo, descrevê-lo em função de seu comportamento, em função da disposição e interconexões entre portas lógicas ou ainda entre elementos de circuito (transistores, resistores, capacitores...).

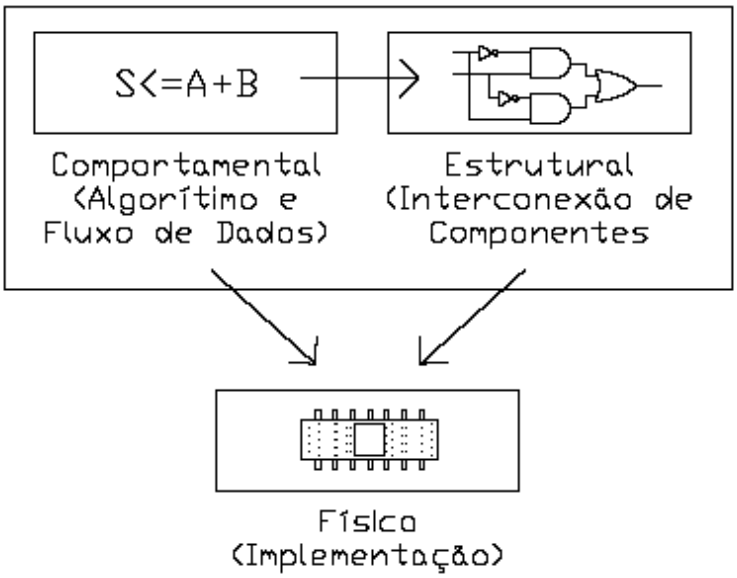


Figura 2.5 – Níveis de abstração

O mais alto nível de abstração é o comportamental onde o sistema é descrito em termos de sua função, não importando sua implementação. É especificado apenas o que o sistema deve realizar, uma relação entre as entradas e as saídas representada, por exemplo, por expressões *booleanas* ou simplesmente por um algoritmo.

Considerando como exemplo um circuito que sinaliza quando os ocupantes de um carro estão sem cinto ou quando a porta não está corretamente fechada ao se introduzir a chave na ignição.

Uma descrição comportamental deste sistema utilizando uma expressão *booleana* pode ser a seguinte:

Sinal = Ignição *AND* (Porta_aberta *OR* Sem_cinto)

Podemos ainda definir este sistema em nível estrutural utilizando portas lógicas conectando-as de forma a implementar a função desejada. Esta descrição se aproxima mais da realização física do circuito, já que temos disponíveis, portas lógicas implementadas fisicamente. Porém esta já é uma abstração, já que portas lógicas são construídas com a conexão de elementos de circuito.

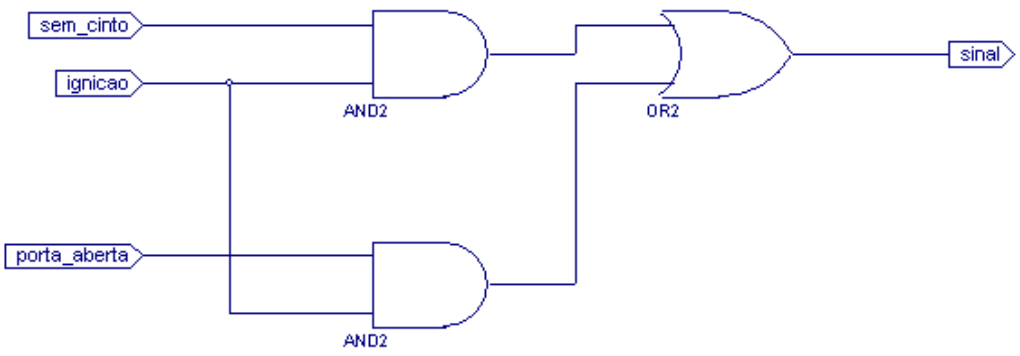


Figura 2.6 – Representação estrutural do circuito sugerido

Utilizando a linguagem VHDL podemos descrever tanto o nível comportamental, quanto o estrutural. Podemos ainda obter o estrutural a partir da descrição comportamental ou o contrário.

Hierarquização

A definição de uma hierarquia no circuito é uma forma de abstração. Desta forma o circuito é dividido em módulos de diferentes níveis hierárquicos. Por exemplo, ao se projetar um flip-flop podemos estabelecer alguns níveis de hierarquia que facilitarão o entendimento.

Podemos definir o nível hierárquico mais baixo como as portas lógicas. Uma vez entendido este nível podemos representá-los por blocos e interconecta-los formando uma nova função; um *latch*. O *latch* será um novo nível hierárquico e por sua vez será representado por blocos e interconectado de forma a obtermos um flip-flop. Teremos então o mais alto nível hierárquico, representado por um bloco que realiza as funções desejadas de um flip-flop.

É importante notar que uma vez em um nível, não estamos interessados na estrutura e nem na implementação de outros níveis. Apenas interessa como as entradas se relacionam com as saídas.

Em VHDL, por meio do conceito de entidade de projeto são definidos os diferentes níveis de hierarquização. Uma entidade de projeto é a principal abstração de hardware em VHDL. Nos permite a separação efetiva da interface e da função, permitindo uma decomposição hierárquica.

Retomando o exemplo do item anterior, se considerarmos que o sinal só funciona para a porta e o cinto do motorista, podemos projetar um circuito que considere também os outros 3 passageiros utilizando a técnica de hierarquização.

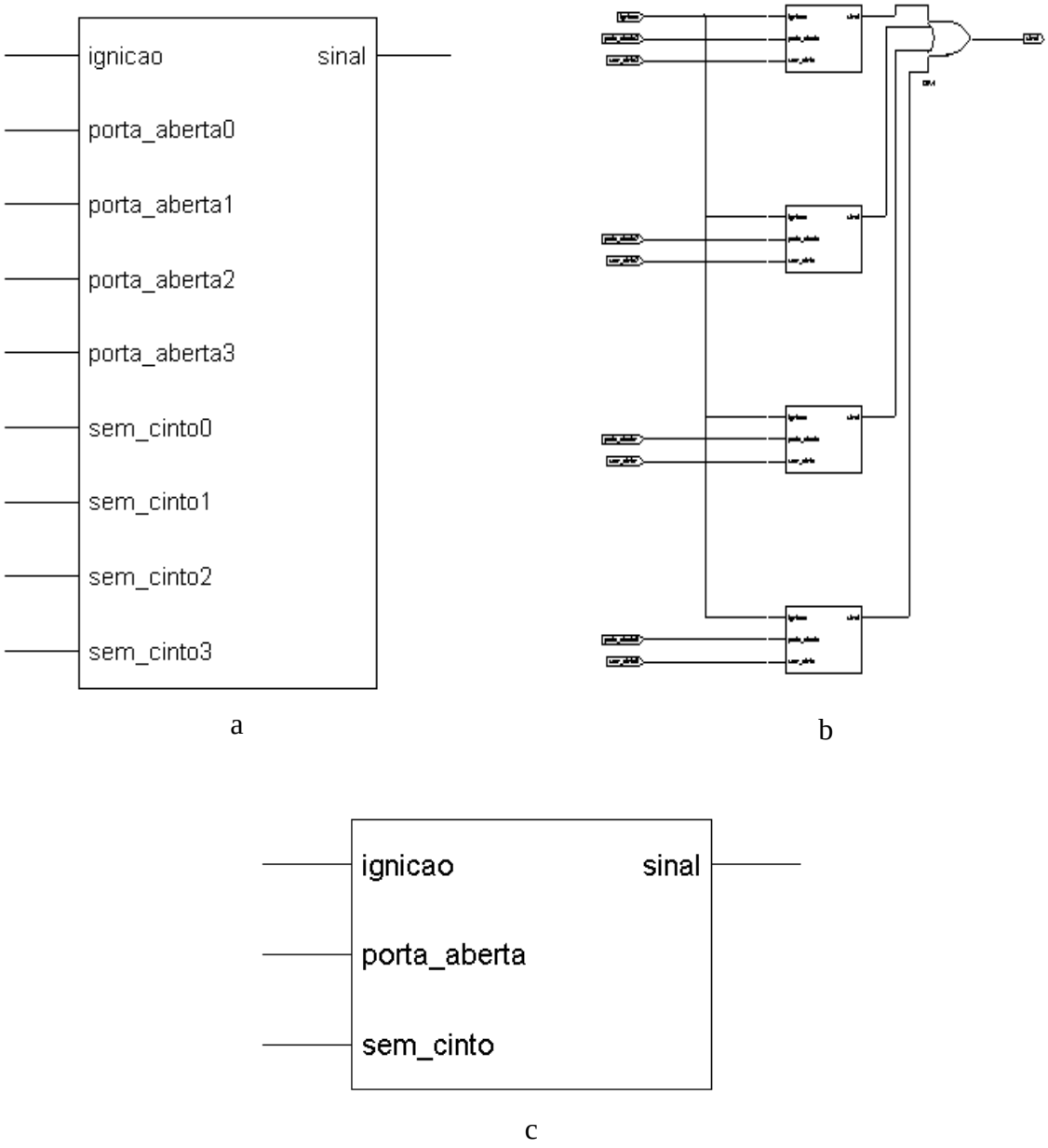


Figura 2.7 – Níveis hierárquicos

Podemos observar na figura 2.6.a, o nível hierárquico mais alto. Tudo o que temos é um bloco com as entradas e a saída não importando o que existe dentro e sim a relação entre as entradas e a saída. Já na figura 2.6.b podemos observar a composição do bloco em ‘a’. Vemos que ele é composto por quatro circuitos do exemplo do item anterior ilustrado na figura 2.6.c, que passa a ser um nível hierárquico abaixo, que por sua vez possui níveis

menores. Da figura 2.6.c para a 2.6.a subimos um nível hierárquico por meio das conexões apresentadas na figura 2.6.b.

Estruturas Básicas VHDL

A linguagem VHDL suporta, basicamente, três tipos de descrição. São elas a descrição estrutural, a descrição por fluxo de dados e a descrição comportamental. Estas são diferentes formas de descrever a função de um determinado bloco.

Como já visto, dividir o projeto em blocos é uma estratégia básica para facilitar a compreensão e a manutenção do mesmo a medida que se hierarquiza o sistema. Estes blocos são conectados para construir um sistema completo. Por meio das diferentes formas de descrição, podemos definir não só a função de cada bloco mas suas interconexões, através de um programa VHDL, os quais são projetados e testados individualmente.

Cada um destes blocos é denominado entidade (*entity*) e na declaração de entidade, é definida a interface do sistema. Podemos nesta parte do código, declarar portas de entrada (*in*), saída (*out*) ou entrada e saída (*inout*) do bloco.

Aproveitando o exemplo inicial de sinalização do cinto e da porta do carro, sua declaração de entidade seria a seguinte.

```
entity ex_carro is
     port (ignicao,porta_aberta,sem_cinto: in std_logic;
          sinal: out std_logic);
end ex_carro;
```

Assim, estão definidas as entradas e saídas do bloco apresentado na figura 2.6.c. A primeira linha indica a definição de uma nova entidade, chamada de `ex_carro`. A última define o fim da definição. O conteúdo entre a primeira e a última linha define nesta ordem, o nome, o modo e o tipo de cada porta da entidade. O final de cada linha é marcado por ponto e virgula. O modo da porta indica se é *in*, *out* ou *inout*. E o tipo pode ser `std_logic` ou `bit` que podem assumir valores lógicos 0 e 1. Podem ainda ser vetores ou matrizes (`std_logic_vector`).

Em seguida, no corpo do código, a segunda parte da descrição do bloco é a definição da operação do mesmo. É nesta parte, denominada declaração de arquitetura, que é feita um dos três tipos de descrição citados.

Descrição fluxo de dados

A descrição de fluxo de Dados (*data flow description*) é uma descrição que se aproxima de uma expressão booleana. Isto porque a linguagem VHDL reconhece componentes primitivas como portas and, or e inversora.

Assim como ilustrado no exemplo, os circuitos são descritos pela indicação de como as entradas e saídas de portas primitivas são conectadas, especificando como os sinais fluem pelo circuito. Continuando com o exemplo do carro, teríamos a seguinte declaração de arquitetura com descrição de fluxo de dados.

```
entity ex_carro is
     port (ignicao,porta_aberta,sem_cinto: in std_logic;
                sinal: out std_logic);
end ex_carro;

architecture Behavioral of ex_carro is
signal b1,b2: std_logic;
begin

b1 <= ignicao and porta_aberta;
b2 <= ignicao and sem_cinto;
sinal <= b1 or b2;

end Behavioral;
```

A primeira parte do código é a entidade, que já foi explicada anteriormente. A segunda parte é a declaração de arquitetura. A primeira linha define o início da declaração. Em seguida temos o termo `BEGIN` que indica o início da descrição. A última linha finaliza a declaração. Entre a primeira linha e o `BEGIN` foram definidos sinais intermediários do circuito. Estes não têm definição de modo pois são sinais internos transitórios e não portas. O corpo da descrição fica entre o `BEGIN` e a última linha. Ele descreve a operação interna do projeto.

A declaração de atribuição de sinal descreve como os dados fluem dos sinais do lado direito para o esquerdo. O operador <= indica uma relação entra sinais e não uma atribuição como em linguagens de operação.

A primeira atribuição de sinal no exemplo apresentado mostra que os dados provêm dos sinais de `ignição` e `porta_aberta`, fluindo por uma porta `and` para determinar o valor do sinal intermediário `b1`. A outra atribuição de sinal mostra que os dados provêm

dos sinais `de ignição` e `sem_cinto`, fluindo por uma porta `and` para determinar o valor do sinal intermediário `b2`. Na última atribuição, os dados dos sinais intermediários `b1` e `b2` fluem pela porta `or` e determinam o sinal de saída sinal.

O lado direito do operador é chamado de expressão e sua avaliação determina o seu valor. A avaliação é realizada substituindo-se os valores dos sinais na expressão e computando o resultado de cada operador na expressão.

Descrição comportamental

Diferentemente dos outros dois tipos de descrição, a comportamental não reflete necessariamente a forma como o projeto é implementado. O projeto é tido como uma caixa preta onde a relação de transferência entrada/saída é detalhada mas a forma como funciona é irrelevante.

A descrição comportamental é usada normalmente para modelar componentes complexos cuja modelagem seria difícil nas outras duas abordagens. Este pode ser o caso, por exemplo, da simulação de operação de um projeto customizado que é conectado a um dispositivo comercial. Neste caso, a operação interna do dispositivo é irrelevante se é sabido o seu funcionamento externo.

Este tipo de descrição é feito no corpo da declaração de arquitetura por meio da estrutura *process.* O conteúdo desta estrutura pode incluir comandos como os encontrados em linguagens de programação comuns (*case, if*...) para determinar as relações entre os sinais de entrada e de saída, ou uma forma de atribuição de sinal semelhante à descrição de fluxo de dados. Este tipo de descrição é poderoso, porém pode não corresponder com a implementação em hardware desejada.

Seguindo o mesmo exemplo do carro, uma descrição comportamental seria da seguinte forma.

```
entity ex_carro is
     port (ignicao,porta_aberta,sem_cinto: in std_logic;
                sinal: out std_logic);
end ex_carro;

architecture Behavioral of ex_carro is

begin

sinal_alerta:process(ignição,porta_aberta,sem_cinto)
```

```
   variable b1,b2: std_logic:= ‘0’;

   begin

    b1 := ignicao and porta_aberta;
    b2 := ignicao and sem_cinto;
    sinal <= b1 or b2;
end process sinal_alerta;

end Behavioral;
```

O rótulo `sinal_alerta:` é opcional e é usado para definir um nome ao processo. A lista de sinais entre parênteses é denominada de *sensitivity list* (lista de sensibilidade). Ela ativa o processo quando um dos sinais presentes muda de valor. Esta lista se faz necessária porque a declaração de processo não contém nenhuma indicação de características estruturais, ficando indefinido quando o processo atualizaria as saídas. O corpo do processo esta entre as palavras-chave BEGIN e END e é aí que ele será avaliado.

As variáveis podem ser usadas apenas em processos. Uma variável se comporta como a de uma linguagem de programação comum e serve para armazenar dados. Elas são modificadas através de atribuições como no exemplo.

