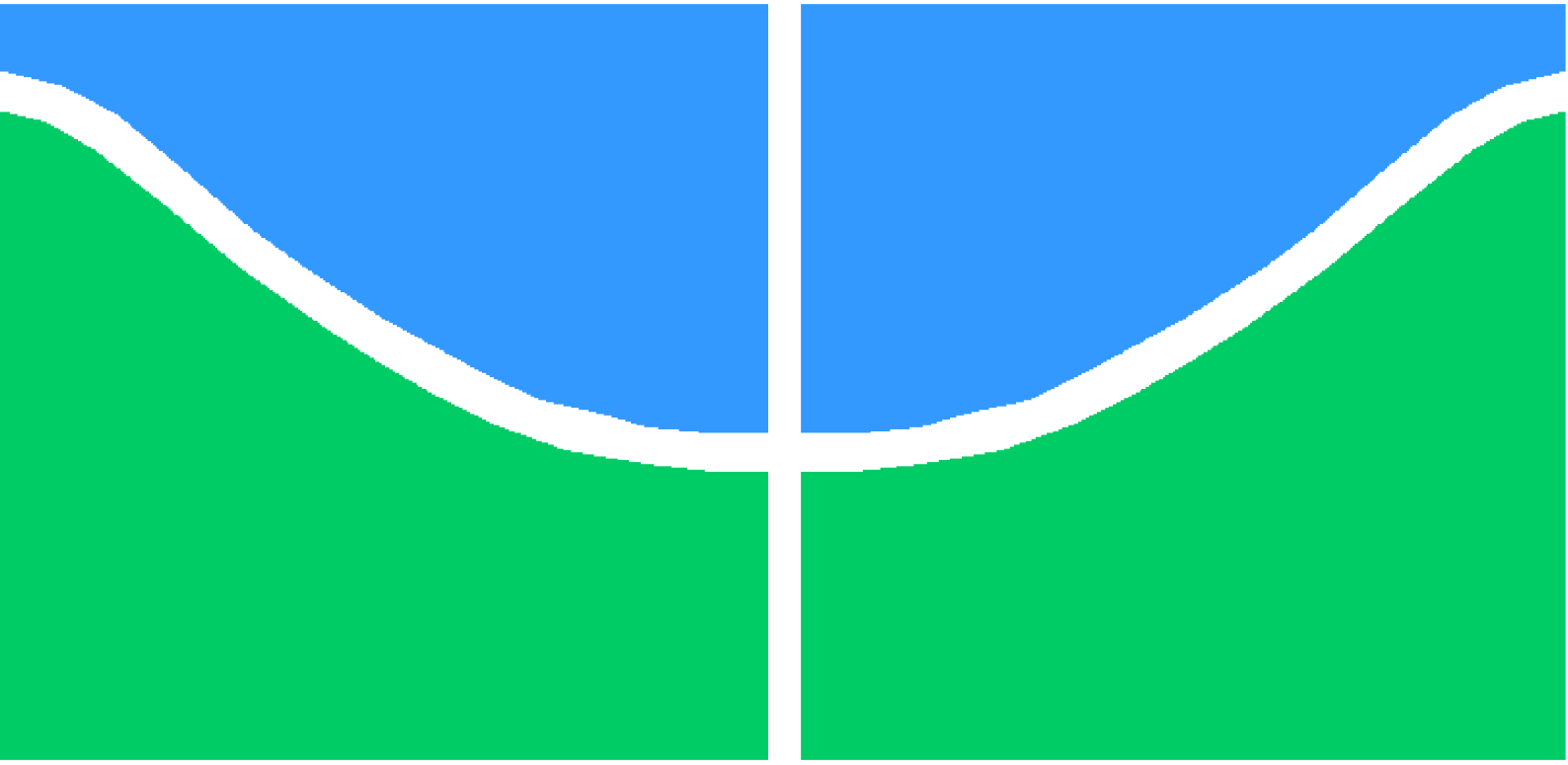

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia Eletronica

Implementação de um Protocolo Anticolisão para uma Tag Passiva de RFID

Autor: Henrique Malloni de Faria
Orientador: Prof. Gilmar Silva Beserra

Brasília, DF
2017

Henrique Malloni de Faria

Implementação de um Protocolo Anticolisão para uma Tag Passiva de RFID

Monografia submetida ao curso de graduação em (Engenharia Eletronica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletronica).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Gilmar Silva Beserra

Coorientador: Prof. Wellington Avelino do Amaral

Brasília, DF

2017

Henrique Malloni de Faria
Implementação de um Protocolo Anticolisão para uma Tag Passiva de RFID/ Henrique Malloni de Faria. – Brasília, DF, 2017-
91 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Gilmar Silva Beserra

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2017.

1. RFID. 2. arquitetura. I. Prof. Gilmar Silva Beserra. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Implementação de um Protocolo Anticolisão para uma Tag Passiva de RFID

CDU 02:141:005.6

Henrique Malloni de Faria

Implementação de um Protocolo Anticolisão para uma Tag Passiva de RFID

Monografia submetida ao curso de graduação em (Engenharia Eletronica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletronica).

Trabalho aprovado. Brasília, DF, 07 de junho de 2017:

Prof. Gilmar Silva Beserra
Orientador

Prof. Wellington Avelino do Amaral
Convidado 1

Prof. Renato Coral Sampaio
Convidado 2

Brasília, DF
2017

Aos meus pais, meu irmão e a toda a minha família que, com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa de minha vida.

Agradecimentos

Agradeço a Deus por minha vida, família e amigos. Agradeço a todos que direta ou indiretamente fizeram parte da minha formação,o meu muito obrigado.

"O insucesso é apenas uma oportunidade
para recomeçar de novo com mais inteligência.
(Henry Ford)

Resumo

A tecnologia RFID vem sendo cada vez mais utilizada em diversas áreas, tais como segurança, controle de acessos e identificação de objetos, como alternativa à tecnologia de código de barras. Nesse contexto, a implementação de uma tag passiva de RFID, operando na faixa de 13,56 MHz, vem sendo tema de diversos trabalhos de graduação e iniciação científica na Faculdade do Gama da Universidade de Brasília. Dentre eles, pode-se citar (FILHO, 2014), onde foram feitas a modelagem, simulação e implementação do bloco de *front end* analógico da tag em tecnologia TSMC 0.18 μm, e (NETO, 2015), onde foram realizadas a caracterização dos blocos implementados e o projeto dos blocos individuais restantes. Neste trabalho, foram abordados o envio de dados entre o leitor e a tag e a parte de controle digital. Com base nas especificações dos trabalhos anteriores, foi considerado para a comunicação entre o leitor e a tag o protocolo ISO/IEC 14443, que é um padrão internacional para Smart Cards sem contato, operando a 13,56 MHz em estreita proximidade de aproximadamente 10 cm entre o leitor e a tag. Mais especificamente, considerando que inicialmente a tag passiva de RFID enviará apenas o número do ID para o leitor e que poderá ocorrer colisão de dados quando o leitor acionar mais de uma tag, um protocolo anticolisão foi descrito e simulado em Verilog.

Palavras-chaves: Processador. Tag. RFID.

Abstract

RFID technology has been increasingly used in a number of areas, such as security, access control, and object identification, as an alternative to bar code technology. In this context, the implementation of a passive RFID tag, operating in the 13.56 MHz range, has been the subject of several undergraduate studies at the Faculty of Gama of the University of Brasília. Among them, we can mention (FILHO, 2014), where the modeling, simulation and implementation of the analogic front end block of the tag using the 0.18 μm technology were performed, and (NETO, 2015), where the characterization of the blocks and the design of the remaining individual blocks were presented. In this work, we approached the data communication between the reader and the tag and the digital control part. Based on the specifications of the previous works, we chose to use the ISO / IEC 14443 protocol, which is an international standard for contactless Smart Cards, operating at 13.56 MHz in close proximity to approximately 10 cm between the reader and the tag. More specifically, and considering that initially the RFID passive tag will send only the ID number to the reader and that data collision may occur when the reader triggers more than one tag, an anti-collision protocol has been described and simulated in Verilog.

Key-words: Processor. tag. RFID.

Lista de ilustrações

Figura 1 – (a) Denominação dos blocos da tag de RFID (b) Detalhamento do bloco *front end* analógico(FILHO, 2014). 25
Figura 2 – Ilustração de um sistema de RFID genérico(PUHLMANN, 2015). . . . 30
Figura 3 – Princípio de acoplamento do sistema indutivo(SANGREMAN, 2003) . 31
Figura 4 – Princípio de funcionamento do acoplamento *backscatter*(SANGREMAN, 2003) 32
Figura 5 – Diagrama de blocos da parte digital(ZHANG WEIPING JING, 2014) . 35
Figura 6 – Arquitetura interna do bloco digital(BARAL1, 2012) 35
Figura 7 – Diagrama de blocos de uma tag RFID com aplicação biomédica(KOTHAMASU, 2012) 36
Figura 8 – Diagrama de blocos do bloco digital(KOTHAMASU, 2012) 37
Figura 9 – Diagrama de estados do controlador do bloco de aplicação(KOTHAMASU, 2012) 37
Figura 10 – Modulação entre o leitor e a tag para o cartão tipo A(ISO/IEC, 2005) 40
Figura 11 – Modulação entre o leitor e a tag para o cartão tipo B(ISO/IEC, 2005) . 40
Figura 12 – Codificação NRZ-L e NRZ-I(TUTORIALSPOINT.COM, 2015) 41
Figura 13 – Formato de um byte de dados(ISO/IEC, 2005) 44
Figura 14 – Requerimentos do SOF e EOF(ISO/IEC, 2005). 45
Figura 15 – Frame de comunicação do leitor(ISO/IEC, 2005) 45
Figura 16 – Frame de comunicação da tag(ISO/IEC, 2005) 46
Figura 17 – tempo de guarda TR0(ISO/IEC, 2005). 47
Figura 18 – Tempo de atraso do frame TR2(ISO/IEC, 2005). 47
Figura 19 – Ordem do byte CRC_B(ISO/IEC, 2005). 48
Figura 20 – Hardware do gerador CRC_B(BOMMENA, 2008). 48
Figura 21 – Cálculo para gerar um CRC(SCHMIDT, 2000). 49
Figura 22 – Checando a mensagem por um erro CRC(SCHMIDT, 2000). 49
Figura 23 – Diagrama de transição dos estados da tag(ISO/IEC, 2005). 51
Figura 24 – Exemplo da anticolisão *Timeslot*(ISO/IEC, 2005). 53
Figura 25 – Exemplo da anti-colisão *Timeslot*(ISO/IEC, 2005). 54
Figura 26 – Comando e Resposta Slot-Marker(ISO/IEC, 2005). 55
Figura 27 – Formato da resposta ATQB(ISO/IEC, 2005). 57
Figura 28 – Definição do byte do Procoloco ATQB(ISO/IEC, 2005). 57
Figura 29 – Comando e resposta ATTRIB(ISO/IEC, 2005). 59
Figura 30 – Definição do byte Param do comando ATTRIB(ISO/IEC, 2005). 60
Figura 31 – Comando e Resposta HLTB(ISO/IEC, 2005). 61
Figura 32 – Bloco always da maquina de estados e os estados 0000 e 0001 63