Podemos ainda descrever o comportamento deste mesmo circuito utilizando declarações seqüenciais. Esta declarações só podem ser utilizadas no corpo do processo e como o próprio nome diz, são executadas sequencialmente, na ordem definida do inicio para o final das declarações no processo.

```
entity ex_carro is
     port (ignicao,porta_aberta,sem_cinto: in std_logic;
                sinal: out std_logic);
end ex_carro;

architecture Behavioral of ex_carro is

begin

sinal_alerta:process(ignição,porta_aberta,sem_cinto)
   variable b1,b2: std_logic:= ‘0’;

   begin

    if ignição = ‘1’ and porta_aberta = ‘1’ then
     b1 := ‘1’;

    elsif ignição = ‘1’ and sem_cinto = ‘1’ then
```

```
     b2 := ‘1’;
    end if;

    sinal <= b1 or b2;

end process sinal_alerta;

end Behavioral;
```

Esta declaração `if` tem duas partes principais, a condição e o corpo da declaração, onde a condição é qualquer expressão booleana. O corpo do processo irá verificar se a chave esta na ignição e se a porta esta aberta. Se for verdade a variável `b1` será atualizada para ‘1’. Se não, o processo verificará se a chave está na ignição e se o passageiro está sem cinto. A variável `b2` será atualizada da mesma forma e, finalmente, se uma das variáveis forem ‘1’ o sinal é ativado.

Descrição estrutural

A descrição estrutural é a descrição utilizada para interligar blocos. Como as portas lógicas também podem ser vistas como blocos, esta descrição também pode definir interconexões básicas.

Uma vez definidos os blocos construtivos do projeto, seja por descrição de fluxo de dados, comportamental ou estrutural, pode-se combiná-los para formar outros projetos mais complexos.

Existe uma importante distinção entre uma entiddade, um componente e uma instância de componente em VHDL. A entidade descreve a interface, o componente descreve a interface de uma entidade que será utilizada como uma instância (ou sub-bloco), e a instância do componente é uma cópia em separado de que componente que foi conectado às outras partes e sinais.

Observando a figura 2.6, podemos identificar a entidade na letra a, o componente na letra c e a instância do componente na letra b. O código desta figura exemplifica a descrição estrutural.

```
entity sinal_completo is
     port
(ignicao,porta_aberta0,sem_cinto0,porta_aberta1,sem_cinto1:
in std_logic;
```

```
     porta_aberta2,sem_cinto2,porta_aberta3,sem_cinto3:    in
std_logic;
               sinal: out std_logic);
end sinal_completo;

architecture Behavioral of sinal_completo is
component ex_carro
     port (ignicao,porta_aberta,sem_cinto: in std_logic;
               sinal: out std_logic);
end component;

signal s0,s1,s2,s3: std_logic;

begin
U0: ex_carro port map (ignicao,porta_aberta0,sem_cinto0,s0);
U1: ex_carro port map (ignicao,porta_aberta1,sem_cinto1,s1);
U2: ex_carro port map (ignicao,porta_aberta2,sem_cinto2,s2);
U3: ex_carro port map (ignicao,porta_aberta3,sem_cinto3,s3);

sinal <= s0 or s1 or s2 or s3;

end Behavioral;
```

Nesta estrutura podemos observar a declaração de entidade, obrigatória em todos os tipos de descrição, e a descrição estrutural na declaração de arquitetura. A estrutura component, presente neste tipo de descrição é a principal responsável pela integração entre os blocos. Ao declarar o ex_carro nesta estrutura, ela se torna disponível no corpo da descrição.

No corpo da descrição estrutural (entre `BEGIN` e `END`), é declarado a instância (ex.: U0:), o componente que será usado (ex_carro) seguido do comando *port map*. (mapa de portas) que descreve como o componente será ligado. Para definir basta especificar, dentro dos parêntesis, na mesma ordem que as portas do componente foram declaradas, quais sinais serão ligados nas entradas e saídas do componente.

A descrição estrutural do projeto é simplesmente uma descrição textual de um diagrama. Uma lista de componentes e suas conexões, geralmente é chamada de *netlist*. A descrição estrutural de um projeto em VHDL é uma forma de especificar uma *netlist*.

Modelo de Atraso

As simulações efetuadas em VHDL são originalmente funcionais, pois modela somente as funções de projeto sem considerações de temporizações. Para realizar uma simulação de temporização deve-se modelar os atrasos interno presentes em circuitos reais.

Dois modelos de atraso do VHDL são o de atraso inercial e o de transporte de atraso. O modelo de atraso inercial pode ser utilizado adicionando a palavra *AFTER.* por exemplo, para considerar um atraso de porta no exemplo do item 2.2.1, devemos adicionar a condição *after*.

```
entity ex_carro is
     generic (atraso_or: time:= 2 ns);
     port (ignicao,porta_aberta,sem_cinto: in std_logic;
               sinal: out std_logic);
end ex_carro;

architecture Behavioral of ex_carro is
signal b1,b2: std_logic;
begin

b1 <= ignicao and porta_aberta after 1 ns;
b2 <= ignicao and sem_cinto after 1 ns;
sinal <= b1 or b2 after atraso_or;

end Behavioral;
```

O tempo de atraso pode ser definido diretamente como é o caso dos atrasos das portas and, ou definidos através de uma variável genérica declarada no corpo da entidade. A vantagem de usar uma variável declarada é de poder alterar facilmente o valor do atraso em todas as portas definidas no código em apenas um lugar.

O modelo de transporte com atraso, além da palavra *after*, tem a palavra *transport* inserida da seguinte forma.

```
entity ex_carro is
     generic (atraso_or: time:= 2 ns);
     port (ignicao,porta_aberta,sem_cinto: in std_logic;
               sinal: out std_logic);
end ex_carro;

architecture Behavioral of ex_carro is
signal b1,b2: std_logic;
begin
b1 <= transport ignicao and porta_aberta after 1 ns;
b2 <= transport ignicao and sem_cinto after 1 ns;
sinal <= transport b1 or b2 after atraso_or;
```

```
end Behavioral;
```

A diferença entre os dois modelos é constatada no caso de ocorrer uma mudança da entrada de 0 para 1 e de 1 para 0, ou ao contrário, em um período inferior ao tempo de atraso. Neste caso, no modelo inercial não ocorrerá alteração na saída e no modelo com transporte de atraso, a saída será alterada após o atraso e voltará à situação inicial após o mesmo período de atraso.

DESCRIÇÃO DO SISTEMA

Na construção do relógio digital, as máquinas de estados foram implementadas em blocos de contadores de 0 a 9, 0 a 5 e 0 a 2.

Ligados em cascata para que funcionem como um relógio, dois contadores, um de 0 a 5 e outro de 0 a 9, formam a dezena e unidade dos segundos respectivamente, outros dois contadores, um de 0 a 5 e outro de 0 a 9, formam a dezena e unidade dos minutos respectivamente e dois contadores, um de 0 a 2 e outro de 0 a 9, formam a dezena e unidade das horas respectivamente. Observa-se que a dezena das horas contém um contador que vai até 2, pois este relógio conta de 00:00:00 até 23:59:59.

Em seu funcionamento, quando o contador da unidade dos segundos atingir 9, ele irá reiniciar seu ciclo e ao mesmo tempo irá incrementar uma dezena ao contador da dezena dos segundos. Quando o contador da dezena dos segundos atingir 5, ele irá reiniciar o seu ciclo a partir do zero e incrementará uma unidade ao contador da unidade dos minutos. Assim, como em um relógio este ciclo se repetirá varrendo todas as vinte e quatro horas do dia.

Há ainda controles externos para que o relógio seja ajustado de acordo com a hora local. Esse ajuste pode ser feito zerando completamente o relógio e habilitando um bit que irá dar acesso aos bits de ajuste das horas, minutos e segundo.

O Relógio de tempo real tem a precisão de segundos. De acordo com o livro branco o relógio despertará o processador a cada 15 minutos para a coleta de medidas e enviará a cada 1 hora esses dados para a estação base. O relógio possui uma função “*load*” para o ajuste das horas e um “*Master Reset*” para zerar os contadores

Para que os devidos ajustes sejam feitos, serão necessários os seguintes bits:

- 1 bit para *Master Reset* zerar todos os contadores (horas, minutos e segundos)
- 1 bit para LD, usado para ajuste das horas.
- 2 bits para o ajuste da dezena das horas (contador de 0 a 2)
- 4 bits para o ajuste da unidade das horas (contador de 0 a 9)
- 3 bits para o ajuste da dezena dos minutos (contador de 0 a 5)
- 4 bits para o ajuste da unidade dos minutos (contador de 0 a 9)
- 3 bits para o ajuste da dezena dos segundos (contador de 0 a 5)

- 4 bits para o ajuste da unidade dos segundos (contador de 0 a 9)

Serão necessários ao todo 22 bits para se ter total controle sobre o relógio, divididos em duas palavras de 16 bits conforme especificação dos registradores.

Seria interessante que o *software* de acesso ao relógio contivesse as seguintes rotinas:

- Zerar o relógio.
- Habilitar através do bit LD o ajuste completo das horas. Para tanto, deverá também fornecer a seqüência de bits que corresponderá à hora atual proveniente da estação base, e em seguida o bit LD deverá ser desabilitado para que o relógio volte à sua contagem.
- Ajustar os momentos em que o processador irá despertar. Isso é feito acrescentando 15 minutos ao valor atual e armazenando esse valor a um registrador que fará a comparação com a hora do relógio a cada momento, gerando um aviso quando estes valores forem iguais.
- Utilizar um contador que incrementará uma unidade a cada despertar do processador. Quando este atingir o valor 4, ou seja, 60 minutos, o processador irá efetuar a coleta de dados e a comunicação com a estação base para transferi-la os dados. Após isto, deverá zerar o contador e os registros de dados coletados para que fiquem livres para novas coletas.

CONTADORES

O relógio conta de zero a vinte e quatro horas com precisão de um segundo. Para tanto, ele precisa de oito dígitos para registrar a hora o minuto e o segundo.

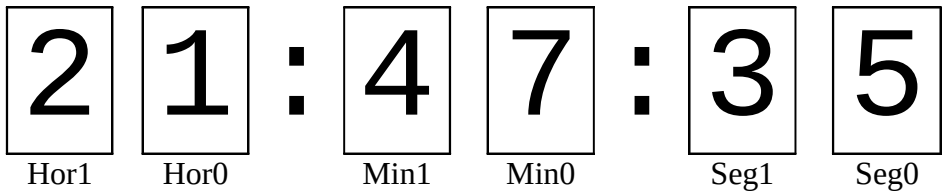


Figura 4.1 – Dígitos do relógio

Para cada dígito, temos um contador. Para o dígito de dezena de hora (Hor1) necessitamos de um contador de 0 a 2. Para o dígito de unidade de hora (Hor0), de um contador de 0 a 9. Para as dezenas de minuto e segundo (Min1 e Seg1), será usado um contador de 0 a 5 em cada. Finalmente, para os dígitos de unidade de minuto e segundo (Min0 e Seg0), serão usados mais dois contadores de 0 a 9.

Assim, serão necessários os seguintes contadores: um de 0 a 2, dois de 0 a 5 e três de 0 a 9. Este capítulo relata o projeto destes contadores, explicando o funcionamento de cada um.

Porém, antes de iniciar o projeto deste nível hierárquico, devemos esclarecer os sub-blocos utilizados nesta etapa do projeto. São estes as portas lógicas AND e OR de duas e de três entradas e a inversora, além do Flip-flop D com set e reset.

UNIDADES BÁSICAS

Considerando as portas lógicas unidades elementares da teoria de circuitos digitais, não cabem aqui muitas delongas a respeito destes dispositivos. Serão utilizadas portas *and* e *or* de duas e três entradas além de portas inversoras.

Já o Flip-flop, apesar de também ser um elemento básico, necessita de uma especificação mais detalhada devido à grande variedade destes dispositivos. Não basta dizer que será utilizado um Flip-flop, pois temos diferentes tipos como D, JK, Toggle, dentre outros. Não basta ainda especificar o tipo, pois cada tipo tem diferentes configurações de entrada e saída.

Inicialmente, qualquer tipo de flip-flop, desde que contenha entradas assíncronas, poderia ser utilizado para se projetar teoricamente os contadores desejados. Porém, visando à implementação física deste projeto, parâmetros além dos teóricos como custo, área ocupada na pastilha de silício e outros devem ser considerados. Diante de uma pesquisa e observação de circuitos semelhantes, foi constatado que os flip-flops mais indicados para estes circuitos são os tipo D e JK.

Utilizando técnicas de sistemas digitais e mapas de Karnaugh para a simplificação das funções que relacionam os flip-flops na implementação dos contadores, foi observado que seriam necessárias menos portas lógicas na implementação com o flip-flop JK. Em contrapartida, considerando a área a ser ocupada pelo circuito final, o flip-flop escolhido foi o tipo D já que sua implementação ocupa cerca de 2/3 da área necessária para implementação do flip-flop JK com as mesmas configurações. Esta economia na área do circuito de implementação do flip-flop tipo D compensa, com vantagem, as portas lógicas que serão usadas a mais para a implementação dos contadores.

As funções de Master Reset e ajuste de hora são independentes do clock, necessitando para sua implementação de entradas assíncronas. Assim, o flip-flop utilizado deve ter entradas do tipo Set e Reset.

As demais configurações do flip-flop não são fatores limitantes do projeto, e por isso, foram definidas de acordo com o que já estava desenvolvido na biblioteca comum do Projeto Milênio no CADENCE. Desta forma, a implementação do flip-flop não foi objeto deste trabalho, já que foi possível conciliar as necessidades deste projeto do relógio com um dispositivo já desenvolvido por outros colegas em outros estágios do trabalho.

Com isto a regularidade do projeto como um todo foi ainda favorecida, já que o flip-flop utilizado nesta parte do projeto é o mesmo utilizado em outras etapas.

Flip-flop D					
Entradas				Saídas	
CLK	Set	Reset	D	Qn+1	QNn+1
X	X	1	X	0	1
X	1	0	X	1	0
↑	0	0	X	Qn	QNn
↓	0	0	0	0	1
↓	0	0	1	1	0

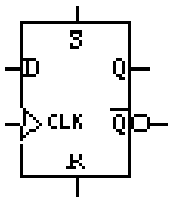


Figura 4.2 – Flip-flop D com SET/RESET

Assim, foi definida a configuração completa do flip-flop a ser utilizado. Será um flip-flop tipo D com SET e RESET ativos em alta, o clock ativo no flanco de decida e saída normal (Q) e invertida (QN). Considerando que os níveis ativos das entradas não implicam em profundas alterações na estrutura do projeto, em estágios iniciais estas especificações não foram observadas com rigor.

O flip-flop é a unidade básica de memória, portanto para armazenar a informação de cada bit de um contador será utilizado um flip-flop.

Como é pretendida uma conexão entre os contadores a fim de implementar o circuito do relógio, com exceção do dígito mais significativo, será necessário que os contadores tenham uma saída que indique o fim de cada ciclo e o início de um novo, quando a maquina de estados passa do último para o primeiro estado. Esta saída, que tem por finalidade excitar o clock do contador do dígito imediatamente mais significativo, será chamada de carry.

CONTADOR DE 0 A 2

O contador 0-2 é de módulo 3 (m=3). Dos contadores a serem utilizados, este é o mais simples. Para implementá-lo são necessários apenas dois bits (n=2). Conseqüentemente, são necessários dois flip-flops.

Este contador tem cinco entradas e apenas as duas saídas que indicam, em binário, o estado em que está o contador. A saída carry, comum aos outros contadores, não precisa estar presente neste, já que ele é utilizado para implementar apenas o dígito mais significativo do relógio.

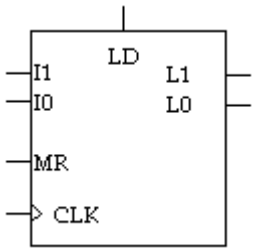


Figura 4.3 – Representação do contador 0 a 2

Das cinco entradas, duas são para os bits de dados no caso de ajuste assíncrono da saída. São elas as entradas I1 e I0. Estas entradas são controladas pela entrada de controle

LD que, quando ativada, coloca as saídas de dados L1 e L0 no mesmo estado de suas respectivas entradas, de forma assíncrona. Tanto as entradas de dados quanto as saídas podem ser representadas como apenas uma entrada e uma saída de um vetor de dois bits. Além das portas citadas, temos uma para zerar o contador ao ser ativada (MR) e a última é o pulso de clock (CLK), referência síncrona do sistema.