Figura 33 – Numeração em decimal dos blocos do diagrama de transição dos estados da tag 64
Figura 34 – Estados 0010 e 0011 65
Figura 35 – Estados 0100 e 0101 66
Figura 36 – Estados 0110 e 0111 66
Figura 37 – Estados 1000, 1001 e 1010 67
Figura 38 – Estados 1011 e 1100 68
Figura 39 – Estado 1101 68
Figura 40 – Bloco always para armazenar as variáveis 69
Figura 41 – Decodificação e armazenamento do número de solt 70
Figura 42 – Gerador do Inteiro N 70
Figura 43 – Test bench do Bloco Digital 71
Figura 44 – Simulação do Bloco Digital no simulador ISim Parte 01 72
Figura 45 – Simulação do Bloco Digital no simulador ISim Parte 02 72
Figura 46 – Simulação do Bloco Digital no simulador ISim Parte 03 74
Figura 47 – Código verilog CRC 75
Figura 48 – Testbench CRC 75
Figura 49 – Simulação CRC no simulador ISim 76
Figura 50 – Código verilog CRC utilizando o loop for 76
Figura 51 – Simulação CRC utilizando o loop for no simulador ISim 77
Figura 52 – Código em verilog do Bloco Digital da tag 87
Figura 53 – Test Bench do Bloco Digital da tag 88
Figura 54 – Código verilog CRC 89
Figura 55 – Test Bench CRC 90
Figura 56 – Código verilog CRC utilizando o loop for 91
Figura 57 – Test Bench CRC utilizando o loop for 91

Lista de tabelas

Tabela 1 – Características e aplicações típicas por frequência de operação (MARTINS, 2005) 33
Tabela 2 – Descrições dos Padrões de ISOs para RFID (MARKHAM, 2012) 38
Tabela 3 – Tabela Verdade porta XOR 48
Tabela 4 – Critérios de correspondência AFI (ISO/IEC, 2005) 53
Tabela 5 – Definição da família do código AFI(ISO/IEC, 2005) 54
Tabela 6 – Codificação dos "N"Parâmentros da anti-colisão(ISO/IEC, 2005) 55
Tabela 7 – Codificação do número do Slot(ISO/IEC, 2005) 56
Tabela 8 – Codificação do tamanho máximo do Frame da tag no byte 2 do Protocolo(ISO/IEC, 2005) 58
Tabela 9 – Codificação do FWI no byte do Protocolo no ATQB(ISO/IEC, 2005) . 58
Tabela 10 – Codificação do tamanho máximo do Frame do leitor do Param 2(ISO/IEC, 2005) 60
Tabela 11 – Codificação do CID no Param 4 e resposta do ATTRIB(ISO/IEC, 2005) 61

Lista de abreviaturas e siglas

A	Unmodulated Signal Amplitude
ADC	Application Data Coding
AFI	Application Family Identifier
AID	Application Identifier Code
ASK	Amplitude Shift Keying Modulation
ATQB	Answer to Request, Type B
ATTRIB	PICC Selection Command, Type B
B	Modulated Signal Amplitude
BPSK	Binary Phase Shift Keying Modulation
CID	Card Identifier
CRC_B	Cyclic Redundancy Check Error Detection Code B
EGT	Extra Guard Time
EOF	End of Frame
EPC	Electronic Product Code
EEPROM	Electrically-Erasable Programmable Read-Only Memory
ETU	Elementary Time Unit
fc	Carrier Frequency = 13.56 MHz
FO	Frame Option
FPGA	Field Programmable Gate Array
fs	Subcarrier Frequency = fc/16 = 847.5 kHz
FWI	Frame Waiting Time Integer
FWT	Frame Waiting Time
HLTB	Halt Command, Type B

I^2C	Inter-Integrated Circuit
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronic Engineers
ISO	International Organization for Standardization
ID	Identification
kbps	Kilobits per Second
LSB	Least Significant Bit
MBLI	Maximum Buffer Length Index of PICC
MSB	Most Significant Bit
N	Number of Anticollision Slots
NAD	Node Address
NRZ-L	Non-Return to Zero (L for Level) Data Encoding
PCD	Proximity Coupling Device (leitor)
PICC	Proximity Integrated Circuit Card (tag)
PUPI	Pseudo Unique PICC Identifier
RAM	Random Access Memory
REQB	Request Command, Type B
RFU	Reserved for Future Use
ROM	Read Only Memory
RF	Radio Frequency
S	Slot Number
SOF	Start of Frame
TR0	Guard Time per 14443-2
TR1	Synchronization Time per 14443-2
TR2	PICC to PCD Frame Delay Time
VHDL	Very high-speed integrated circuit Hardware Description Language
WUPB	Wake Up Command, Type B

Sumário

1 INTRODUÇÃO 24
1.1 Motivação 24
1.2 Justificativa 25
1.3 Objetivos 26
1.4 Aspectos Metodológicos 27
1.5 Organização do Trabalho 27

2 FUNDAMENTAÇÃO TEÓRICA 28
2.1 Visão Geral de Sistemas RFID 28
2.2 Componentes do RFID 28
2.2.1 Tags ou transponders 28
2.2.2 Leitor com antena 29
2.3 Funcionamento do RFID 29
2.3.1 Acoplamento indutivo 30
2.3.2 Acoplamento *backscatter* 31
2.4 Faixas de Frequências do sistema RFID 32
2.5 Vantagens da tecnologia RFID 34
2.6 Arquiteturas de Processadores Banda Base 34
2.6.1 1ª arquitetura 34
2.6.2 2ª arquitetura 35
2.6.3 3ª arquitetura 36
2.7 Protocolos de Comunicação 37
2.7.1 Protocolo ISO/IEC 14443 38
2.7.1.1 Cartao Tipo A 39
2.7.1.2 Cartao Tipo B 40
2.8 Linguagem Verilog 41

3 ESPECIFICAÇÃO DO PROTOCOLO DE COMUNICAÇÃO ENTRE O LEITOR E A TAG 44
3.1 Formato de dados 44
3.1.1 Formato dos frames 45
3.1.2 Transmissão de Dados do leitor 45
3.1.3 Transmissão de dados da tag 46
3.1.4 Tempo de resposta 46
3.1.5 Detecção de erros CRC 47
3.2 Anticolisão 49

3.2.1 Anticolisão *Timeslot* . . . 52
3.2.2 Comando REQB/WUPB . . . 53
3.2.3 Comando Slot-Marker . . . 55
3.2.4 Resposta ATQB . . . 56
3.2.5 Comando ATTRIB . . . 59
3.2.6 Comando HLTB . . . 61
3.3 Definição do bloco a ser implementado . . . 62

4 RESULTADOS E DISCUSSÃO . . . 63
4.1 Descrição em Verilog . . . 63
4.1.1 Máquina de estado . . . 63
4.1.2 Armazenando as variáveis . . . 69
4.1.3 Gerador do Inteiro N . . . 70
4.1.4 *Testbench* do Bloco Anticolisão . . . 71
4.2 Código CRC . . . 74

5 CONCLUSÃO . . . 78

REFERÊNCIAS . . . 80

ANEXOS 83

ANEXO A – CÓDIGO VERILOG DO BLOCO DIGITAL . . . 84

ANEXO B – CÓDIGO VERILOG DO CRC . . . 89

,

1 Introdução

1.1 Motivação

RFID (Radio Frequency Identification – Identificação por Rádio Frequência) é uma tecnologia utilizada para identificar, rastrear e gerenciar desde produtos e documentos até animais, sem contato e sem a necessidade de um campo visual (COE, 2005). Tal tecnologia permite a captura automática de dados, para identificação de objetos com dispositivos eletrônicos, conhecidos como etiquetas eletrônicas, *tags* ou *transponders*, que emitem sinais de radiofrequência para leitores que captam estas informações. Esta tecnologia existe desde a década de 40 e veio para complementar a tecnologia de código de barras, bastante difundida no mundo (SANGREMAN, 2003), e uma das maiores vantagens dos sistemas baseados em RFID é o fato de permitir a codificação em ambientes hostis e em produtos onde o uso de código de barras não é eficaz.