Utilizando o índice 1 para o bit mais significativo e o índice 0 para o menos significativo, podemos especificar as conexões entre os flip-flops para implementar a contagem de 0 a 2.

Para uma melhor visualização dos estados presentes (EP), próximos estados (PE) e as entradas D1 e D0 necessárias para promover a mudança de um estado para outro na decida do clock, podemos elaborar uma tabela.

Tabela 4.1 – Tabela de Estados do contador 0-2

EP		PE		FF-D	
Q1	Q0	Q1	Q0	D1	D0
0	0	0	1	0	1
0	1	1	0	1	0
1	0	0	0	0	0
1	1	X	X	X	X

A coluna EP indica todos os estados possíveis da máquina de estados. Na coluna PE estão descritos os respectivos estados a que se deseja passar ao comando do clock. Para que ocorram as mudanças de estado desejadas, as entradas D1 e D0 dos flip-flops devem estar como indica a coluna FF-D.

Desta forma, analisando a primeira coluna, temos o momento em que a máquina está no estado 00. Como esta máquina é um contador, o próximo estado é a seqüência numérica 01. Ao pulso do clock, para que Q1 seja mantido em zero, a entrada respectiva do flip-flop deve ser zero. E para que Q0 comute de zero para um, sua entrada respectiva deve ser um.

Assim, se torna necessário determinar a função que fará com que as entradas D se comportem da forma desejada. A partir da tabela 4.1 podemos, por meio dos mapas de karnaugh, determinar a função mais simples para cada entrada D.

Q1\Q0	0	1
0	0	1
1	0	X

D1=Q0

Q1\Q0	0	1
0	1	0
1	0	X

D0=Q1N . Q0N

a.

b.

Figura 4.4 – Mapas de Karnaugh contador 0-2

Como o contador é de 0 a 2, a máquina não passará pelo estado 11 (3), por isso o próximo estado não foi definido já que o contador não passará por este estado.

Na figura 4.4 a. temos a função para D1 e na 4.5 b., a função para D2.

$$D1 = Q0$$

$$D0 = Q1N \times Q0N$$

As entradas LD e MR que são assíncronas e a de dados (I) e a e suas conexões devem ser projetadas juntas para comandarem o bit de saída por meio das entradas assíncronas (set/reset) do flip-flop. Assim, teremos, para cada bit, uma função para set (S) e uma para reset (R) em termos das entradas assíncronas que serão as mesmas para todos os flip-flops, do mais ao menos significativo.

Podemos, neste caso, definir as funções de S e R para um bit e utilizá-las para todos os outros. Para facilitar esta definição temos a tabela verdade.

Tabela 4.2 – Tabela verdade função load e Master Reset

Entradas			Saídas	
I	LD	MR	S	R
X	X	1	0	1
0	1	0	0	1
1	1	0	1	0
X	0	0	0	0

Da mesma forma, podemos obter as funções simplificadas para S e R.

MR \ LD I				
	0	0	1	0
	0	0	0	0

S = MRN . LD . I

a.

MR \ LD I				
	0	0	0	1
	1	1	1	1

R = MR + LD . IN

b.

Figura 4.5 – Mapas de Karnaugh função load e Master Reset

Na figura 4.5 a. temos a função para cada entrada S do flip-flop e na 4.6 b., a função para cada entrada R. Podemos desta forma controlar as saídas dos flip-flops independentemente do clock.

$$S = MRN \times LD \times I$$

$$R = MR + LD \times IN$$

Com todas as funções definidas, já estamos aptos a desenhar o esquemático do circuito em um simulador para validar o projeto.

CONTADOR DE 0 A 5

O módulo do contador de 0 a 5 é seis (m=6). Portanto, dois bits não são suficientes para implementá-lo, já que com n=2 podemos implementar contadores de módulo até 4. Com n=3 pode-se implementar contadores com módulo máximo igual a 8. Então, para este contador serão usados três flip-flops, um para cada bit.

Comparativamente com o contador 0-2, este terá uma entrada a mais, justamente referente ao bit adicional necessário. Conseqüentemente, tem também uma saída a mais pelo mesmo motivo, além da saída *carry* ausente até então.

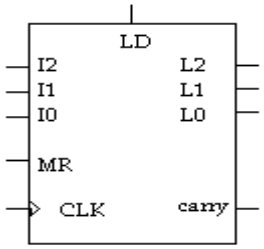


Figura 4.6 – Representação do contador 0 a 5

As entradas e saídas de dados (I e L) podem ser representadas por vetores de 3 bits cada uma. A saída *carry* é utilizada, como já explicado, para excitar o dígito mais significativo na conexão dos contadores em cascata. As demais entradas (MR e LD) são semelhantes às entradas de mesmo nome do contador 0-2, tendo inclusive as mesmas funções, mas, neste caso, para três bits.

Os índices 0, 1 e 2 são referentes ao bit menos ao mais significativo, respectivamente. Isto facilita a especificação das conexões entre os flip-flops que irão implementar esta máquina de estados.

A tabela de estados ilustra a seqüência de estados que o contador deve seguir. Com três bits temos 8 estados possíveis, dos quais a máquina só passará por 6. Os estados 110 e 111 não são utilizados, já que o contador deve reiniciar ao chegar em 5 (101).

Tabela 4.3 – Tabela de Estados do contador 0-5

EP			PE			FF-D		
Q2	Q1	Q0	Q2	Q1	Q0	D2	D1	D0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0	0
1	0	0	1	0	1	1	0	1
1	0	1	0	0	0	0	0	0
1	1	0	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X

Podemos notar que a coluna FF-D é igual à coluna PE. Isto pode se deve ao comportamento do flip-flop D que simplesmente escreve os dados da entrada na saída ao comando do clock.

Para definir, a partir da tabela 4.3, as equações simplificadas que definem a relação entre os sinais de entrada e saída para que a máquina passe por todos os estados desejados, utiliza-se os mapas de karnaugh.

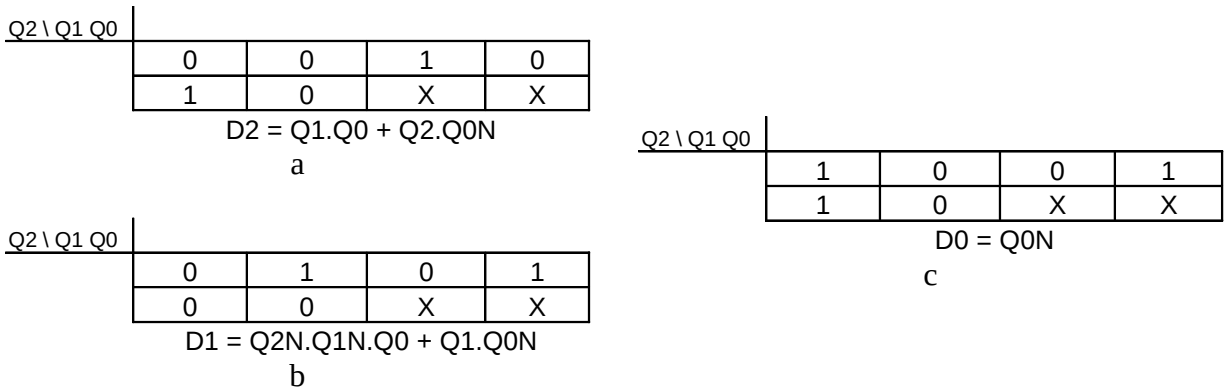


Figura 4.7 - Mapas de Karnaugh contador 0-5

Dos mapas de karnaugh da figura 4.7, temos as equações para as entradas D de cada flip-flop do circuito.

$$D2 = Q1 \times Q0 + Q2 \times Q0N$$
$$D1 = Q2N \times Q1N \times Q0 + Q1 \times Q0N$$
$$DO = Q0N$$

As equações para S e R são as mesmas definidas para o contador 0-2. Então as funções para o contador 0-5 estão todas definidas prontas para serem validadas por simulações.

CONTADOR DE 0 A 9

O contador de 0 a 9 tem módulo dez. Portanto o mínimos de bits necessários para implementá-lo é quatro (2^4 = 16). Assim, este será o maior contador utilizado, já que possui quatro flip-flops, um a mais do que o contador 0-5.

Assim como a diferença entre os contadores 0-2 e 0-5, desconsiderando-se a saída carry, é de uma entrada e uma saída a mais referentes ao bit adicionado, o contador 0-9 terá uma entrada e uma saída a mais do que o contador 0-5. Ambas decorrentes da necessidade de mais um bit.

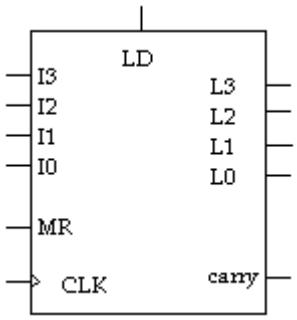


Figura 4.8 - Representação do contador 0 a 9

As entradas e saídas de dados (I e L) podem ser representadas por vetores de quatro bits. As outras portas têm a mesma função de suas respectivas no contador 0-5.

Agora o bit mais significativo é o de índice 3, decrescendo até 0, que é o índice do menos significativo. Desta forma, podemos montar a tabela de estados deste contador.

Tabela 4.4 – Tabela de Estados do contador 0-9

EP				PE				FF-D			
Q3	Q2	Q1	Q0	Q3	Q2	Q1	Q0	D3	D2	D1	D0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	1	0	0	1	1
0	0	1	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	0	1	1	1
0	1	1	1	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1	1	0	0	1
1	0	0	1	0	0	0	0	0	0	0	0
1	0	1	0	X	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X

1	1	1	0	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X

Com 4 bits temos 2^4 estados possíveis, dos quais apenas 10 serão percorridos pelo contador.

Com a Tabela de Estados obtemos as equações simplificadas por meio de mapas de karnaugh.

Q3Q2\Q1Q0

0	0	0	0
0	0	1	0
X	X	X	X
1	0	X	X

D3 = Q2.Q1.Q0 + Q3.Q0N

a

Q3Q2\Q1Q0

0	0	1	0
1	1	0	1
X	X	X	X
0	0	X	X

D2 = Q2N.Q1.Q0 + Q2.Q1N + Q2.Q0N

b

Q3Q2\Q1Q0

0	1	0	1
0	1	0	1
X	X	X	X
0	0	X	X

D1 = Q3N.Q1N.Q0 + Q1.QON

c

Q3Q2\Q1Q0

1	0	0	1
1	0	0	1
X	X	X	X
1	0	X	X

D0 = Q0N

d

Figura 4.9 - Mapas de Karnaugh contador 0-9

Dos mapas de karnaugh da figura 4.9, temos as equações para as entradas D de cada flip-flop do circuito.

$$D3 = Q2.Q1.Q0 + Q3.Q0N$$

$$D2 = Q2N.Q1.Q0 + Q2.Q1N + Q2.Q0N$$

$$D1 = Q3N.Q1N.Q0 + Q1.QON$$

$$D0 = Q0N$$

Como as equações para S e R são as mesmas definidas para o contador 0-2, então as funções para o contador 0-9 estão todas definidas prontas para serem validadas por simulações.

SIMULAÇÃO DOS CONTADORES

Concluída a especificação funcional do circuito, o próximo passo seria definir a tecnologia a ser utilizada. Como o projeto fruto deste trabalho é uma pequena parte a ser integrada ao projeto maior, o Milênio, a tecnologia a ser utilizada será a utilizada pelo próprio projeto Milênio.

Simulação no CircuitMaker

No CircuitMaker é possível desenhar graficamente as funções lógicas definidas na especificação funcional dos contadores. Assim, podemos facilmente definir a topologia do circuito utilizando dispositivos digitais básicos.

Contador 0-2

Para o circuito do contador 0-2 foi definido o seguinte esquema.

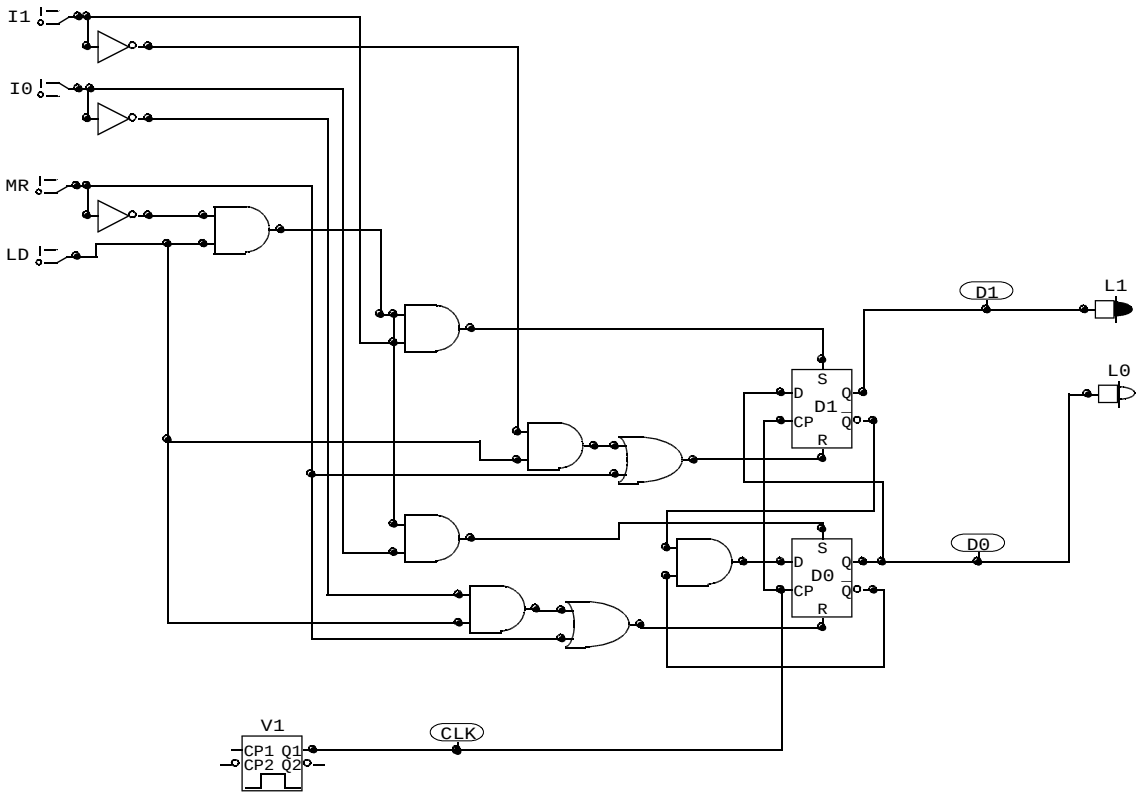


Figura 4.10 – Esquemático do contador 0-2

As portas lógicas estão implementando as funções definidas no item 4.2.

Com o circuito montado no simulador, pode-se fazer uma análise preliminar a partir da observação dos sinalizadores lógicos identificados na figura 4.10 como L1 e L0. O estado atual e suas sucessivas alterações podem ser observadas com este elemento. Desta forma, ao manter as entradas assíncronas desativadas e o clock comutando, pode-se visualizar a sucessão de estados correspondente a contagem de 0 a 2 em ciclos.

Porém esta observação é muito limitada. Um registro da variação de cada porta ao longo do tempo pode propiciar uma análise mais completa do funcionamento do sistema.

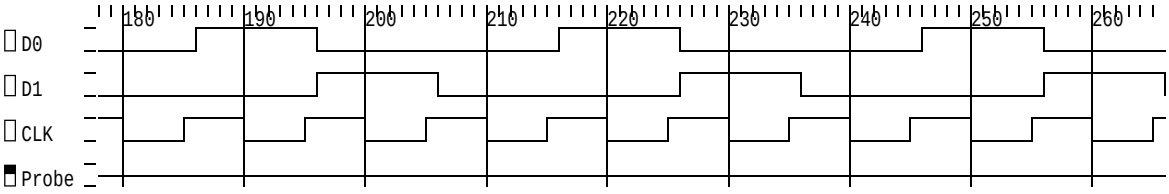


Figura 4.11 – Diagrama temporal do contador 0-2

O diagrama temporal da figura 4.11 ilustra o funcionamento da parte síncrona do contador. Para tanto, as entradas MR e LD devem ser mantidas inativas. Desta forma teremos as saídas Q1 e Q0 variando com o pulso do clock. Na primeira subida do clock, Q1 se matem em zero e Q0 comuta para 1. Nos flancos de subida subseqüentes, as saídas seguem a seqüência determinada na tabela 4.1 indeterminadamente em ciclos.