Um sistema RFID digital funciona como um sistema poderoso de aquisição de dados com a vantagem de eliminação de intervenções humanas manuais e visuais, dinamizando assim o tempo de transições e assegurando eficiência e eficácia (COE, 2005). Por exemplo, RFIDs fixados nos pára-brisas de carros alugados podem guardar a identificação do veículo, de tal forma que as locadoras possam ter acesso a relatórios automaticamente usando leitores de RFID nos estacionamentos, além de ajudar na localização dos veículos. Este sistema também é usado em pedágios. O consumidor possui uma *tag* em seu carro e os leitores estáticos se situam no pedágio. Ao passar por ele, a tag é ativada e as informações dos carros são recolhidas e salvas em um banco de dados. No final do mês, a empresa responsável remete uma conta a ser paga em banco para o usuário, cobrando uma taxa proporcional ao número de vezes que ele passou no pedágio (SANGREMAN, 2003).

Na indústria, os sistemas de RIFD são largamente utilizados. Uma aplicação é a identificação de ferramentas, que no caso de grandes indústrias facilita os processos de manutenção e de substituição e administração das mesmas. Outro campo em que sistemas RFID podem melhorar tanto a rapidez quanto a qualidade do serviço, bem como ter um papel de segurança nas indústrias, é na identificação de recipientes, embalagens e garrafas, principalmente em produtos químicos e gases, onde um erro na hora de embalar pode causar sérios problemas. Hoje, a identificação de recipientes nas indústrias é feita através do código de barras, porém indústrias que trabalham com reações químicas precisam de um nível de confiabilidade muito maior do que o fornecido pelo código de barras. Além disso, as etiquetas eletrônicas são mais adaptáveis e robustas, podendo resistir a condições hostis como poeira, impactos, radiação, ácidos e temperaturas muito altas ou muito baixas, podendo variar de -40ºC a +120ºC (SANGREMAN, 2003).

As áreas de aplicação são igualmente as mais variadas: setor público (controle de passaportes, identificação de livros em bibliotecas), médico-hospitalar (identificação de pacientes, controle da administração de medicamentos, monitoramento de equipamentos hospitalares), automotivo (é utilizada como complemento do GPS para localização e rastreamento com aplicações na gestão de frotas), varejista (controle do fluxo de mercadoria), aéreo (identificação e movimentação de bagagens em aeroportos), entre outros (COE, 2005). Por isso, o desenvolvimento de projetos que relacionam RFID no campo da microeletrônica desperta interesse dos alunos neste setor de enorme aplicações no mercado, gerando oportunidades de emprego e novas possibilidades para projetos junto a indústrias que projetam circuitos integrados (COE, 2005).

1.2 Justificativa

Este trabalho se encontra no contexto de um projeto maior, cujo objetivo é implementar uma tag passiva de RFID para a frequência de 13,56 MHz. Sendo assim, partes da tag foram temas de diversos trabalhos de alunos do campus Gama da Universidade de Brasília.

A modelagem de um sistema completo de RFID usando SystemC e SystemC-AMS foi realizada em (GUIMARAES, 2013), com o objetivo de verificar sua funcionalidade através de simulações em alto nível de abstração. A mesma autora realizou também, como trabalho de iniciação científica, a descrição em nível RTL dos sub-blocos digitais da tag.

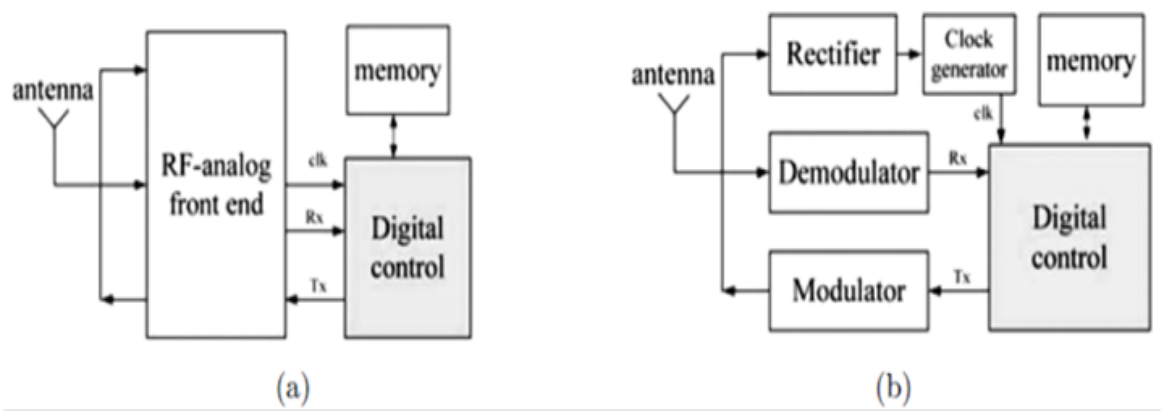

Figura 1 – (a) Denominação dos blocos da tag de RFID (b) Detalhamento do bloco *front end* analógico(FILHO, 2014).

Em (FILHO, 2014), os blocos individuais de *front end* analógico, mostrados nas figuras 1a e 1b e descritos a seguir, foram projetados e implementados.

- Retificador: Esse bloco retifica a entrada do sinal de RF e gera o nível DC de tensão necessário para alimentar os demais blocos.
- Demodulador: O demodulador de onda ASK (Amplitude Shift Keying) utiliza, basicamente, um estágio de comparação que irá identificar a média do sinal retificado e comparar com o próprio sinal. Essa comparação fornecerá um sinal em nível digital.
- Controlador Lógico: É a parte digital do sistema que controla os demais blocos. Basicamente ele irá controlar quando a tag deverá receber e enviar o sinal. Armazena o ID do tag que será enviado ao leitor pelo modulador.
- Clock Interno: É um oscilador que gerará um clock para alimentar a parte digital do circuito.
- Modulador: O modulador do sistema é um Modulador BPSK (Binary Phase Shift-keying), que gerará o sinal digital para uma modulação BPSK que enviará o sinal para o reader(FILHO, 2014).

Foi utilizada uma metodologia *top-down*, ou seja, inicialmente o sistema foi validado através da modelagem em Verilog-AMS, uma linguagem que permite a simulação de sistemas de sinais mistos, e posteriormente em nível de circuito. Finalmente, os layouts foram gerados na tecnologia TSMC 0.18 μm e enviados para fabricação.

Em (NETO, 2015), foi feita a caracterização dos blocos implementados em (FILHO, 2014) por meio da realização de uma placa de teste para o chip fabricado. Além disso, os blocos restantes de *front end*, tais como *charge pump* e LDO (*Low Dropout circuit*) foram projetados, simulados e implementados.

Considerando que a tag deve se comunicar com o leitor e que essa parte não foi abordada em nenhum dos trabalhos citados, o foco deste trabalho é a comunicação entre o leitor e a tag. Para isso, considerou-se como estudo de caso um processo de identificação no sistema RFID, ou seja, um processo no qual a informação a ser enviada para o leitor é somente o ID da tag. Sendo assim, o leitor deve enviar uma mensagem requisitando o ID das tags que se encontram dentro de seu alcance.

Um dos problemas desse processo ocorre quando duas ou mais tags respondem ao mesmo tempo, gerando a colisão de sinais e impedindo o reconhecimento dos IDs enviados. Por isso, é necessário utilizar um protocolo anticolisão para reduzir ou resolver os conflitos.

1.3 Objetivos

O objetivo geral deste trabalho é descrever um protocolo anticolisão em Verilog utilizando o software ISE Design Suite da Xilinx e testar o funcionamento do mesmo

por meio de simulações. Com isso, serão adquiridos conhecimentos em microeletrônica de radiofrequência que favorecem uma formação acadêmica mais completa.

Como objetivos específicos, tem-se a definição do protocolo de comunicação que será utilizado, o estudo da comunicação entre o leitor e a tag com base no protocolo definido, a especificação e descrição do protocolo anticolisão e, finalmente, a validação por meio de simulações.

1.4 Aspectos Metodológicos

Inicialmente, foram realizados um estudo dos trabalhos anteriores e uma revisão bibliográfica sobre o funcionamento do sistema RFID. A proposta original era o projeto e implementação em ASIC dos blocos do processador banda base da tag de RFID utilizando o fluxo de projeto para circuitos integrados digitais. Entretanto, devido à indisponibilidade da licença do software necessária para realizar o projeto, a proposta foi adaptada e o foco do trabalho foi alterado para a comunicação entre o leitor e a tag, mais especificamente para a definição, descrição e simulação do protocolo anticolisão, etapas que podem ser executadas usando ferramentas de acesso gratuito.

Após a redefinição dos objetivos do trabalho, foi utilizada a metodologia *top down*, partindo-se da especificação funcional do bloco a ser implementado para a descrição em nível RTL, na qual foi utilizada a linguagem de descrição de *hardware* Verilog. Essa linguagem foi escolhida para facilitar a realização de simulações mistas com os blocos do *front end* analógico descritos nos trabalhos anteriores. Em seguida, foi realizada a simulação para validar a funcionalidade do bloco e verificar se a mesma estava de acordo com a especificação. As seções a seguir descrevem os fundamentos da linguagem Verilog e a ferramenta utilizada para simulação.

1.5 Organização do Trabalho

O trabalho se divide em cinco capítulos. Inicialmente é apresentada uma introdução geral que define, contextualiza e justifica o problema. Após essa introdução, no capítulo 2 é apresentada uma fundamentação teórica mostrando uma visão geral do sistema RFID, juntamente com arquitetura de processadores banda base e protocolos de comunicação. Em seguida, no capítulo 3 são descritos as especificações do projeto que são mostradas através da descrição do envio de dados entre o leitor e a tag. Também neste capítulo será mostrado os procedimentos da anti-colisão. Os resultados e discussões são apresentados no capítulo 4 e, finalmente, as conclusões são mostradas no capítulo 5.

2 Fundamentação Teórica

2.1 Visão Geral de Sistemas RFID

RFID (Radio Frequency Identification – Identificação por Rádio Freqüência) é uma tecnologia utilizada para identificar, rastrear e gerenciar desde produtos e documentos até animais ou mesmo indivíduos, sem contato e sem a necessidade de um campo visual. Um sistema RFID digital traz a vantagem de eliminação de intervenções humanas manuais e visuais, dinamizando assim o tempo de transições e proporcionando eficiência. O baixo custo de implementação e a facilidade de utilização são fatores determinantes para a sua alta aceitação.

Apesar de o sistema RFID ter algumas funções similares ao do código de barras, cada vez mais se comprova que o RFID, com suas etiquetas inteligentes, completa o código de barras. A perspectiva é que uma grande revolução na gestão da cadeia de suprimentos será proporcionada através da larga adoção de RFID, fornecendo informações para o seu gerenciamento (COE, 2005).

2.2 Componentes do RFID

Um sistema de RFID é basicamente composto por dois componentes: tag(ou transponder), que se situa no objeto a ser identificado, e leitor com antena, que é um dispositivo de captura ou de captura/transmissão de dados. As subseções a seguir descrevem, de forma resumida, cada um desses componentes.

2.2.1 Tags ou transponders

As tags são os dispositivos que contêm a informação sobre o item a que desejam representar. Elas podem ser somente de leitura ou de leitura e escrita. Consistem normalmente de uma antena e um microchip eletrônico. Estão disponíveis em diversos formatos (pastilhas, argolas, cartões, etc), tamanhos e materiais utilizados para o seu encapsulamento, que podem ser o plástico, vidro, epóxi, etc. O tipo de tag também é definido conforme a aplicação, ambiente de uso e desempenho (SANGREMAN, 2003).

Há geralmente dois tipos de tags: ativa e passiva. Tags ativas têm a sua própria fonte de alimentação na forma de uma bateria. Isto lhes permite transmitir dados com maior níveis de potência e, assim, serem lidas ou escritas a distâncias maiores. Como resultado, estas tags são relativamente grandes e, devido à tecnologia, mais caras do que as passivas.

Tags passivas obtêm sua energia a partir de um campo eletromagnético do dispositivo de leitura. Isto significa que elas não necessitam de sua própria fonte de alimentação, o que as torna muito pequenas e econômicas. A desvantagem é que elas têm menor alcance que as tags ativas e exigem um leitor mais potente para a leitura dos dados.

2.2.2 Leitor com antena

A antena constitui parte vital no processo de leitura/escrita da tag. É ela que gera e emite as ondas eletromagnéticas que irão servir de meio de energização e comunicação com tag. É também a principal responsável pela troca de informações entre a tag e o leitor. O leitor opera pela emissão de um campo eletromagnético (radiofreqüência), no qual é a fonte que alimenta a tag, que por sua vez, responde ao leitor com o conteúdo de sua memória (SANGREMAN, 2003). O leitor então decodifica os dados que estão codificados na tag, passando-os para um computador realizar o processamento (PINHEIRO, 2004).

As antenas são fabricadas em diversos formatos e tamanhos com configurações e características diferentes, cada uma para um tipo de aplicação (SANGREMAN, 2003). Os sistemas RFID utilizam dois métodos de acoplamento: proximidade eletromagnética ou indutiva e propagação por ondas eletromagnéticas. O princípio é similar ao de um transformador, onde a "antena" transfere energia e os dados são trocados pelos dois elementos (PORTO, 2005).

2.3 Funcionamento do RFID

Para que ocorra a comunicação por RFID, são necessários basicamente dois componentes: a tag e um leitor com antena (Figura 2). Quando aproximamos uma tag passiva do leitor, o campo de radiofrequências do leitor alimenta a tag. As tags passivas dependem da energia da onda eletromagnética incidente para funcionar. Portanto, a tag passiva absorve uma quantidade dessa energia e envia dados, contidos na memória, que podem ser número de série, data de fabricação, entre outros dados. A antena, ao receber o sinal enviado pela tag, converte as ondas de rádio (filtragem) para informações digitais para serem lidas e compreendidas (processadas) por um computador, que processa e armazena informações da tag em um banco de dados.

Portanto, RFID é uma tecnologia que utiliza ondas eletromagnéticas, propagadas pelo ar, para transmitir dados armazenados em um microchip. Esse sistema possibilita saber onde estão estoques e mercadorias, informações de preço, prazo de validade, número do lote, entre outras, além de permitir uma relação mais estreita entre a linha de produção e os sistemas de informação da empresa. Para maior segurança, essas informações armazenadas podem ainda ser criptografadas e atualizadas remotamente (PINHEIRO, 2004).

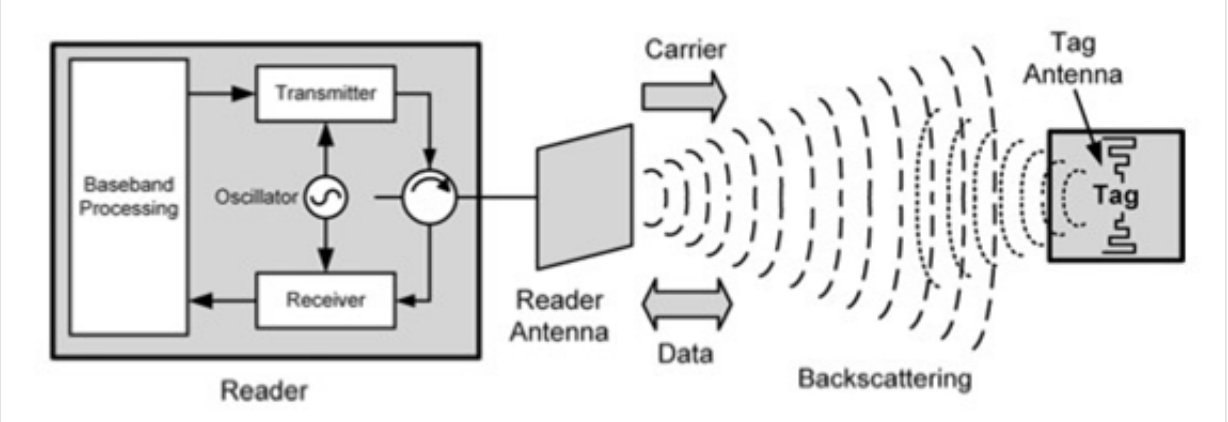

Figura 2 – Ilustração de um sistema de RFID genérico(PUHLMANN, 2015).

A conexão entre a tag e o leitor pode ser realizada de duas maneiras: acoplamento indutivo e acoplamento *backscatter*, que serão abordados com mais detalhes a seguir (SANGREMAN, 2003).

2.3.1 Acoplamento indutivo

Uma tag indutiva é composta de um microchip e uma bobina, que funciona como uma antena. Essas tags são geralmente passivas. Para a transmissão de energia, a bobina da antena do leitor gera um campo eletromagnético em alternância, de alta freqüência, que penetra a seção transversal tanto da bobina como da área a sua volta. Uma parte pequena do campo emissor interage com a bobina da antena da tag por indução magnética, gerando tensão na bobina da antena da tag. Esta tensão é retificada e serve como a fonte de alimentação para o microchip. A tensão na bobina da tag alcança um valor máximo devido ao amplificador de ressonância no circuito ressonante paralelo.

Um capacitor é conectado paralelamente à bobina da antena do leitor. A capacitância é selecionada de forma a combinar com a indutância da bobina da antena para dar forma a um circuito ressonante paralelo, ou seja, para se obter uma freqüência ressonante que corresponda com a freqüência da transmissão do leitor. Correntes muito elevadas são geradas pela bobina da antena do leitor pelo amplificador de ressonância no circuito ressonante paralelo, que pode ser usado para gerar a energia requerida do campo para a operação da tag remotamente.

Como descrito acima, o sistema indutivo é baseado em um transformador do tipo acoplamento entre a bobina do leitor e da tag. Porém, isto ocorre quando a distância entre as bobinas não excede 0.16 do comprimento de onda, de modo que a tag esteja situada nas proximidades da antena do transmissor. Este consumo de potência adicional da tag pode ser medido como a queda de tensão na resistência interna nas antenas do leitor através

da corrente da fonte à antena do leitor.

Conseqüentemente, a alternância da resistência na antena da tag efetua mudanças na tensão na antena do leitor e tem assim o efeito de uma modulação de amplitude de tensão da antena feito pela tag. Se a alternância do resistor for controlada por dados, então estes dados podem ser transferidos da tag ao leitor. Este tipo de transferência de dados é chamado de modulação de carga. Para recuperar os dados no leitor, a tensão medida na antena do leitor é retificada, realizando uma demodulação de um sinal modulado por amplitude. A Figura 3 mostra o princípio de acoplamento do sistema indutivo. (SANGREMAN, 2003).