O atraso que pode ser observado entre a subida do clock e a comutação da saída é o correspondente ao atraso dos dispositivos da biblioteca do CircuitMaker. Este atraso não é o mesmo atraso esperado na implementação deste projeto, pois cada dispositivo tem uma característica de atraso. O atraso esperado do dispositivo será discutido posteriormente.

Incluindo as portas assíncronas na simulação teremos as seguintes formas de ondas.

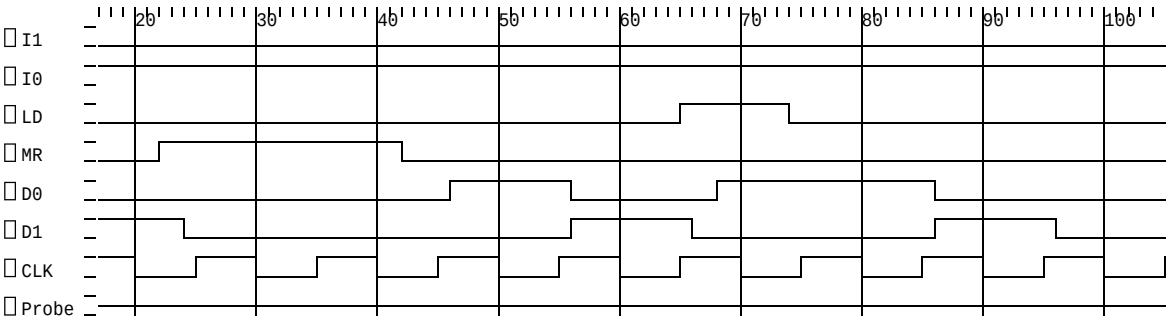


Figura 4.12 – Diagrama Temporal do contador 0-2 com load e Master Reset

É importante notar que as entradas assíncronas prevalecem sobre a contagem síncrona. Enquanto MR ou LD estão ativos, o pulso do clock não leva a máquina para o próximo estado. Ao ativar MR as saídas são zeradas. Ao ativar LD, os dados de I1 e I0 passam para a saída.

Se os dados I1 e I0 introduzirem um estado não considerado na tabela de estados (11, por exemplo), o próximo estado será indefinido até que a máquina seja zerada ou até que ocorra um estado definido.

Contador 0-5

O esquema do circuito do contador 0-5 é um pouco mais complexo do que o do contador 0-2. Além de ter um flip-flop a mais, as funções de D2 e D1 são mais complexas.

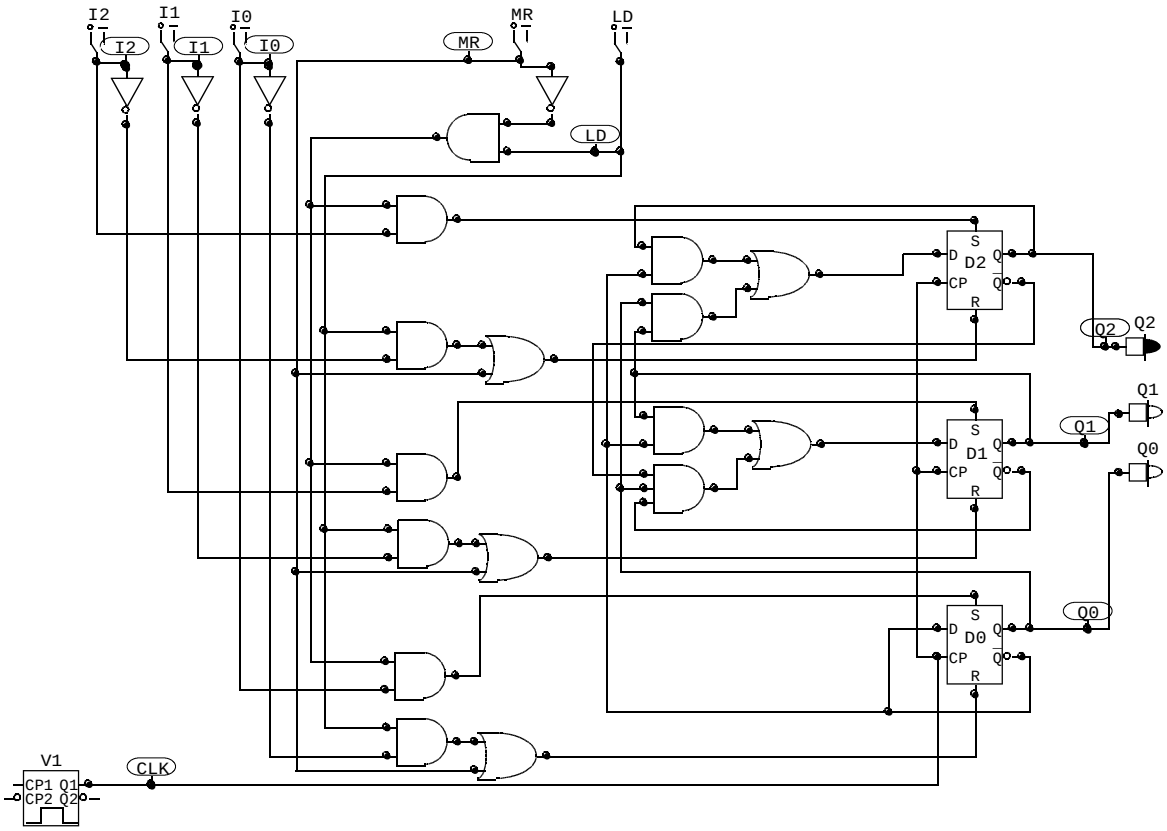


Figura 4.13 - Esquemático do contador 0-5

Com as entradas assíncronas desabilitadas obtemos o seguinte digrama temporal.

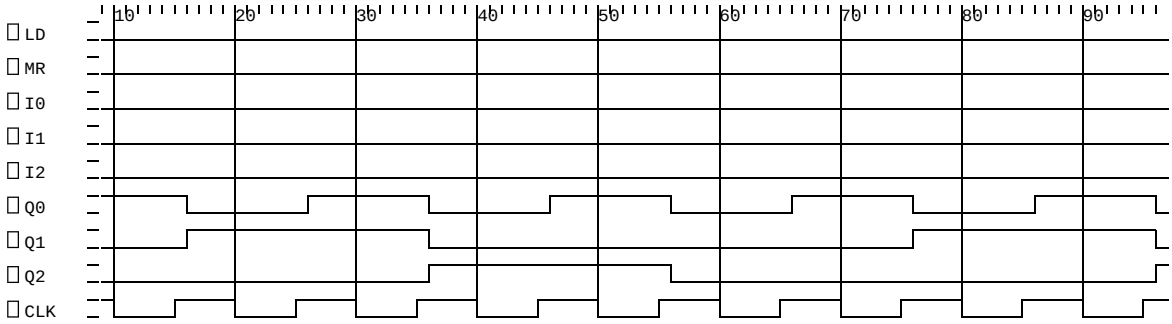


Figura 4.14 – Diagrama Temporal do contador 0-5

Comparando o diagrama temporal da figura 4.14 com a tabela de estados desta máquina (tabela 4.3), podemos observar a correta seqüência de estados que resulta em um contador de 0 a 5 sob dois prismas.

A figura 4.15 ilustra o funcionamento da parte assíncrona do contador. Podemos observar que com a entrada de controle LD desativada, a comutação das entradas de dados não influencia na saída do contador.

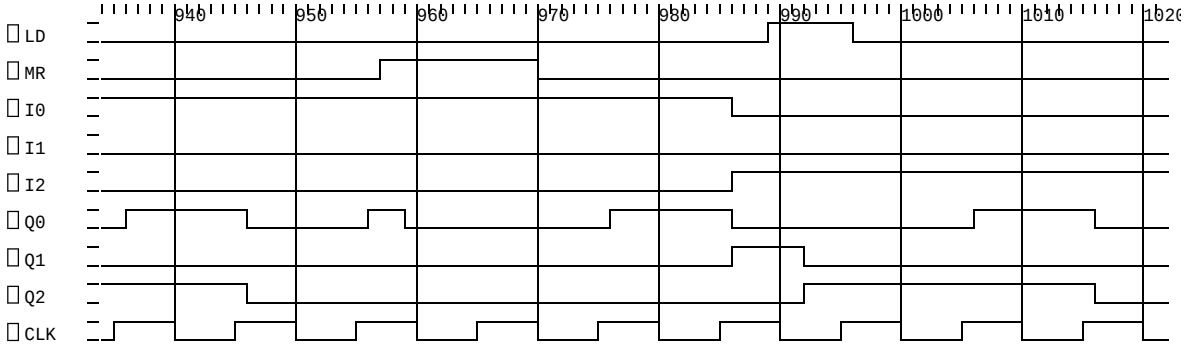


Figura 4.15 - Diagrama Temporal do contador 0-5 com load e Master Reset

Enquanto a porta LD estiver ativada, a saída de dados (L) seguirá às entradas (I) independentemente do pulso do clock. Se as entradas forem mantidas estáveis, a saída será estável. Se comutarmos I2, I1 ou I0, as saídas correspondentes seguirão as entradas de forma assíncrona.

Contador 0-9

Podemos observar no esquemático do circuito do contador 0-9 (figura 4.16), a repetição das estruturas que controlam as entradas S e R dos flip-flops. Estas estruturas são a implementação das equações de S e R que são idênticas para cada bit.

O contador 0-9 utiliza dez portas lógicas para conectar os flip-flops em cascata. Dos três contadores projetados, este é o mais complexo e o que ocupará uma maior área em sua implementação física.

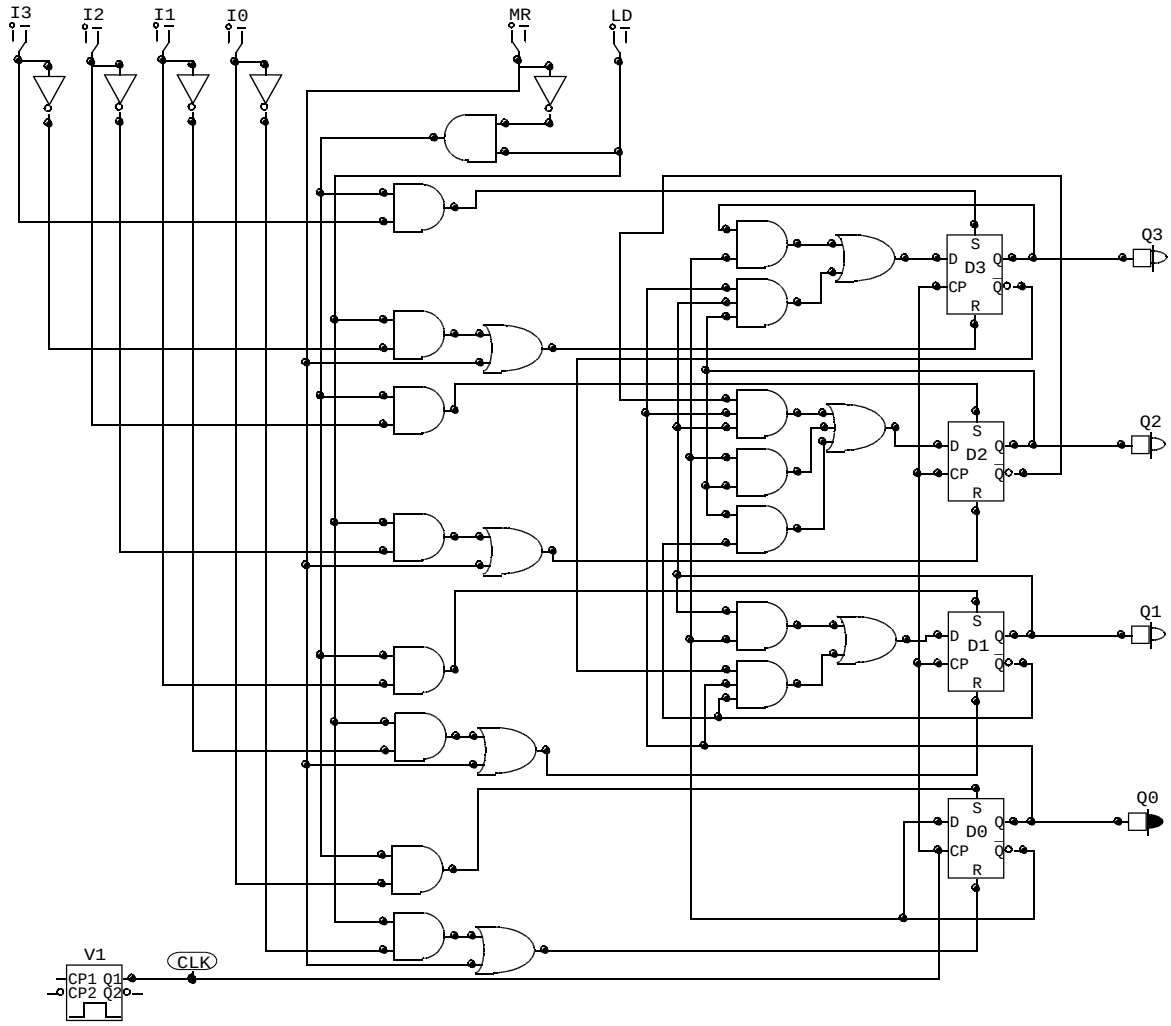


Figura 4.16 - Esquemático do contador 0-9

Podemos observar no diagrama temporal, de baixo para cima, as formas de onda do clock, de Q0, Q1, Q2 e Q3. Com exceção do bit mais significativo, a razão entre a freqüência de comutação de um bit e o bit imediatamente mais significativo que ele, é dois. Esta é uma característica da seqüência binária de códigos. O bit Q3 é uma exceção neste caso porque seu ciclo individual não está completo. Estaria se fosse um contador de 0 a 15.

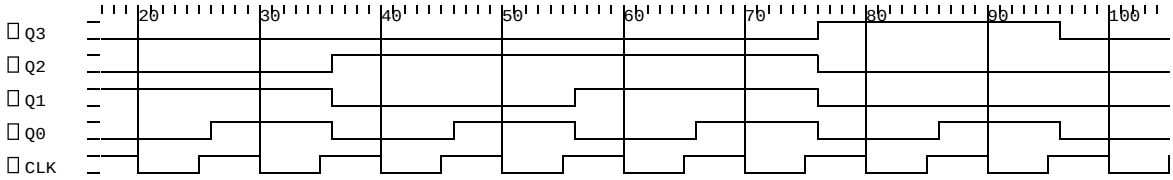


Figura 4.17 – Diagrama Temporal do contador 0-9

Projeto em VHDL

Buscando uma descrição mais consistente e que possibilite uma melhor aproximação do projeto físico, foi desenvolvida a descrição VHDL do sistema. Esta descrição é uma potente ferramenta para se trabalhar de forma simples e objetiva com os diferentes níveis de abstração do projeto. Possibilita ainda, com pequenas adaptações, a simulação e posterior desenvolvimento do leiaute do circuito no pacote CADENCE. CAD utilizado para o desenvolvimento de leiaute no Projeto Milênio.

O nível hierárquico mais baixo a ser considerado neste projeto é o de portas lógicas, que não necessitam serem descritas, por serem elementos básicos.

O flip-flop utilizado não foi projetado neste trabalho. Ele já havia sido objeto de estudo em outros trabalhos dentro do Projeto Milênio e foi aproveitado como um elemento básico. Desta forma o que interessa neste projeto é o funcionamento externo deste dispositivo abstraindo a sua implementação. Assim, ele está descrito neste trabalho, apenas de forma comportamental. Esta descrição está disponível no apêndice A deste trabalho, assim como todos os códigos desenvolvidos na descrição do sistema.