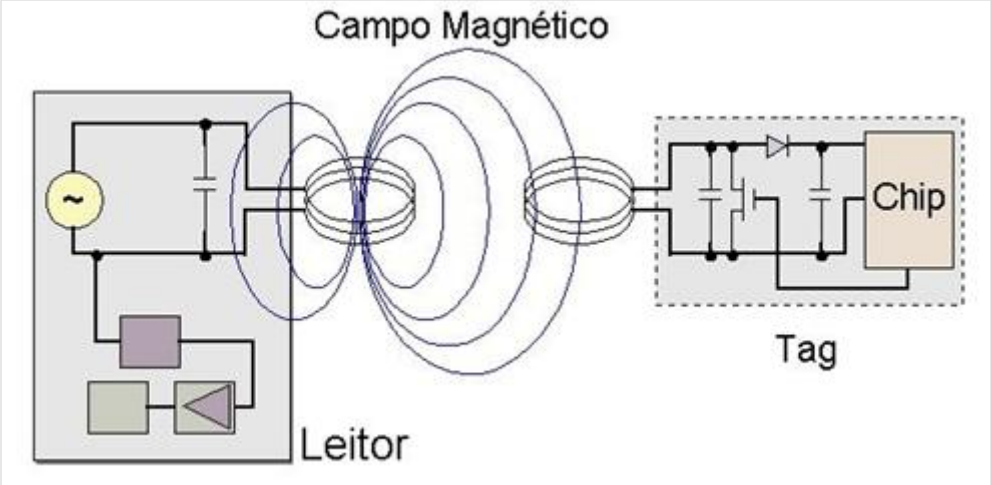

Figura 3 – Princípio de acoplamento do sistema indutivo(SANGREMAN, 2003)

2.3.2 Acoplamento *backscatter*

Pela tecnologia de radares, sabemos que as ondas eletromagnéticas são refletidas por objetos com dimensões superiores à metade do seu comprimento de onda. A eficiência com que um objeto reflete ondas eletromagnéticas é descrita pela sua seção transversal de reflexão. Este tipo de acoplamento pode ser visto na Figura 4.

A energia P1 é emitida pela antena do leitor, mas somente uma pequena parte desta energia alcança a antena da tag. A energia P1' (P1'< P1) é fornecida às conexões da antena como a tensão e, após a retificação feita pelos diodos D1 e D2, esta pode ser usada como voltagem inicial tanto para a desativação como para a ativação do modo econômico de energia. Os diodos usados são os diodos de barreira fraca de Schottky, que têm uma tensão particularmente baixa no ponto inicial. A tensão obtida pode também ser suficiente para servir como uma fonte de alimentação para distâncias curtas. Uma parte da energia entrante P1 ' é refletida pela antena e retornada como o sinal P2. As características da reflexão da antena podem ser influenciadas alterando a carga conectada à antena.

Com o intuito de transmitir dados do transponder ao leitor, um resistor RL é ligado conectado em paralelo com a antena, trabalhando de modo alternado em sincronia com a transmissão dos dados. A amplitude de P2 refletida do transponder pode assim ser modulada. O sinal P2 refletido pelo transponder é irradiado no espaço livre. Uma parte pequena deste é captado pela antena do leitor. O sinal refletido viaja através da conexão da antena do leitor no sentido contrário, e pode ser desacoplado usando um acoplador direcional, para depois ser transferido ao receptor de um leitor. Em média, o sinal do transmissor é aproximadamente dez vezes mais forte do que o recebido de um transponder passivo (SANGREMAN, 2003).

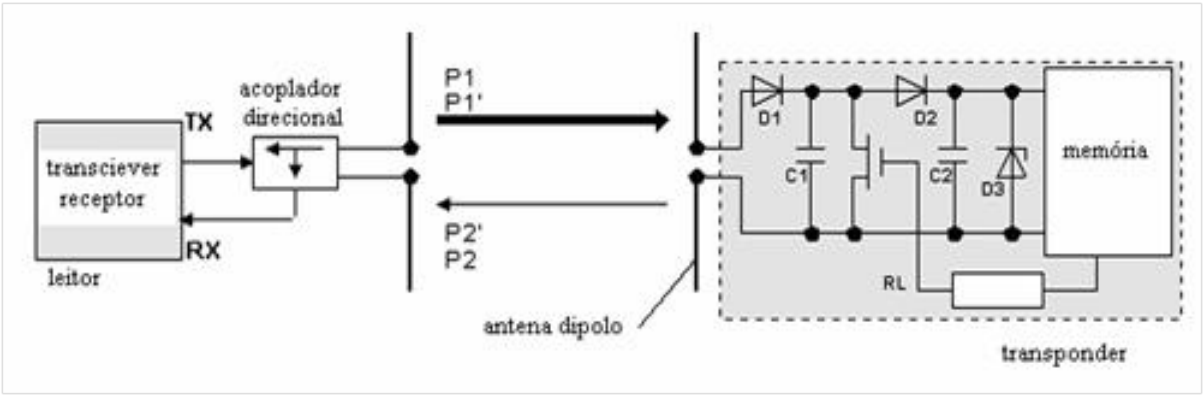

Figura 4 – Princípio de funcionamento do acoplamento *backscatter*(SANGREMAN, 2003)

2.4 Faixas de Frequências do sistema RFID

Como consequência da física dos campos eletromagnéticos, a faixa de frequências da tag determina também as características de atuação do sistema RFID. Com o objetivo de padronizar as tags, as normas técnicas disponíveis definem a construção de sistemas dentro de determinadas faixas de frequências específicas:

- **LF (*low frequency*):** de 30 kHz até 300 kHz. As *tags* desta faixa de frequências operam em 125 kHz ou 134,2 kHz. Geralmente, são *tags* passivas e seu maior uso é na identificação de animais e de pessoas, também na forma de implantes subcutâneos. A distância de leitura é de alguns centímetros;
- **HF (*high frequency*)**: de 3 MHz até 30 MHz. São *tags* construídas em 13,56 MHz, normalmente utilizadas como crachás para identificação individual ou então como meio de pagamento, por exemplo para o transporte público, tal como os bilhetes únicos. Também podem ser utilizadas para identificar objetos individuais, como nas lojas de departamento, em sistemas antifurto. A distância de leitura chega a ser maior do que 10 cm;

- **UHF (*ultra-high frequency*)**: de 300 MHz até 1 GHz. Nesta faixa, as tags são fabricadas nas faixas de frequências de 433 MHz, para uso em rastreamento de cargas, tais como contêineres, vagões, caminhões etc. Nessa faixa de frequências, as tags são ativas, ou seja, energizadas com baterias, e robustas. Outra faixa bastante utilizada é a de 868 MHz, na Europa, e de 915 MHz, nos Estados Unidos e Brasil. Essas *tags* também são empregadas em processos de rastreamento de ativos ou produtos em lojas, controle de estoque, inventários e identificação veicular, como por exemplo em pedágios eletrônicos;
- **Micro-ondas**: acima de 1 GHz. Duas frequências para RFID: 2,45 GHz e 5,8 GHz. Esta faixa de frequências é utilizada em aplicações industriais, científicas e médicas (ISM). No Brasil, a principal aplicação de transponders nessa faixa de frequências é a de identificação veicular para o pedágio eletrônico (PUHLMANN, 2015).

A Tabela 1 mostra o sumário das 3 principais faixas, suas características e aplicações típicas.

Banda de Freqüência	**Características**	**Aplicações Típicas**
Baixa: 100 a 500 KHz 1	Faixa de curta até média leitura Baixo custo Baixa velocidade de leitura	Controle de acesso Identificação de animal Controle de inventário
Média: 10 a 15 MHz (também denominada Alta)	Faixa de curta até média leitura Potencialmente de baixo custo Média velocidade de leitura	Controle de acesso Smart cards
Alta: 850 a 950 MHz e 2,4 a 5,8 GHz (também denominada Ultra Alta)	Faixa larga de leitura Alta velocidade de leitura Alto custo Linha de visão requerida	Monitoração de veículos em estradas

Tabela 1 – Características e aplicações típicas por frequência de operação (MARTINS, 2005)

A quantidade de energia transferida do transceptor para a tag é proporcional ao tamanho das antenas de transmissão e recepção, respectivamente. Já os sistemas na faixa de alta freqüência operam com o princípio de comunicação de antenas de radar. A taxa de transferência de dados é influenciada pela freqüência da faixa em uso entre a tag e o transceptor. Quanto maior a frequência, maior é a taxa de transferência de dados(MARTINS, 2005).

2.5 Vantagens da tecnologia RFID

A tecnologia de RFID não tem a pretensão de substituir o código de barras em todas as suas aplicações. Ela deve ser vista como um método adicional de identificação, a ser utilizado em aplicações onde o código de barras e outras tecnologias de identificação não atendam a todas as necessidades.

Cada tipo de identificação tem suas vantagens, e o necessário é saber aproveitar os melhores benefícios de cada tecnologia para montar uma solução ideal. Os benefícios primários de RFID são: a eliminação de erros de escrita e leitura de dados, coleção de dados de forma mais rápida e automática, redução de processamento de dados e maior segurança. Quanto às vantagens da RFID em relação às outras tecnologias de identificação e coleção de dados, temos: operação segura em ambientes severos (lugares úmidos, molhados, sujos, corrosivos, altas temperaturas, baixas temperaturas, vibração, choques), operação sem contato e sem necessidade de campo visual direto e grande variedade de formatos e tamanhos, redução de custos operacionais, eliminação de erros humanos, aumento da satisfação dos clientes, aumento na velocidade dos processos, melhor controle em processos de qualidade, redução de perdas e inventários(PUHLMANN, 2015).

2.6 Arquiteturas de Processadores Banda Base

Nesta seção, são descritos 3 diferentes tipos arquitetura para o bloco digital da tag de RFID.

2.6.1 1ª arquitetura

Esta arquitetura é baseada na ISO/IEC 15693 e a frequência de operação da tag é 13,56MHz. ISO/IEC 15693 é um padrão para cartão de proximidade que opera em uma distância de aproximadamente 1,5 m. O diagrama de bloco da parte digital é mostrado na Figura 5. Ele inclui os blocos: decodificador, codificador, gerador e verificador CRC, bloco de interface da memória EEPROM, bloco *master control* e o bloco de gerenciamento do clock.

O processo dos blocos da parte digital é descrito a seguir. O bloco de gerenciamento do *clock* faz a divisão da freqüência do clock para fornecer o valor adequado para a tag. O decodificador deve decodificar os dados que o bloco *front end* analógico recebeu a partir do pedido do leitor, em seguida, repassá-los ao bloco de verificação CRC e ao bloco de controle principal. Quando a verificação do CRC for concluída, deve-se dar um *feedback* dos resultados para o bloco *master control*. Este último, por sua vez, deve corresponder aos pedidos, como a prevenção de conflitos, ler e escrever no bloco da memória EEPROM, que deve retornar os dados para o bloco CRC e codificar os dados após o processamento

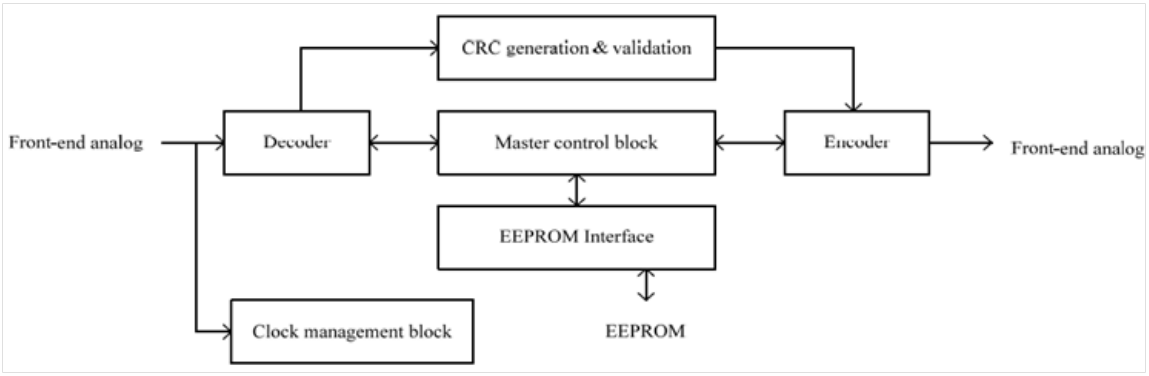

Figura 5 – Diagrama de blocos da parte digital(ZHANG WEIPING JING, 2014)

ser concluído. O bloco *CRC generation* deve receber os códigos CRC para o codificador. O codificador deve responder ao *front end* analógico, que, por sua vez, enviará os dados para o leitor (ZHANG WEIPING JING, 2014).

2.6.2 2ª arquitetura

Esta arquitetura utiliza as normas ISO 14443 para o projeto do bloco digital da tag. Para reduzir a complexidade, o tipo de tag é passiva e utiliza uma fonte de alimentação através de um acoplamento indutivo de um campo eletromagnético gerado pelo leitor. Para maior simplicidade, a tag projetada é de somente leitura e apenas mostra a sua identidade para o leitor. Lógica assíncrona é utilizada no bloco digital para minimizar a atividade do clock. A especificação tipo A da ISO 14443 é aplicada para o bloco digital da tag. Uma ROM de 128 bits foi utilizada para armazenar o código do ID da tag, que é obtido, processado e enviado de volta para o leitor através dos blocos do decodificador, multiplexador e modulador. Esta arquitetura é mostrada na Figura 6.

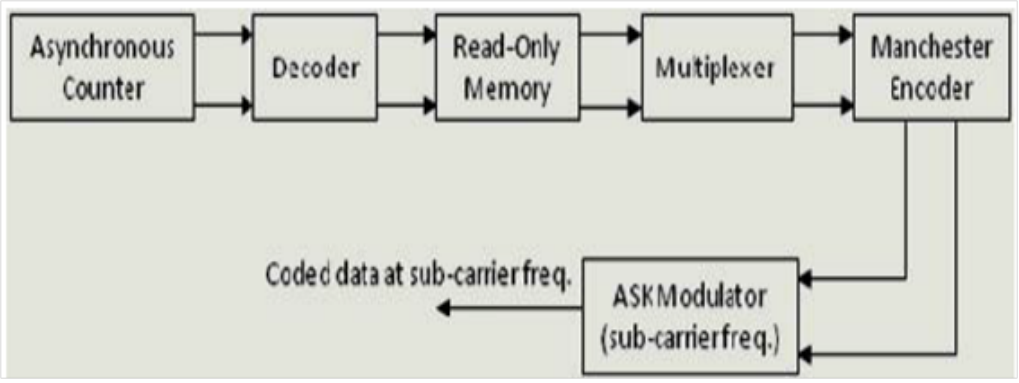

Figura 6 – Arquitetura interna do bloco digital(BARAL1, 2012)

O bloco digital contém os módulos para armazenamento, busca e processamento de dados. Lógica síncrona tem sido a escolha de muitos anos para projetos digitais devido à sua simplicidade e confiabilidade para um sistema complicado. Mas, para um sistema

autônomo como a tag, onde o consumo de energia é extremamente importante, o projeto da tag assíncrona é a escolha mais adequada. Como a maior parte da fonte de alimentação é consumida pelo *front end* analógico, o bloco digital deve realizar a sua operação com uma quantidade de energia em torno de 20 μW. A técnica assíncrona também favorece o projeto dos blocos da tag, visto que cada um dos módulos pode ser separadamente projetado, e que os módulos inteiros podem ser montados posteriormente (BARAL1, 2012).

2.6.3 3ª arquitetura

A freqüência de operação para diferentes aplicações do RFID muda de acordo com as restrições associadas à aplicação da tag. Alguns blocos da tag podem ser projetados de maneira que a mesma funciona para diversas frequências. Na Figura 7, é mostrado um diagrama de blocos básico para uma tag RFID com aplicação na área de biomédica.

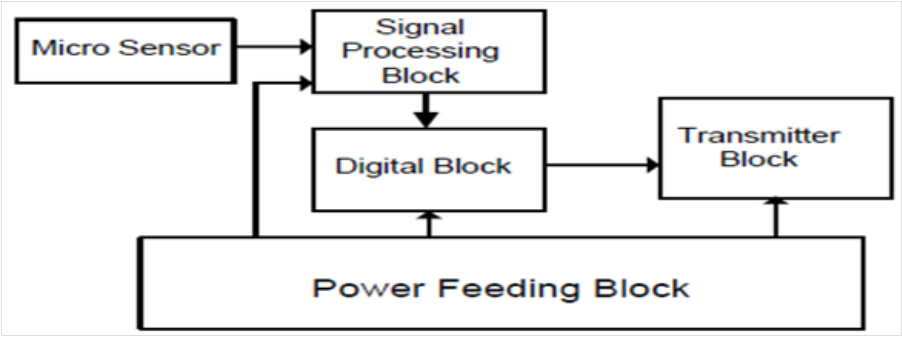

Figura 7 – Diagrama de blocos de uma tag RFID com aplicação biomédica(KOTHAMASU, 2012)

Cada etiqueta tem a sua própria identidade e as tags são identificadas pelo código armazenado no mesmo. O código de identificação é armazenado em uma memória ROM que está presente no bloco digital. O ID da tag é transmitido com objetivo da mesma ser identificada pelo leitor. A identificação, juntamente com os sinais de batimento do coração, são enviados para o leitor pelo bloco de transmissão, e a transferência de dados também é controlada neste mesmo bloco. O bloco digital pode ser utilizado para as três frequências RFID frequentemente utilizadas, que são 13,56 MHz, 915 MHz e 2,45 GHz. A única variação poderia ser na taxa de dados e no *master clock*.

As principais tarefas para a parte digital da tag são armazenar o ID da tag e transmitir o ID quando a tag estiver ligada. Além disso, ela também deve enviar o sinal de batimento cardíaco que foi processado pelo bloco de processamento de sinais. O diagrama de blocos da parte digital da tag é mostrado na Figura 8.

O bloco de aplicação contém uma memória ROM de 32 bits para armazenar o código de identificação da tag. O bloco de aplicação contém também um controlador para

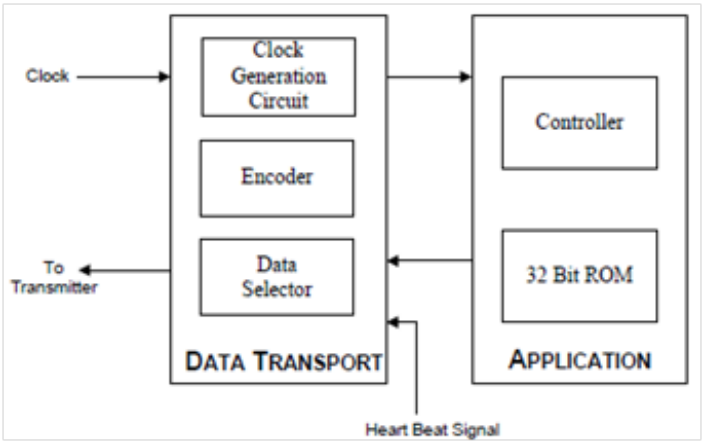

Figura 8 – Diagrama de blocos do bloco digital(KOTHAMASU, 2012)

controlar o fluxo do código ID da tag para o bloco de transporte de dados do bloco digital. A máquina de estado do controlador é mostrada na Figura 9.

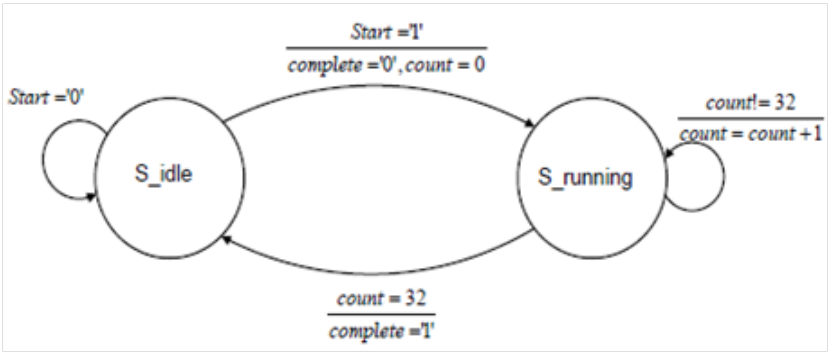

Figura 9 – Diagrama de estados do controlador do bloco de aplicação(KOTHAMASU, 2012)

Quando a tag não é ligada, ela permanece no estado ocioso. Quando a energia suficiente é induzida no tag, ela entra em estado de execução. Neste estado, para cada clock, os dados armazenados na memória ROM são recuperados e enviados para o bloco de transporte de dados. Uma vez que os dados foram enviados totalmente, a tag entra em estado ocioso. O sinal de clock necessário para realizar esta operação é gerado no bloco de transporte de dados(KOTHAMASU, 2012).