Função Load

Como já observado no projeto lógico e na simulação no CircuitMaker, a estrutura que implementa as funções assíncronas dos contadores se repetem. Visando uma simplificação do projeto, esta estrutura repetitiva será definida como Função Load.

Esta função implementa tanto a carga assíncrona de dados na saída, quanto a Master Reset, que zera o contador ao ser ativada.

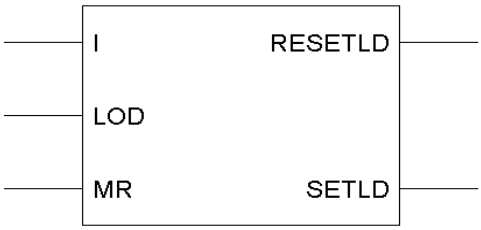


Figura 4.18 – Função Load

A entrada I corresponde a cada bit de entrada de dados. As saídas RESETLD E SETLD comandam as entradas S e R de cada flip-flop. Desta forma, as entradas LOD e MR têm a função de carregar os dados da entrada na saída e de zerar o bit, respectivamente.

Resumidamente, este bloco realiza as funções S e R, definidas anteriormente.

$$SETLD = MRN \times LOD \times I$$
$$RESETLD = MR + LOD \times IN$$

Onde SETLD corresponde ao Set e RESETLD corresponde a Reset do flip-flop.

Contador 0-2

O contador 0-2 , que foi descrito na seção 4.2, é definido pelas seguintes funções em seus flip-flops.

$$D1 = Q0$$
$$D0 = Q1N \times Q0N$$
$$S = MRN \times LD \times I$$
$$R = MR + LD \times IN$$

Porém, utilizando a técnica de abstração, não é necessário considerar todas as equações definidas. Afinal, as equações para Set e Reset já foram implementados no bloco Função Load. Desta forma, para implementar as entradas assíncronas de cada flip-flop, basta conectar este bloco sem se preocupar comas funções.

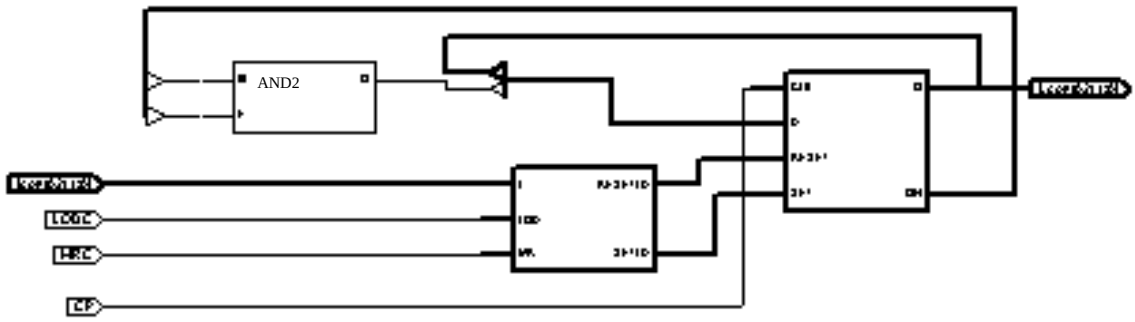


Figura 4.19 – Estrutura do contador 0-2

As entradas e saídas de dados foram definidas como vetores de dois bits. O bloco que representa o flip-flop,está representando os dois flip-flops do contador assim como o bloco da Função Load corresponde a dois. Desta forma, o esquemático do sistema fica sensivelmente simplificado.

A porta AND2 implementa a função D0 enquanto que a função D1 é implementada apenas pela conexão de uma das saídas Q à uma entrada D de um flip-flop. As duas saídas QN (Q1N e Q0N) estão conectadas na porta lógica AND2 e a saída desta porta na entrada D do flip-flop correspondente ao bit menos significativo.

Desta forma o funcionamento do dispositivo descrito pode ser analisado por meio das formas de onda de suas entradas e saídas.

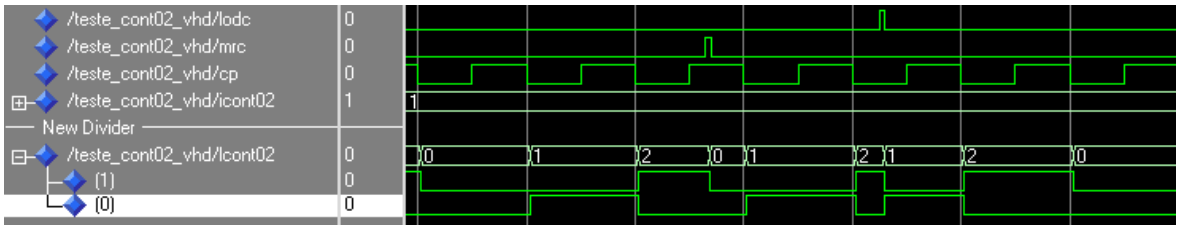


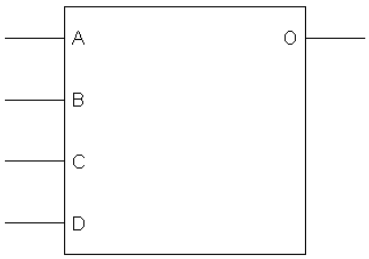
Figura 4.20 – Formas de onda do contador 0-5 (VHDL)

Os vetores de entrada e saída podem ser observados diretamente na forma decimal. Os dois bits do vetor de saída estão representados separadamente abaixo do vetor.

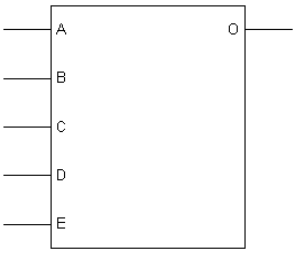
Funções D do contador 0-5

As funções D dos flip-flops do contador 0-5 são mais complexas do que as do contador 0-2. Assim, para simplificar a implementação do contador, estas funções podem primeiramente ser implementadas e depois utilizadas como blocos.

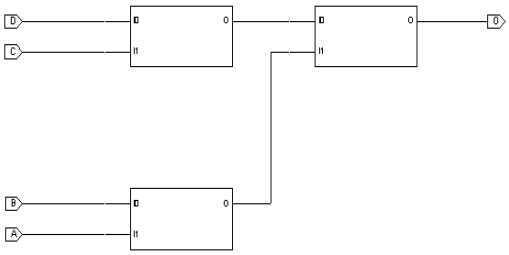
As funções D2 = Q1 x Q0 + Q2 x Q0N e D1 = Q2N x Q1N x Q0 + Q1 x Q0N do contador 0-5, ao serem implementadas podem ser vistas como os seguintes blocos.



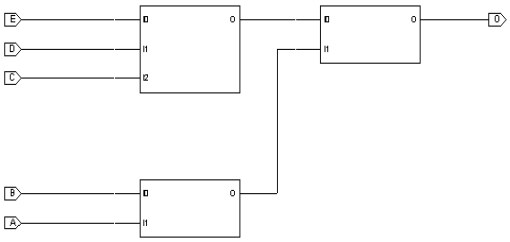
a) Função D2 contador 0-5



c) Função D1 contador 0-5



b) Estrutura Função D2 0-5



d) Estrutura Função D1 0-5

Figura 4.21 – Funções D do contador 0-5

Nas estruturas de cada função, as caixas à esquerda são portas AND e as caixas à direita, que estão diretamente ligadas às saídas respectivas, são portas OR. Desta forma, estes blocos apresentados na figura 4.21 (a. e c.) podem implementar as funções desejadas.

Contador 0-5

Utilizando as funções D e a Função Load implementadas, a implementação do contador se restringe simplesmente à correta conexão destes blocos.

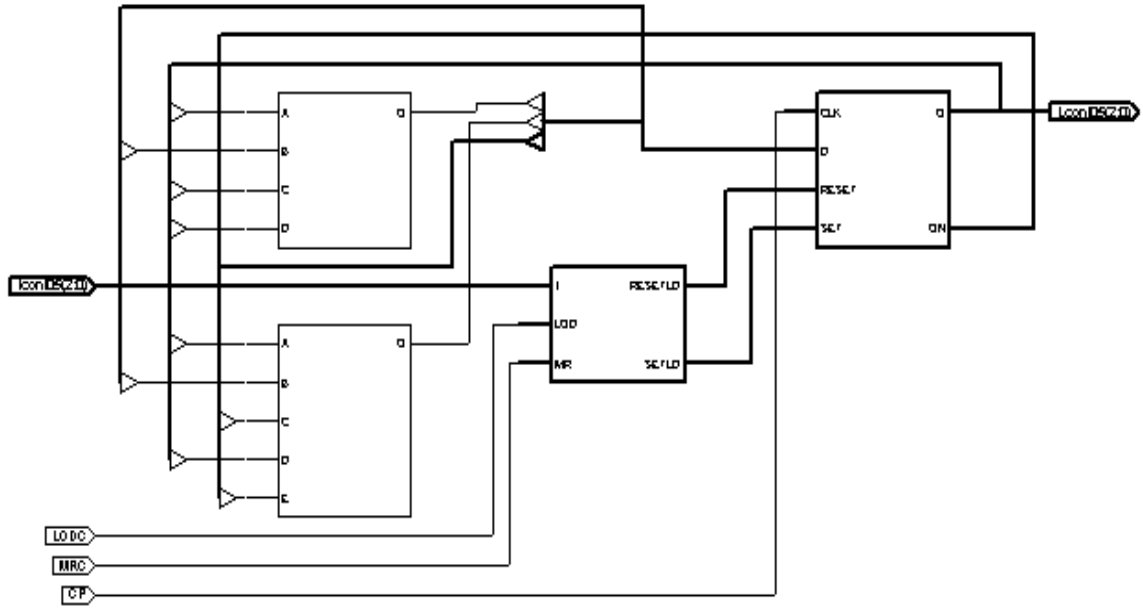


Figura 4.22 – Estrutura do contador 0-5 (VHDL)

Assim como no contador 0-2 os três flip-flops e os três blocos de função Load estão representados por apenas um. Podemos observar que as entradas D recebem as funções D e, no caso de D0, recebe Q0N.

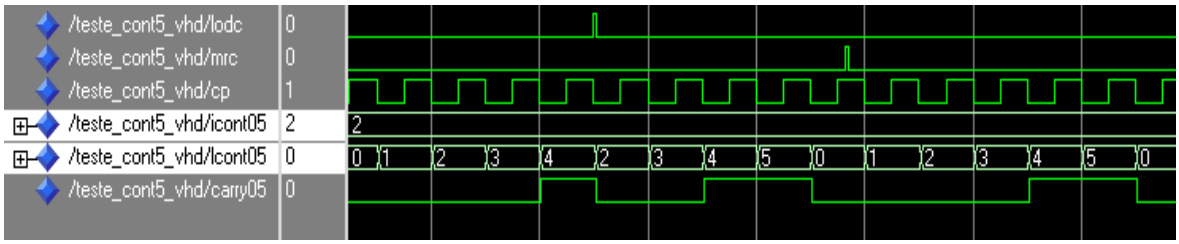


Figura 4.23 – Formas de onda contador 0-5 (VHDL)

A saída *carry* pode ser observada nesta figura. Como os flip-flops são ativos no flanco de decida, sempre que o contador reiniciar seu ciclo, esta saída pode excitar o clock do contador mais significativo.

Funções D do contador 0-9

Da seção 4.4 deste capítulo, temos as seguintes funções D para os flip-flops do contador 0-9.

$$D3 = Q2.Q1.Q0 + Q3.Q0N$$

$$D2 = Q2N.Q1.Q0 + Q2.Q1N + Q2.Q0N$$

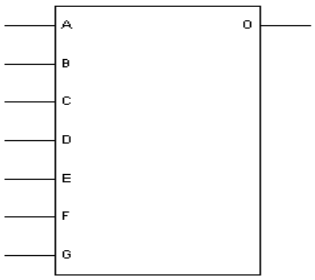
$$D1 = Q3N.Q1N.Q0 + Q1.QON$$

$$D0 = Q0N$$

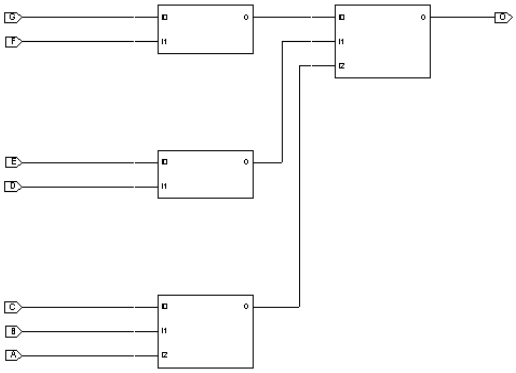
Observando as funções D3 e D1 do contador 0-9 podemos notar que ambas utilizam a mesma configuração de portas lógicas para serem implementadas. Notamos ainda que esta é também a configuração de portas da função D1 do contador 0-5.

Desta forma, podemos utilizar blocos com a mesma estrutura para implementar qualquer uma destas funções, bastando para tanto, efetuar corretamente as conexões indicadas por cada equação.

Como a função D0 é apenas uma conexão e portanto não precisa de implementação a parte, falta apenas implementar a equação D2 = Q2N.Q1.Q0 + Q2.Q1N + Q2.Q0N para partimos para a montagem do contador.



a) Função D2 contador 0-9



b) Estrutura Função D2 0-9

Figura 4.24 – Função D2 do contador 0-9

A caixa que está diretamente conectada à saída é uma porta lógica OR e as outras são portas AND.

Contador 0-9

Com todas as funções implementadas, basta conectá-las corretamente para implementar o contador 0-9.

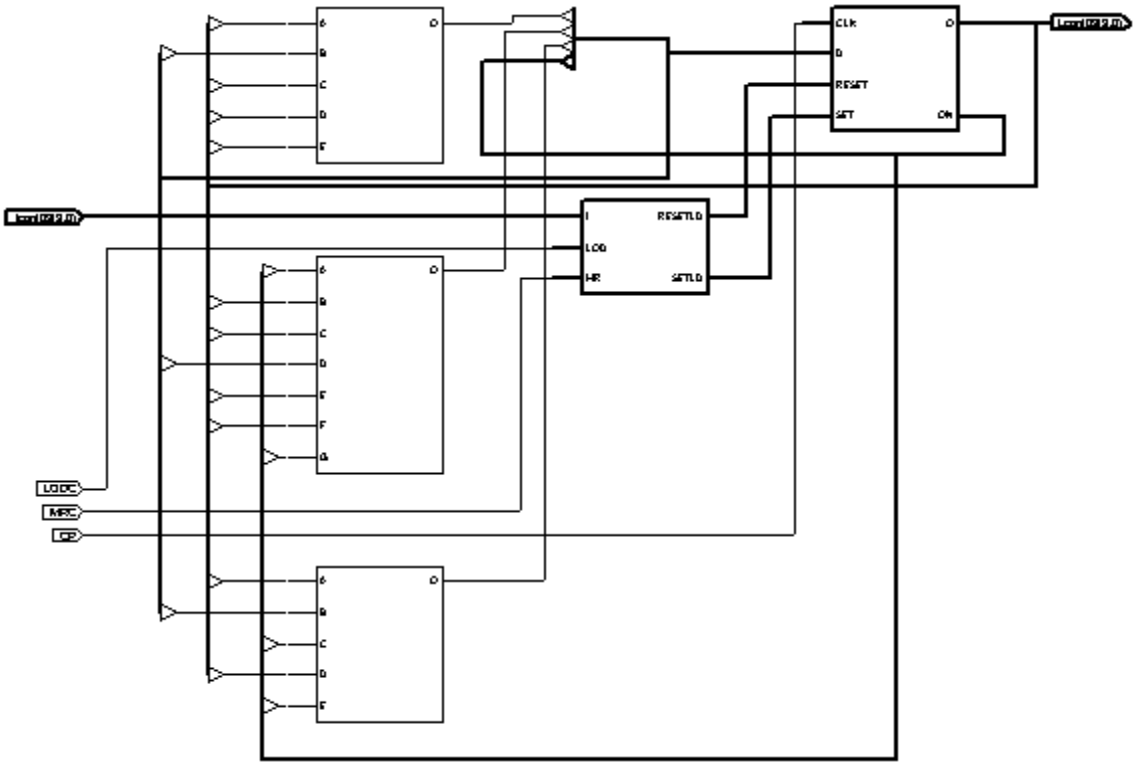


Figura 4.25 – Estrutura do contador 0-9 (VHDL)

Podemos ver, no barramento de entrada D dos flip-flops, as quatro funções necessárias para excitar cada entrada D.