2.7 Protocolos de Comunicação

A finalidade da padronização e das normas é definir as plataformas em que uma indústria possa operar de forma eficiente e segura. Os maiores fabricantes de RFID oferecem sistemas proprietários, o que resulta numa diversidade de protocolos de sistemas de RFID numa mesma planta industrial. Na luta pela padronização de protocolos, existem

muitas organizações envolvidas nos projetos de tecnologias RFID. As mais conhecidas na área dos sistemas RFID são a ISO (*International for Standardization*) e a EPC Global. A Tabela 2 apresenta a relação de padrões publicados pela ISO (MARTINS, 2005).

ISO Standard	**Descrição**
ISO 11784	RFID para animais – estrutura de código
ISO 11785	RFID para animais – concepção técnica
ISO/IEC 14443	Identificação de cartões – cartões com circuitos integrados sem contato – cartões de proximidade
ISO/IEC 15693	Identificação de cartões – cartões com circuitos integrados sem contato – cartões de vizinhança
ISO/IEC 18001	Tecnologia da Informação – Gerenciamento de Itens de RFID – Perfil de Requisitos de Aplicação
ISO/IEC 18000-1	Parâmetros Gerais para Comunicação por Interface por Ar para Freqüências Globalmente Aceitas
ISO/IEC 18000-2	Parâmetros para Comunicação por Interface por Ar abaixo de 135 KHz
ISO/IEC 18000-3	Parâmetros para Comunicação por Interface por Ar em 13,56 MHz
ISO/IEC 18000-4	Parâmetros para Comunicação por Interface por Ar em 2,45 GHz
ISO/IEC 18000-6	Parâmetros para Comunicação por Interface por Ar em 860 a 930 MHz
ISO/IEC 15961	Gerenciamento de Itens de RFID – Protocolo de Dados: Interface de Aplicação
ISO/IEC 15962	Gerenciamento de Itens de RFID – Protocolo: Regras de Codificação de Dados e Funções de Memória Lógica
ISO/IEC 15963	Gerenciamento de Itens de RFID – Identificação única do RF Tag

Tabela 2 – Descrições dos Padrões de ISOs para RFID (MARKHAM, 2012)

2.7.1 Protocolo ISO/IEC 14443

As normas ISO/IEC 14443 são um padrão internacional para *Smart Cards* sem contato operando a 13,56 MHz em estreita proximidade com uma antena do leitor. Cartões de Proximidade de Circuito Integrado (PICC), cartões ou tags, destinam-se a operar dentro de aproximadamente 10 centímetros da antena do leitor. Essas normas são divididas em quatro partes:

1. **Parte 1 [ISO / IEC 14443-1: 2000]**: Define o tamanho e as características físicas do cartão. Ela também lista diversas condições ambientais que o cartão deve ser capaz de suportar, sem danos permanentes para a funcionalidade. Estes testes destinam-se a ser realizados ao nível do cartão e são dependentes da construção do

cartão e do projeto da antena. A maioria dos requisitos não pode ser facilmente convertida para o nível de matriz. A gama de temperaturas de funcionamento do cartão é especificada na parte 1 como um intervalo de temperatura ambiente de 0ºC a 50ºC.

2. **Parte 2 [ISO / IEC 14443-2: 2001]**: Define a potência de RF e a interface do sinal. Dois esquemas de sinalização, tipos A e B, são definidos na parte 2. Ambos os esquemas de comunicação são *half-duplex* com taxa de dados de 106 kbps em cada direção.*Half-duplex* significa que o fluxo de dados é unidirecional. Portanto, o envio e o recebimento de dados não são executados ao mesmo tempo. Os dados transmitidos pelo cartão de carga são modulados com uma subportadora de 847,5 kHz. O cartão é alimentado pelo campo de RF e não é necessária nenhuma bateria.

3. **Parte 3 [ISO / IEC 14443-3: 2001]**: Define a inicialização e anticolisão dos protocolos de tipo A e tipo B. Os comandos da anticolisão, respostas, frame de dados, e o tempo são definidos na parte 3. O regime de inicialização e anticolisão é projetado para permitir a construção de leitores multiprotocolo com capacidade de comunicação com ambas as placas do tipo A e tipo B. Ambos os tipos de cartões esperam silenciosamente no campo por um comando *polling*. Um leitor multiprotocolo comunica com um tipo de cartão, completa qualquer transação com o mesmo e, em seguida, comunica com outro tipo de cartão e realiza transações com ele.

4. **Parte 4 [ISO / IEC 14443-4: 2001]**: Define os protocolos de alto nível de transmissão de dados para o tipo A e tipo B. Os protocolos descritos na parte 4 são elementos opcionais da ISO/IEC 14443. Cartões de proximidade podem ser concebidos com ou sem apoio para protocolos da parte 4. A tag informa ao leitor se ela suporta comandos da parte 3 na resposta dos comandos *polling* (conforme definido na parte 3).

As principais diferenças entre o tipo A e o tipo B são os métodos de modulação, esquemas de codificação (parte 2) e procedimentos de inicialização do protocolo (parte 3). Ambos os cartões do tipo A e tipo B usam o mesmo protocolo de transmissão descrito na parte 4 (ISO/IEC, 2005).

2.7.1.1 Cartao Tipo A

A sinalização tipo A utiliza 100% de modulação de amplitude do campo de RF para comunicação do leitor para a tag, com os dados codificados utilizando uma modificação de *Miller* (figura 10). A comunicação do cartão para o leitor utiliza modulação OOK na frequência da subportadora de 847,5 kHz, com uma codificação *Manchester* nos dados. Na sinalização do tipo A, o campo RF é desligado por curtos períodos de tempo

quando o leitor está transmitindo. O circuito integrado deve armazenar energia suficiente em capacitores internos para continuar funcionando, enquanto o campo de RF é momentaneamente desligado durante a modulação campo.

Figura 10 – Modulação entre o leitor e a tag para o cartão tipo A(ISO/IEC, 2005)

2.7.1.2 Cartao Tipo B

O tipo de sinalização B utiliza 10% da amplitude da modulação ASK do campo de RF para comunicação do leitor para a tag, com os dados codificados em NRZ (11). Comunicação da tag para o leitor utiliza modulação BPSK de uma subportadora 847,5 kHz, com dados codificados em NRZ-L. O campo de RF é continuamente ligado para comunicações do tipo B.

O bloco *front end* analógico construído contém as mesmas especificações de modulação apresentadas no cartão tipo B, ou seja, modulação ASK do leitor para a tag e modulação BPSK, de uma subportadora 847,5 kHz, da tag para o leitor.

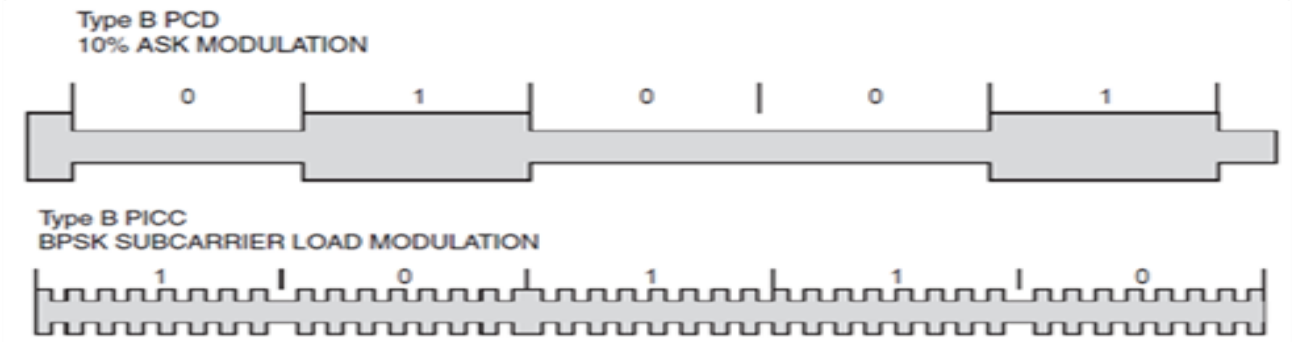

Figura 11 – Modulação entre o leitor e a tag para o cartão tipo B(ISO/IEC, 2005)

A codificação NRZ, que significa *No Return to Zero*, é o primeiro sistema de codificação, pois é mais simples. Ela consiste simplesmente em transformar o 0 em - X e o 1 em +X. Desta maneira, tem-se uma codificação bipolar na qual o sinal nunca é nulo. Por conseguinte, o receptor pode determinar a presença ou não de um sinal.

Dois tipos de código NRZ serão considerados: o NRZ–Level (NRZ–L) e o NRZ–Invert-on-one (NRZ–I), mostrados na Figura 12. Para o NRZ–L, é estritamente o nível de tensão que determina os valores de dados, sendo um nível alto para o bit “1” e um nível baixo para o bit “0”. No NRZ–I, a transição de tensão no início do pulso determina o valor associado de dado. Por exemplo, a transição no início do pulso pode ser usada para representar o binário “1”, e a falta de transição, o “0”.