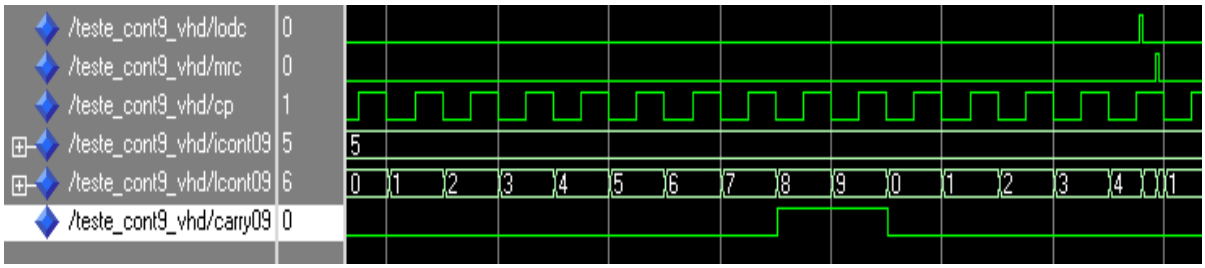


Figura 4.26 – Formas de onda do contador 0-9 (VHDL)

Com todos os contadores projetados, o próximo passo é conectá-los de forma a implementar o relógio.

RELÓGIO

Para implementar o relógio a partir dos contadores, basta conectá-los em cascata de forma que o contador correspondente ao dígito menos significativo, incremente o dígito imediatamente acima ao reiniciar seu ciclo. Este incremento é realizado conectando a saída carry do menos significativo, no clock do mais significativo.

Por exemplo, quando o dígito de unidade de segundo chegar a nove, ao voltar para zero ele incrementa a dezena de segundo. Após cinco ciclos da unidade de segundo, a dezena de segundo estará em cinco, e ao ser incrementada deverá voltar a zero e, por meio de sua saída carry, incrementar a unidade de minuto. E assim sucessivamente.

Como já definido anteriormente, os dígitos Seg0, Min0 e Hor0, serão compostos por contadores 0-9, os dígitos Seg1 e Min1 serão compostos por contadores 0-5 e o dígito Hor1 por um contador 0-2.

Porém, ao atingir 23:59:59, o dígito Hor0, que é um contador 0-9, deve voltar a zero, e não continuar sua contagem até 9, como seria sua seqüência natural. Para que o contador 0-9 volte a zero em vez de continuar sua contagem, será necessário um circuito síncrono adicional. Esta função será definida como função 24 horas.

FUNÇÃO 24 HORAS

Como este circuito deve funcionar ao comando de um clock, ele deve utilizar no mínimo um flip-flop. Para manter a regularidade do projeto, o flip-flop a ser usado neste circuito adicional será o mesmo tipo D.

Este circuito é uma máquina de estados que deverá zerar o contador Hor0 quando o contador Hor1 estiver no estado 2, o contador Hor0 no estado 3 e o contador Min1 reiniciar seu ciclo. A partir daí podemos definir as entradas e saídas necessárias para este circuito.

Para identificar se Hor1 está em 2, precisamos apenas do seu bit mais significativo. Para definir se Hor0 esta em 3, basta observar os dois bits menos significativos. Para monitorar estes parâmetros, necessitamos de três entradas.

Como este circuito irá excitar o reset do Hor1, ele deverá ter em sua entrada o master reset. A última entrada necessária é a referência síncrona, o clock.

As saídas serão duas. Uma para excitar o Hor1 e outra para excitar o master reset do Hor0.

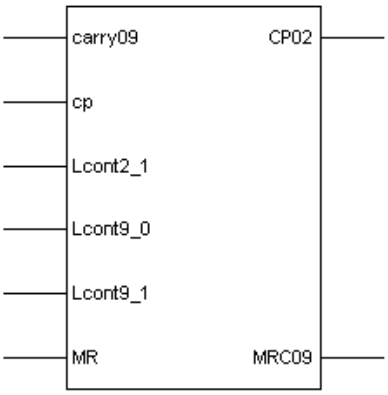


Figura 5.1 – Bloco Função Load

O circuito pode ser analisado de forma mais simples se for observada uma saída separadamente da outra.

A saída CP02 será a mesma da entrada carry09 enquanto Hor1 for diferente de 2. Então esta saída será a entrada carry09 ou a saída da porta AND3 que tem em sua entrada os três parâmetros que informam que Hor está em 23.

A saída MRC09 será ativada pela entrada MR ou quando Lcont2_1 igual a 1 e o carry de min1 passe de 1 para 0.

Tabela 5.1 – Tabela de estados parcial função 24h

Entradas			Saídas	
CP (carry Min1)	Lcont2_1	AND3	Q	OUT
1-0	0	0	0	0
1-0	0	1	1	0
1-0	1	0	0	0
1-0	1	1	1	1

Ou seja, MR09 será ativada quando MR ou OUT(Tab.5.1) for igual a 1.

Desta forma podemos obter o esquemático do circuito.

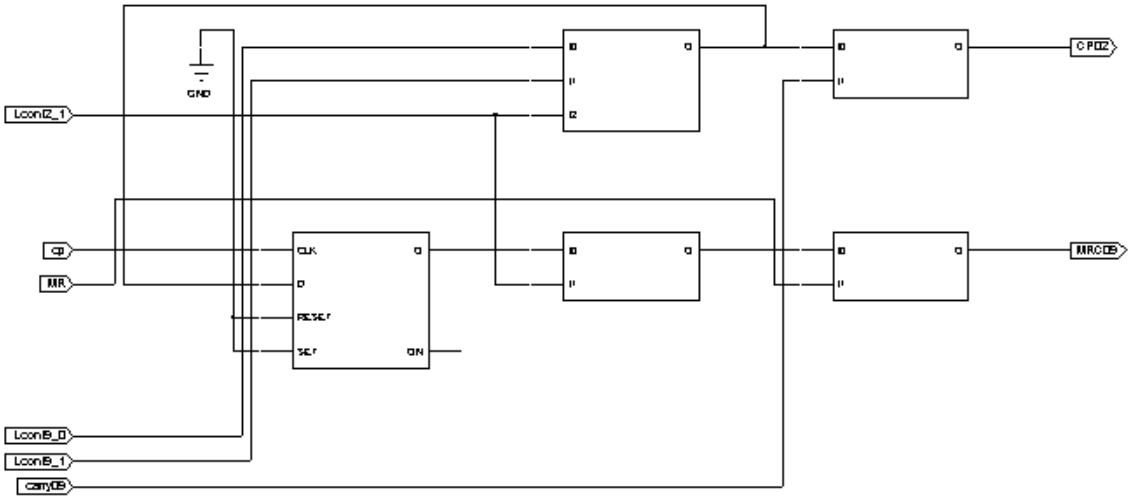


Figura 5.2 – Esquemático Função 24 horas

Novamente as caixas ligadas diretamente às saídas são portas lógicas OR e as da esquerda são portas AND.

A porta AND3 só tem sua saída igual a 1 se Hor estiver em 23. Assim que o Hor1 é zerado, as saídas das duas portas AND vão para zero. Isto faz com que a saída MRC09 pare de ativar o master reset do Hor0 e ele volte a contar normalmente de zero a nove.

SIMULAÇÃO COM CIRCUIT MAKER

Depois de feita a lógica para os contadores, implementados e simulados no Circuit Maker, chegou o momento de montar o relógio e testá-lo para certificar-se que a lógica é válida. Os contadores foram inseridos em macros para melhorar a integração e visualização dos resultados.

Foram utilizados decodificadores 74LS48 para que houvesse uma visualização gráfica, por meio de *display* digital, do comportamento do relógio. Foi acrescentado ainda portas lógicas que auxiliaram no funcionamento da função “Hora”.

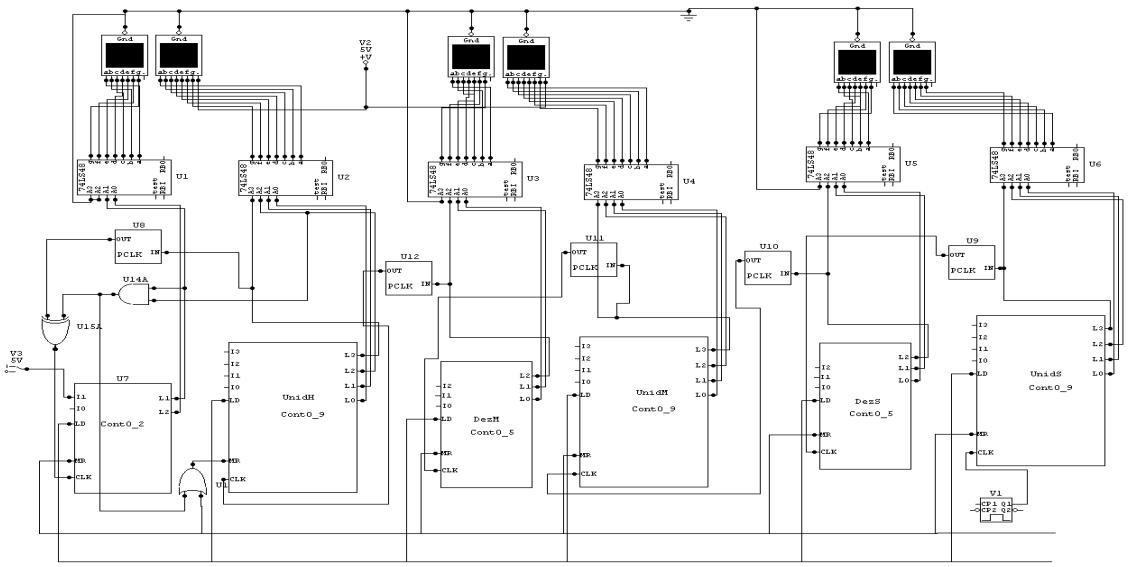


Figura 5.3 – Relógio Digital simulado no Circuit Maker

Nesta simulação foi possível acompanhar o funcionamento completo do relógio. O *clock* da simulação foi acelerado para que se pudesse observar em tempo hábil todas as passagens de tempo registradas pelos *displays* do simulador.

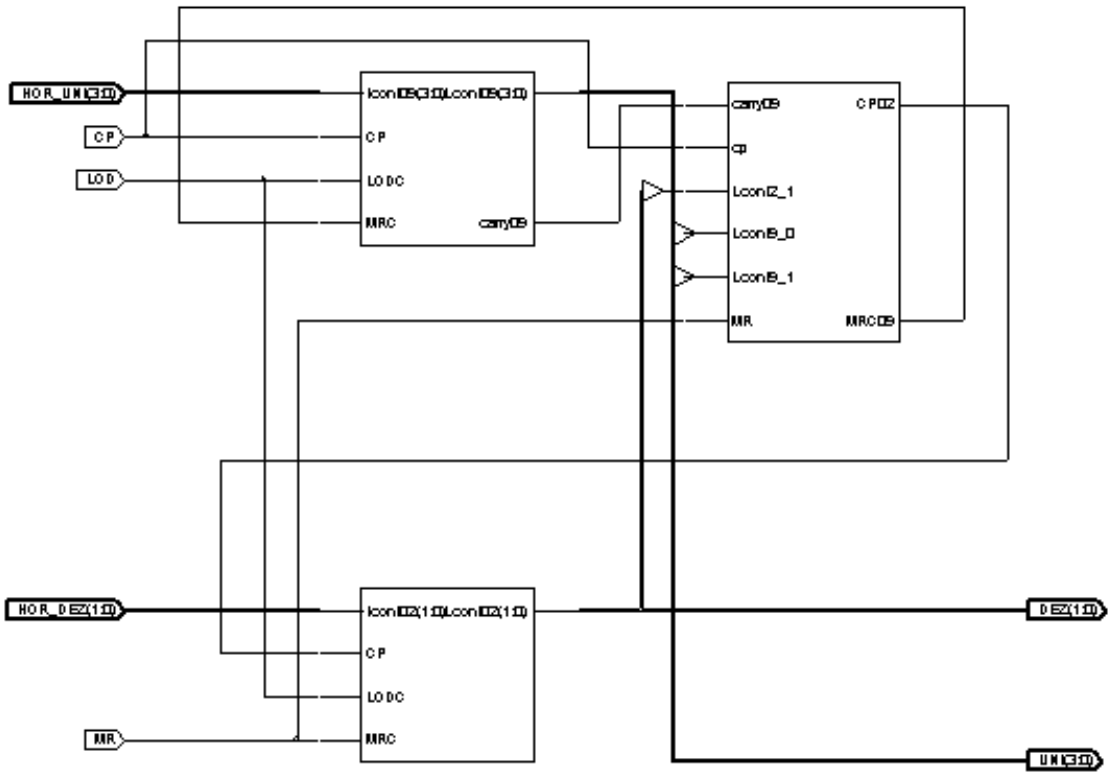
Ao iniciar a simulação, a chave MR foi acionada e quando ela foi desligada o relógio começou sua contagem do zero. A chave LD também foi testada com sucesso, com o relógio funcionando perfeitamente enquanto se ajustavam os valores.

SIMULAÇÃO VHDL

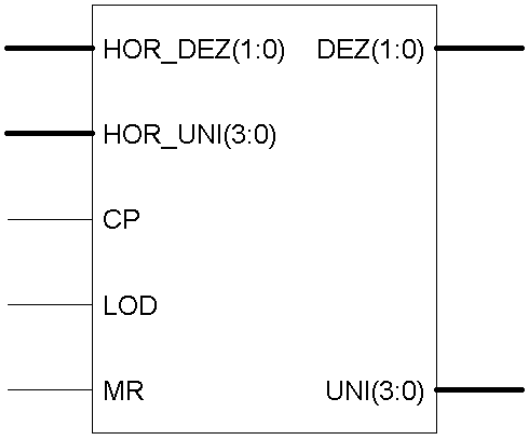
Já seria possível implementar o relógio a partir dos contadores projetados, porém podemos inserir mais um nível hierárquico a fim de facilitar ainda mias a conexão final do relógio.

Agrupando as unidades e dezenas de hora, minuto e segundo, teremos um módulo para cada um. Um módulo de dois dígitos para hora, um de dois dígito para minuto e um de dois dígitos para segundo.

O módulo de hora seria da seguinte forma.



a) Esquemático



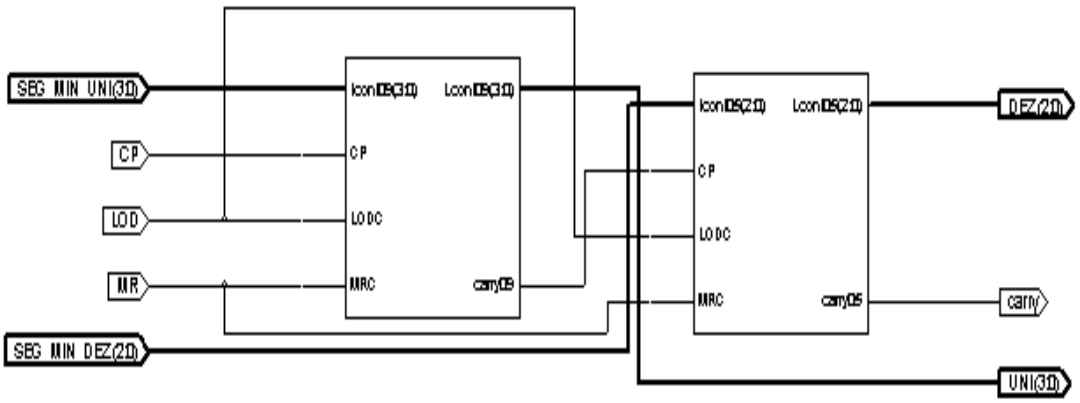
b) Módulo

Figura 5.4 – Módulo de hora

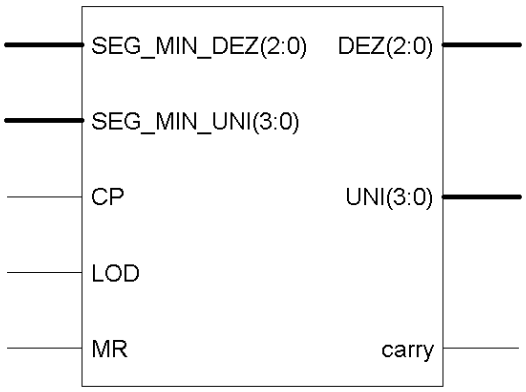
No esquemático temos os contadores 0-9 e 0-2 interligados com o módulo função 24 horas. Como já visto, o contador Hor1 tem como referência síncrona o carry do Hor0 exceto quando Hor está em 23. Neste caso, o clock de Hor1 será excitado quando Hor0 for zerado.

Visualizando o módulo externamente, temos apenas uma entrada de clock, que deverá ser excitada pelo carry do módulo de minuto.

O módulo de minuto seria semelhante, porém mais simples. Ele não possui a função 24 horas. Basta a ligação em cascata explicada anteriormente.



a) Esquemático



b) Módulo

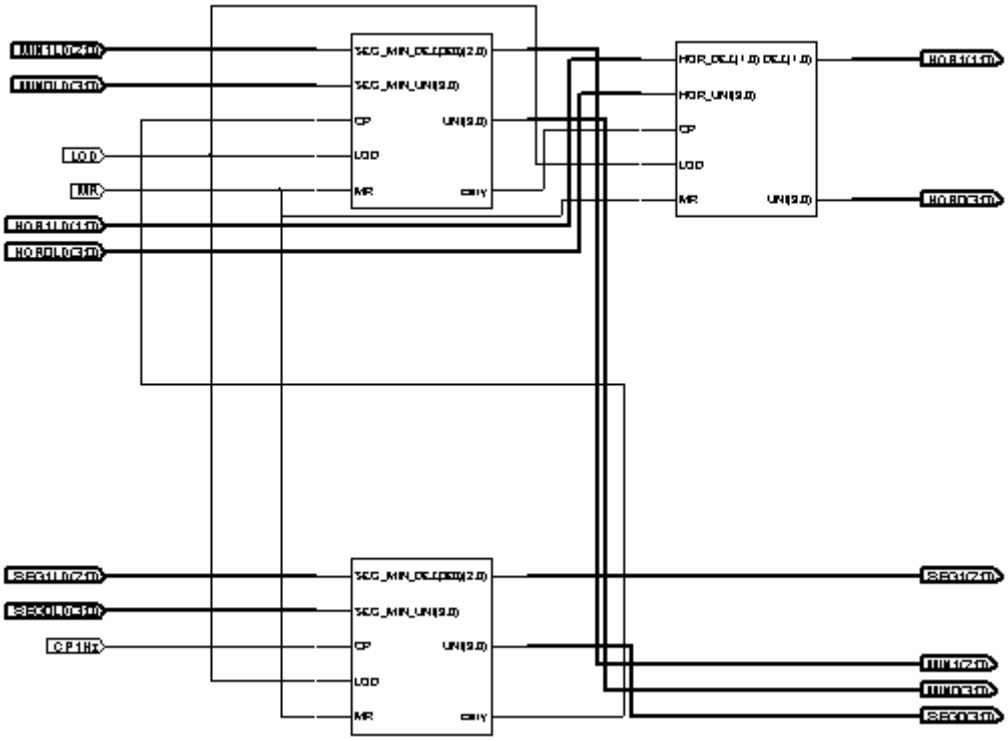
Figura 5.5 – Módulo de minuto e segundo

Este módulo, por ter um mais significativo a ser excitado, tem a saída carry.

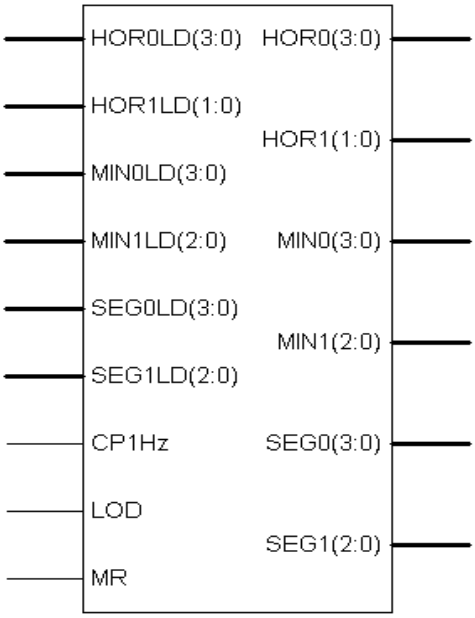
As entradas de master reset e de carga são simplesmente ligadas no mesmo ponto e o carry da unidade excita o carry da dezena.

Esta também é a configuração do módulo de segundo, já que os dois usam os mesmos tipos de contadores conectados da mesma forma. A única diferença entre os dois será a excitação. O de segundo receberá o clock primário de 1Hz e o módulo de minuto receberá o carry do segundo.

Desta forma, conectando os três módulos, teremos finalmente o relógio completo. Este será composto por dois módulos de segundo/minuto e um módulo de hora.



a) Esquemático



b) Módulo do relógio

Figura 5.6 – Ligação dos módulos em cascata

Devido às técnicas de projeto utilizadas, a etapa de implementação do relógio se torna extremamente simples. Basta conectar as entradas de master reset e de ajuste assíncrono em um mesmo ponto para a entrada MR do relógio e em um outro ponto para

entrada LD. A conexão em cascata já abordada anteriormente, é implementada conectando-se as saídas carry dos módulos menos significativos nas respectivas entradas de clock dos módulos mais significativos.

Ao ligar um pulso digital de 1hz na entrada de clock do módulo de segundo, os contadores irão contar as horas, obtendo-se assim o relógio de tempo real.

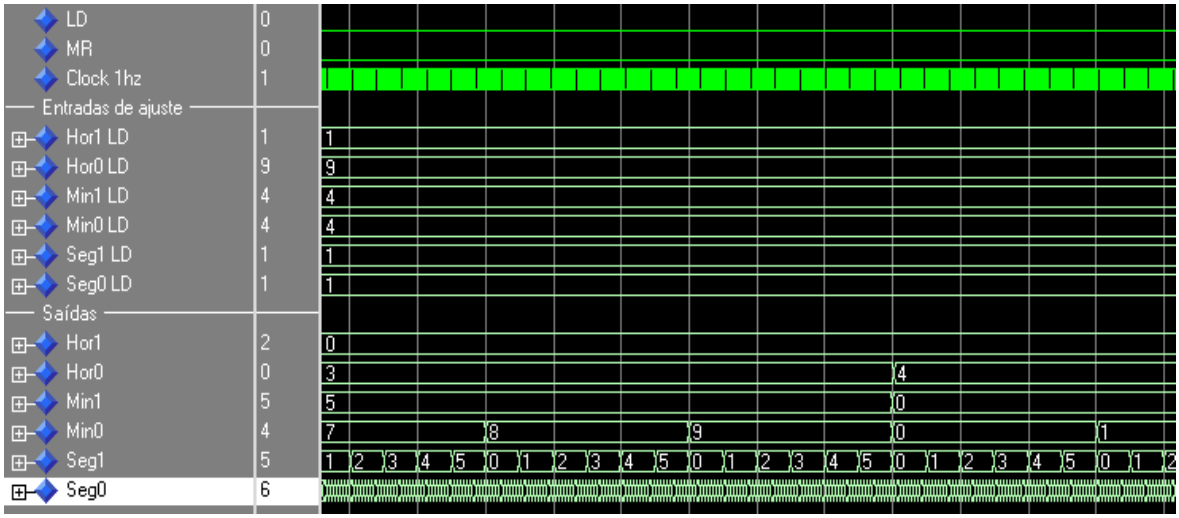


Figura 5.7 – Relógio. Início da contagem

A fim de visualizar um maior período, a unidade de segundo não está visível. Podemos observar na figura acima o relógio do estado 03:57:10 até o estado 04:01:20. Durante todo este período as entradas MR e LD são mantidas inativadas.

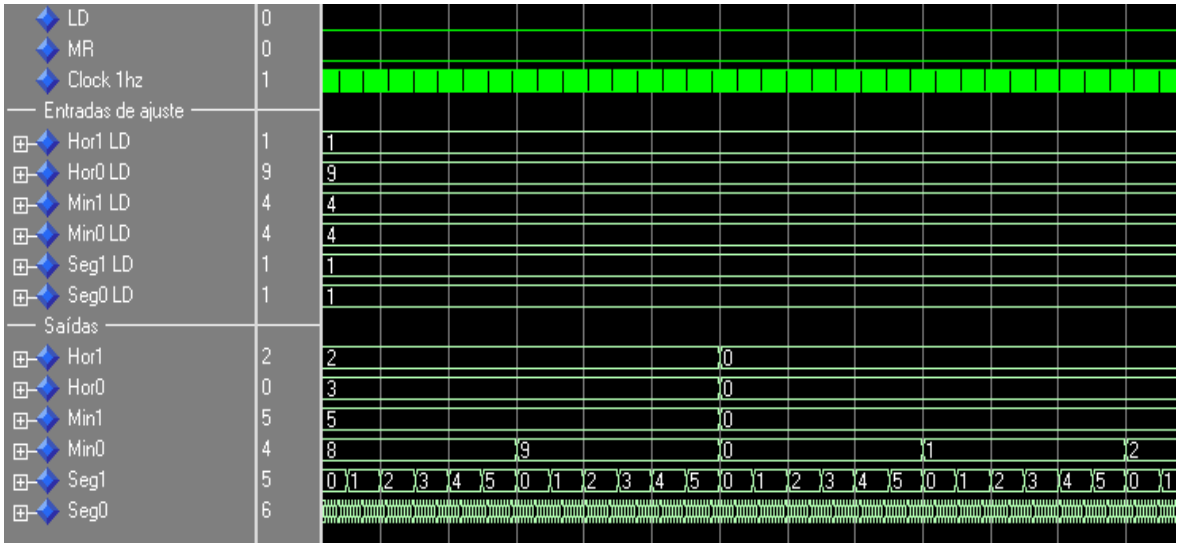


Figura 5.8 - Relógio. Reiniciando o ciclo

Na figura 5.6 podemos visualizar o final de um ciclo e o início do próximo. Ao chegar em 23:59:59 o relógio reinicia do zero.

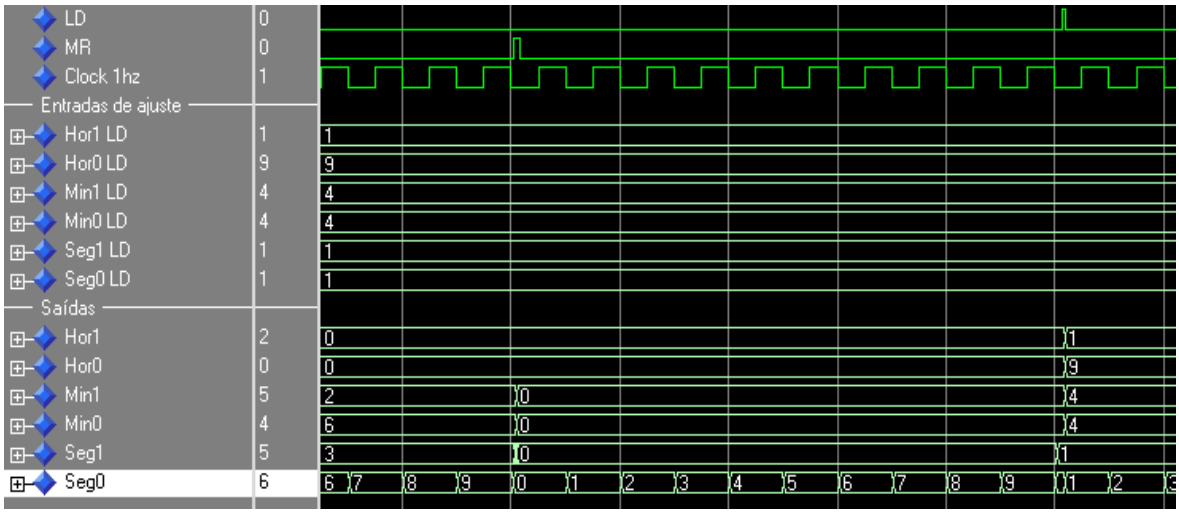


Figura 5.9 – Relógio. Zerando e ajustando

Nesta figura, com um zoom maior, podemos observar a unidade de segundo e os atrasos na atualização de cada dígito.

Ao acionar MR, todos os contadores são levados a zero após o atraso respectivo. Também após o atraso, o horário da saída é ajustado de acordo com as entradas de dados ao pulso de LD. O horário ajustado foi de 19:44:11.

As entradas MR e LD nunca devem ser acionadas simultaneamente, caso isto ocorra o relógio teoricamente irá zerar, pois o MR foi projetado como predominante sobre o LD. Porém, esta predominância foi definida de forma comportamental no flip-flop. Assim, este pode ser um problema a ser superado na implementação física do flip-flop.

A ligação em cascata pode trazer um problema de atraso para o circuito. Afinal, ao conectar dispositivos em cascata, os atrasos são somados. Neste projeto, o atraso medido na simulação do flip-flop no CADENCE foi inferior a 10 ns. Como o atraso pode ser um grave problema para este circuito, foi considerado na simulação um atraso de 10 ns para uma porta lógica e de 30 ns para o flip-flop. Mesmo majorados, podemos ver nas figuras 5.5 e 5.6 que o funcionamento do relógio não foi comprometido.

CONCLUSÕES

Neste trabalho foi desenvolvido um relógio de tempo real com contagem de 24 horas completas. Este relógio tem como objetivo integrar um circuito em SoC. Este último se destina à agricultura de precisão e tem como objetivo propiciar uma melhor gestão do uso da água. Foram utilizadas técnicas de projeto digital e testados com as ferramentas do Circuit Maker e VHDL.

Foram propostas também rotinas para o software, de forma a orientar outros membros do Projeto Milênio em seus trabalhos. Trabalhos futuros incluem o projeto elétrico, simulação no CADENCE e envio para fabricação já integrado ao Risc. Com o SoC pronto, poderão ser feitos outros testes do sistema que dependam deste relógio.

Os ensaios aqui desenvolvidos mostraram que esse projeto tem viabilidade técnica, e uma posterior implementação elétrica do relógio será facilitada com o auxílio deste projeto. A estrutura do Flip-Flop D bem como as portas lógicas já existem no software CADENCE. Assim a implementação do leiaute do relógio pode ser feita com a orientação do presente documento.

REFERÊNCIAS BIBLIOGRÁFICAS

[ASH2002] Ashenden, P.J. (2002), *"The Designer's Guide to VHDL"*, 2a Ed., Morgan Kaufmann, San Francisco, EUA.

[BES2004] Beserra, G. S. (2004), Projeto de Estruturas de Armazenamento Digital em um SoC para Controle de Irrigação, Dissertação de Mestrado, Universidade de Brasília, Faculdade de Tecnologia, Departamento de Engenharia Elétrica.

[COS2004] Costa, J. D. (2004), Implementação de um Processador RISC 16-Bits CMOS num Sistema em Chip, Dissertação de Mestrado, Universidade de Brasília, Faculdade de Tecnologia, Departamento de Engenharia Elétrica.

[MAR2004] MARRA, J.C. (2004); *Projeto de Interface Serial para Microprocessador de 16 bits*, (Projeto Final de Graduação em Engenharia Elétrica), Universidade de Brasília, Brasília, 2004.

[PIL2003] Pilla Jr, V. (2003), VHDL: Uma introdução, (Apostila elaborada pelo professor Valfredo Pilla Júnior), Centro Universitário Positivo, Curitiba, 2003.

[REL2001] Relatório do experimento de Sistemas Digitais 2, Departamento de Engenharia Elétrica, UnB, 2/2001.

[WAK2000] Wakerly , J. F. (2000), "Digital Design Principles and Practices", 3ª Ed., Prentice Hall, New Jersey, EUA.

www.symphonyeda.com (VHDL Simili)

www.xilinx.com (ISE 7.1)

APENDICE A – CÓDIGOS VHDL DOS MÓDULOS

1. FLIP-FLOP D SET/RESET

```
entity Flipflop_DSR is

                  generic (atrasoff: Time:= 30 ns);
                  port ( D,CLK,SET,RESET: in std_logic;
                                          Q,QN: out std_logic);

end Flipflop_DSR;


architecture Behavioral of Flipflop_DSR is

begin

      ffD:process(D,CLK,SET,RESET)
            variable sgnQ,asgnQ: std_logic:='0';

      begin

      if RESET = '1' then
                  sgnQ := '0';
      elsif SET = '1' then
                  sgnQ := '1';
      end if;

      if falling_edge (CLK)then
                  asgnQ := D;
      end if;

      if RESET = '1' or SET = '1' then
            Q <= sgnQ after atrasoff;
            QN <= not sgnQ after atrasoff;
            elsif falling_edge (CLK) then
            Q <= asgnQ after atrasoff;
            QN <= not asgnQ after atrasoff;
      end if;

      end process ffD;

end Behavioral;
```

2. FUNÇÃO LOAD

```
entity Func_load is

      port (I,LOD,MR: in std_logic;
                        SETLD,RESETLD: out std_logic);
end Func_load;

architecture Behavioral of Func_load is

      component INV_b
            port (I: in std_logic;
```

```
                        O: out std_logic);
      end component;

      component AND2_b
            port (I0,I1: in std_logic;
                        O: out std_logic);
      end component;

      component OR2_b
            port (I0,I1: in std_logic;
                        O: out std_logic);
      end component;

      signal MRN,LOD_MRN,INOT,LOD_IN: std_logic;

begin
      P1: INV_b port map (MR,MRN);
      P2: INV_b port map (I,INOT);
      P3: AND2_b port map (MRN,LOD,LOD_MRN);
      P4: AND2_b port map (LOD_MRN,I,SETLD);
      P5: AND2_b port map (LOD,INOT,LOD_IN);
      P6: OR2_b port map (LOD_IN,MR,RESETLD);

end Behavioral;
```

3. FUNÇÃO DO D1 DO CONTADOR 0-5

```
entity funcD1_C05 is
      port (A,B,C,D,E:  in std_logic;
                  O:                      out std_logic);
end funcD1_C05;

architecture Behavioral of funcD1_C05 is

      component AND2_b
            port (I0,I1: in std_logic;
                        O: out std_logic);
      end component;

      component AND3_b
            port (I0,I1,I2: in std_logic;
                        O: out std_logic);
      end component;

      component OR2_b
            port (I0,I1: in std_logic;
                        O: out std_logic);
      end component;

      signal A_B,C_D_E: std_logic;

begin

      P1: AND2_b  port map    (B,A,A_B);
      P2: AND3_b  port map (E,D,C,C_D_E);
      P3: OR2_b   port map    (C_D_E,A_B,O);

end Behavioral;
```

4. FUNÇÃO DO D2 DO CONTADOR 0-5

```
entity funcD2_C05 is
      port (A,B,C,D:    in std_logic;
                  O:                out std_logic);

end funcD2_C05;

architecture Behavioral of funcD2_C05 is

      component AND2_b
            port (I0,I1: in std_logic;
                        O: out std_logic);
      end component;

      component OR2_b
            port (I0,I1: in std_logic;
                        O: out std_logic);
      end component;

      signal A_B,C_D: std_logic;

begin

      P1: AND2_b  port map    (B,A,A_B);
      P2: AND2_b  port map    (D,C,C_D);
      P3: OR2_b   port map    (C_D,A_B,O);

end Behavioral;
```

5. FUNÇÃO DO D2 DO CONTADOR 0-9

```
entity funcD2_C09 is
      port (A,B,C,D,E,F,G:    in std_logic;
                                          O:    out std_logic);
end funcD2_C09;

architecture Behavioral of funcD2_C09 is

      component AND2_b
            port (I0,I1: in std_logic;
                        O: out std_logic);
      end component;

      component AND3_b
            port (I0,I1,I2: in std_logic;
                        O: out std_logic);
      end component;

      component OR3_b
            port (I0,I1,I2: in std_logic;
                        O: out std_logic);
```

```
      end component;

      signal A_B_C,D_E,F_G: std_logic;

begin

      P1: AND3_b  port map (C,B,A,A_B_C);
      P2: AND2_b  port map (E,D,D_E);
      P3: AND2_b  port map (G,F,F_G);
      P4: OR3_b   port map (F_G,D_E,A_B_C,O);

end Behavioral;
```

6. CONTADOR 0-2

```
entity cont02 is
      port (Icont02: in std_logic_vector (1 downto 0);
                  LODC,MRC,CP: in std_logic;
                  Lcont02: out std_logic_vector (1 downto 0));

end cont02;

architecture Behavioral of cont02 is

      component func_load
            port (I,LOD,MR: in std_logic;
                        SETLD,RESETLD: out std_logic);
      end component;

      component Flipflop_DSR
            port ( D,CLK,SET,RESET: in std_logic;
                                          Q,QN: out std_logic);
      end component;

      component AND2_b
            port (I0,I1: in std_logic;
                        O: out std_logic);
      end component;

      signal SETLD1,SETLD0,RESETLD1,RESETLD0,Q1,QN1,Q0,QN0,QN1_QN0,I1,I0:
STD_LOGIC;

begin
      I1 <= Icont02(1);
      I0 <= Icont02(0);
      U1: func_load port map (I1,LODC,MRC,SETLD1,RESETLD1);
      U2: func_load port map (I0,LODC,MRC,SETLD0,RESETLD0);
      U3: Flipflop_DSR port map (Q0,CP,SETLD1,RESETLD1,Q1,QN1);
      U4: Flipflop_DSR port map (QN1_QN0,CP,SETLD0,RESETLD0,Q0,QN0);
      P1: AND2_b port map (QN0,QN1,QN1_QN0);

      Lcont02(1) <= Q1;
      Lcont02(0) <= Q0;

end Behavioral;
```

7. CONTADOR 0-5

```
entity cont05 is
      port (Icont05:                in std_logic_vector(2 downto 0);
                  LODC,MRC,CP:      in std_logic;
                  Lcont05:                out std_logic_vector (2 downto
0);
                  carry05:                out std_logic);
end cont05;

architecture Behavioral of cont05 is

      component func_load
            port (I,LOD,MR:         in std_logic;
                        SETLD,RESETLD:    out std_logic);
      end component;

      component funcD2_C05
            port (A,B,C,D:    in std_logic;
                        O:                out std_logic);
      end component;

      component funcD1_C05
            port (A,B,C,D,E:  in std_logic;
                        O:                          out std_logic);
      end component;

      component Flipflop_DSR
            port ( D,CLK,SET,RESET: in std_logic;
                                                Q,QN: out std_logic);
      end component;

      signal
SETLD2,RESETLD2,SETLD1,RESETLD1,SETLD0,RESETLD0,D2_C05,D1_C05,Q2,Q1,Q0,QN
2,QN1,QN0,I2,I1,I0: std_logic;

begin

      I2 <= Icont05(2);
      I1 <= Icont05(1);
      I0 <= Icont05(0);

      U1: func_load           port map (I2,LODC,MRC,SETLD2,RESETLD2);
      U2: func_load           port map (I1,LODC,MRC,SETLD1,RESETLD1);
      U3: func_load           port map (I0,LODC,MRC,SETLD0,RESETLD0);
      U4: funcD2_C05          port map (Q2,QN0,Q0,Q1,D2_C05);
      U5: funcD1_C05          port map (Q1,QN0,QN2,Q0,QN1,D1_C05);
      U6: Flipflop_DSR  port map (D2_C05,CP,SETLD2,RESETLD2,Q2,QN2);
      U7: Flipflop_DSR  port map (D1_C05,CP,SETLD1,RESETLD1,Q1,QN1);
      U8: Flipflop_DSR  port map (QN0,CP,SETLD0,RESETLD0,Q0,QN0);

      carry05 <= Q2;
      Lcont05(2) <= Q2;
      Lcont05(1) <= Q1;
      Lcont05(0) <= Q0;

end Behavioral;
```

8. CONTADOR 0-9

```
entity cont09 is
```

```
      port (Icont09:           in  std_logic_vector (3 downto 0);
                  LODC,MRC,CP: in  std_logic;
                  Lcont09:          out std_logic_vector (3 downto 0);
                  carry09:          out std_logic);
end cont09;

architecture Behavioral of cont09 is

      component func_load
            port (I,LOD,MR:         in std_logic;
                        SETLD,RESETLD:    out std_logic);
      end component;

      component funcD2_C09
            port (A,B,C,D,E,F,G:    in std_logic;
                        O:                              out std_logic);
      end component;

      component funcD1_C05
            port (A,B,C,D,E:  in std_logic;
                        O:                        out std_logic);
      end component;

      component Flipflop_DSR
            port ( D,CLK,SET,RESET: in std_logic;
                                                Q,QN: out std_logic);
      end component;

      signal
SETLD3,SETLD2,SETLD1,SETLD0,RESETLD3,RESETLD2,RESETLD1,RESETLD0,D3_C09,D2
_C09,D1_C09,Q3,Q2,Q1,Q0,QN3,QN2,QN1,QN0,I3,I2,I1,I0: std_logic;




begin

      I3 <= Icont09(3);
      I2 <= Icont09(2);
      I1 <= Icont09(1);
      I0 <= Icont09(0);

      U1:  func_load          port map (I3,LODC,MRC,SETLD3,RESETLD3);
      U2:  func_load          port map (I2,LODC,MRC,SETLD2,RESETLD2);
      U3:  func_load          port map (I1,LODC,MRC,SETLD1,RESETLD1);
      U4:  func_load          port map (I0,LODC,MRC,SETLD0,RESETLD0);
      U5:  funcD1_C05   port map (Q3,QN0,Q0,Q1,Q2,D3_C09);
      U6:  funcD2_C09   port map (QN2,Q0,Q1,QN0,Q2,Q2,QN1,D2_C09);
      U7:  funcD1_C05   port map (Q1,QN0,QN3,Q0,QN1,D1_C09);
      U8:  Flipflop_DSR port map (D3_C09,CP,SETLD3,RESETLD3,Q3,QN3);
      U9:  Flipflop_DSR port map (D2_C09,CP,SETLD2,RESETLD2,Q2,QN2);
      U10: Flipflop_DSR port map (D1_C09,CP,SETLD1,RESETLD1,Q1,QN1);
      U11: Flipflop_DSR port map (QN0,CP,SETLD0,RESETLD0,Q0,QN0);

      carry09 <= Q3;
      Lcont09(3) <= Q3;
      Lcont09(2) <= Q2;
      Lcont09(1) <= Q1;
      Lcont09(0) <= Q0;
```

```
end Behavioral;
```

9. MÓDULO DE SEGUNDO E/OU MINUTO

```
entity seg_min is
      port (SEG_MIN_UNI: in std_logic_vector (3 downto 0);
                  SEG_MIN_DEZ: in std_logic_vector (2 downto 0);
                  LOD,MR,CP: in std_logic;
                  UNI: out std_logic_vector (3 downto 0);
                  DEZ: out std_logic_vector (2 downto 0);
                  carry: out std_logic);
end seg_min;

architecture Behavioral of seg_min is

      component cont05
            port (Icont05: in std_logic_vector (2 downto 0);
                        LODC,MRC,CP: in std_logic;
                        Lcont05: out std_logic_vector (2 downto 0);
                        carry05: out std_logic);
      end component;

      component cont09
            port (Icont09: in std_logic_vector (3 downto 0);
                        LODC,MRC,CP: in std_logic;
                        Lcont09: out std_logic_vector (3 downto 0);
                        carry09: out std_logic);
      end component;

      signal CP05: std_logic;

begin

      U1: cont09  port map (SEG_MIN_UNI,LOD,MR,CP,UNI,CP05);
      U2: cont05  port map (SEG_MIN_DEZ,LOD,MR,CP05,DEZ,carry);

end Behavioral;
```

10. FUNÇÃO 24 HORAS

```
entity func_24h is
      port (Lcont9_1,Lcont9_0,Lcont2_1,cp,MR,carry09: in std_logic;
                  MRC09,CP02: out std_logic);
end func_24h;

architecture Behavioral of func_24h is

      component Flipflop_DSR
            port ( D,CLK,SET,RESET: in std_logic;
                                          Q,QN: out std_logic);
      end component;

      component AND2_b
            port (I0,I1: in std_logic;
                        O: out std_logic);
```

```
      end component;

      component AND3_b
            port (I0,I1,I2: in std_logic;
                  O: out std_logic);
      end component;

      component OR2_b
            port (I0,I1: in std_logic;
                  O: out std_logic);
      end component;

      signal L9_L2,MR9,R9,MR9n: std_logic;

begin

      P1: AND3_b  port map (Lcont9_0,Lcont9_1,Lcont2_1,L9_L2);
      P2: OR2_b   port map (L9_L2,carry09,CP02);
      P3: OR2_b   port map (R9,MR,MRC09);
      P4: AND2_b  port map (MR9,Lcont2_1,R9);

      U1: Flipflop_DSR  port map (L9_L2,CP,'0','0',MR9,MR9n);

end Behavioral;
```

11. MÓDULO DE HORA

```
entity hora is
      port (HOR_UNI: in std_logic_vector (3 downto 0);
                  HOR_DEZ: in std_logic_vector (1 downto 0);
                  LOD,MR,CP: in std_logic;
                  UNI: out std_logic_vector (3 downto 0);
                  DEZ: out std_logic_vector (1 downto 0));
end hora;

architecture Behavioral of hora is

      component cont02
            port (Icont02: in std_logic_vector (1 downto 0);
                        LODC,MRC,CP: in std_logic;
                        Lcont02: out std_logic_vector (1 downto 0));

      end component;

      component cont09
            port (Icont09: in std_logic_vector (3 downto 0);
                        LODC,MRC,CP: in std_logic;
                        Lcont09: out std_logic_vector (3 downto 0);
                        carry09: out std_logic);
      end component;

      component func_24h
            port (Lcont9_1,Lcont9_0,Lcont2_1,cp,MR,carry09: in std_logic;
                        MRC09,CP02: out std_logic);
      end component;

      signal car9,cp2,mrc9,CAR2: std_logic;
      signal L9: std_logic_vector (3 downto 0);
      signal L2: std_logic_vector (1 downto 0);
```

```
begin

      U1: cont02   port map (HOR_DEZ,LOD,MR,cp2,L2,CAR2);
      U2: cont09   port map (HOR_UNI,LOD,mrc9,CP,L9,car9);
      U3: func_24h port map (L9(1),L9(0),L2(1),CP,MR,car9,mrc9,cp2);

      DEZ <= L2;
      UNI <= L9;

end Behavioral;
```

12. RELÓGIO

```
entity relogio is
      port (HOR1LD: in std_logic_vector (1 downto 0);
                  MIN1LD,SEG1LD: in std_logic_vector (2 downto 0);
                  HOR0LD,MIN0LD,SEG0LD: in std_logic_vector (3 downto 0);
                  LOD,MR,CP1Hz: in std_logic;
                  HOR1: out std_logic_vector (1 downto 0);
                  MIN1,SEG1: out std_logic_vector (2 downto 0);
                  HOR0,MIN0,SEG0: out std_logic_vector (3 downto 0));

end relogio;

architecture Behavioral of relogio is

      component seg_min
            port (SEG_MIN_UNI: in std_logic_vector (3 downto 0);
                        SEG_MIN_DEZ: in std_logic_vector (2 downto 0);
                        LOD,MR,CP: in std_logic;
                        UNI: out std_logic_vector (3 downto 0);
                        DEZ: out std_logic_vector (2 downto 0);
                        carry: out std_logic);
      end component;

      component hora
            port (HOR_UNI: in std_logic_vector (3 downto 0);
                        HOR_DEZ: in std_logic_vector (1 downto 0);
                        LOD,MR,CP: in std_logic;
                        UNI: out std_logic_vector (3 downto 0);
                        DEZ: out std_logic_vector (1 downto 0));
      end component;

      signal CPhora,CPmin: std_logic;

begin

      U1: seg_min port map (SEG0LD,SEG1LD,LOD,MR,CP1Hz,SEG0,SEG1,CPmin);
      U2: seg_min port map (MIN0LD,MIN1LD,LOD,MR,CPmin,MIN0,MIN1,CPhora);
      U3: hora    port map (HOR0LD,HOR1LD,LOD,MR,CPhora,HOR0,HOR1);

end Behavioral;
```