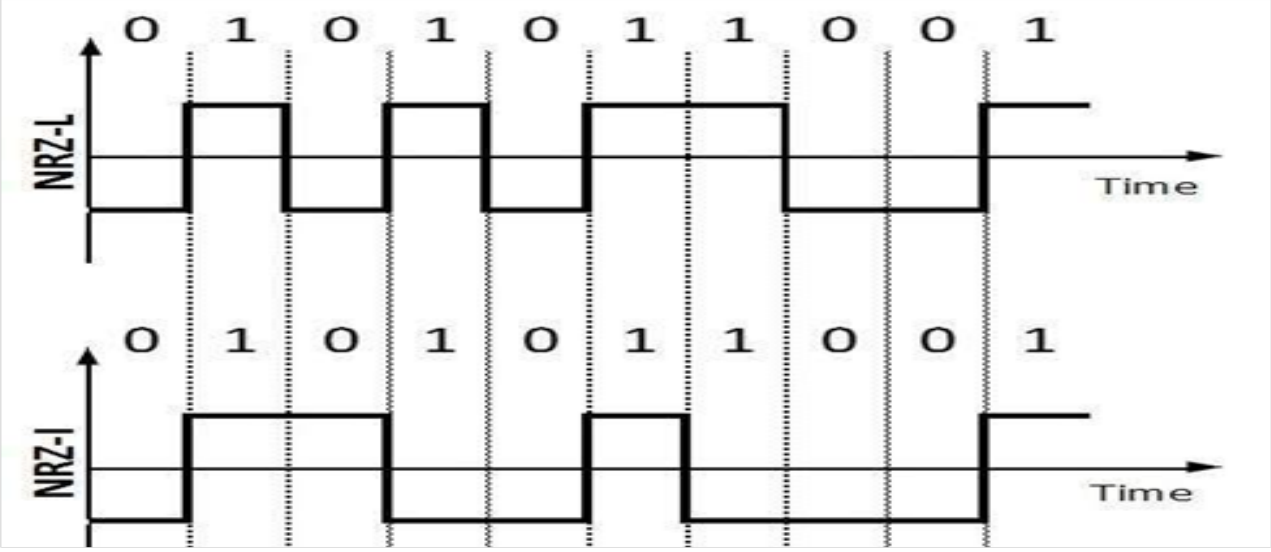

Figura 12 – Codificação NRZ-L e NRZ-I(TUTORIALSPOINT.COM, 2015)

O tempo de subida e de descida do envelope de modulação deve ser de 2 microssegundos ou menos, como mostrado na Seção 9.1.2 da ISO/IEC 14443-2. O *overshoot* e o *undershoot* não podem exceder [0,1(A-B)] em amplitude. *Overshoot* ocorre quando os valores transitórios excedem o valor final. Quando eles são menores do que o valor final, o fenómeno é chamado de *undershoot*.

2.8 Linguagem Verilog

Verilog é uma linguagem de descrição de *hardware* (HDL), cujo objetivo é especificar a descrição do comportamento de um circuito que eventualmente será implementado em *hardware*. É uma linguagem padrão IEEE (*Institute of Electrical and Electronic Engineers*), que tem uma grande semelhança com a linguagem C. Ela é altamente utilizada tanto para simulação quanto para síntese e tem se tornado uma das principais linguagens utilizadas por projetistas de *hardware*, tanto na indústria como no meio acadêmico, por suportar o projeto, verificação e implementação de projetos analógicos e digitais em vários níveis de abstração. Além disso, o Verilog é de fácil aprendizado, principalmente no ambiente acadêmico para projetos de circuitos e sistemas digitais.

O Verilog permite representar projetos complexos através de estruturas que constroem a hierarquia, que são os módulos e as portas. Módulo é o bloco básico de construção de um modelo. Portas são a interface do módulo. A estrutura de um módulo em Verilog é mostrada abaixo:

```
module nome_do_modulo (lista das portas);
   declaração das portas;
   ...
   declaração das variáveis;
   ...
   descrição do comportamento;
endmodule.
```

A definição do módulo é terminada como a palavra **endmodule**. A descrição do comportamento do módulo pode ser de diversas formas: estrutural, fluxo de dados, comportamental e RTL (*Register Transfer Level*). Essas diversas formas são descritas resumidamente neste item.

Estrutural: Representa circuitos lógicos usando primitivas de linguagem Verilog. Exemplo:

```
and c1(Carry, A, B);
```

"**c1**"é o nome da porta lógica AND. O primeiro parâmetro é uma variável *output*, no caso "**Carry**". Os outros parâmetros recebem variáveis *inputs*, no caso "**A**"e "**B**".

Fluxo de dados: Representa sinais de saída em função de sinais de entrada. Exemplo:

```
module and2 (a, b, y);
   input a, b;
   output y;
   assign y = a & b;
endmodule
```

"**assign**"é usado para modelar circuitos combinacionais, onde as saídas mudam quando uma das entradas muda. Ele é executado continuamente e não possui lista de sensibilidade. Neste caso, "**y**"é o resultado da operação lógica AND entre as portas "**a**"e "**b**".

Comportamental: Representa o comportamento na forma de um algoritmo. Exemplo:

```
if (select == 0)
   begin
    out = a;
   end
else if (select == 1)
   begin
    out = b;
   end
```

RTL: Também conhecido como nível de transferência de registradores, define basicamente relações entre entradas e saídas em termos de fluxo de operações dentro do modelo de *hardware*. Exemplo:

```
always @(posedge clock)
 begin
    pisca = ~pisca;
 end
```

"**Always**"delimita o bloco de código que será executado sempre que um certo sinal for ativado. Possui uma lista de sensibilidade para determinar quais sinais ativarão a execução do código. No exemplo acima, o código será executado em cada borda de subida do clock. Para o bloco de código ser executado apenas uma vez, no início da simulação, utiliza-se "**initial**".

3 Especificação do Protocolo de Comunicação entre o Leitor e a Tag

As normas ISO/IEC 14443, 15693, 15961 e 15962 operam na freqüência de 13,56 MHz. Cada norma possui um conjunto de parâmetros de projeto para interface de comunicação. As normas ISO/IEC 15693 utilizam modulação ASK com índice de 10% ou 100% da amplitude para comunicação do leitor para a tag. A comunicação da tag para o leitor utiliza modulação ASK com índice 100% de uma subportadora 423,75kHz com codificação *Manchester* de dados.

O bloco *front end* analógico implementado nos trabalhos anteriores contém as mesmas especificações de modulação apresentada no cartão tipo B da ISO/IEC 14443, ou seja, modulação ASK do leitor para a tag e modulação BPSK, de uma subportadora 847,5 kHz, da tag para o leitor. Portanto, será utilizado o protocolo ISO/IEC 14443 para a comunicação entre a tag e o leitor.

3.1 Formato de dados

A comunicação de dados entre a tag e o leitor é realizada utilizando um formato de dados *LSB-first*. Cada byte de dados é transmitido com um bit de início "0"e um bit de parada "1", conforme mostrado na Figura 13. O bit de parada, bit de partida, e cada bit de dados são definidos por uma unidade de tempo elementar (ETU) que possui comprimento de 9.439 μS. O ISO/IEC 14443 define um caractere que consiste de um bit de início, oito bits de dados, e um bit de parada.

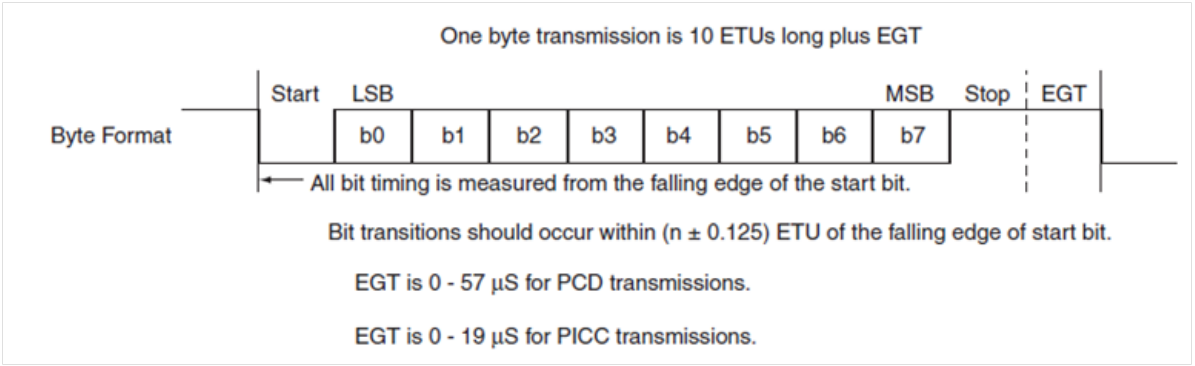

Figura 13 – Formato de um byte de dados(ISO/IEC, 2005)

Cada caractere pode ser separado do seguinte por um *extra guard time* (EGT). O EGT pode ser zero ou uma fração de uma ETU. EGT não pode exceder 19 μS aos

dados transmitidos pela tag, ou 57 microssegundos para dados transmitidos pelo leitor. A posição de cada bit é medida em relação à borda de descida do bit de início.

Apesar do fato de as transmissões de dados ocorrerem como *LSB-first*, todos os comandos e de dados da norma ISO/IEC 14443 estão listados na maneira convencional, com o *MSB* do lado esquerdo e com o *LSB* do lado direito (ISO/IEC, 2005).

3.1.1 Formato dos frames

Os dados transmitidos pelo leitor ou pela tag são enviados como frames. O frame padrão consiste no início do frame (SOF), vários caracteres, e o fim do frame(EOF). Os requisitos SOF e EOF são ilustrados na Figura 14 (ISO/IEC, 2005).

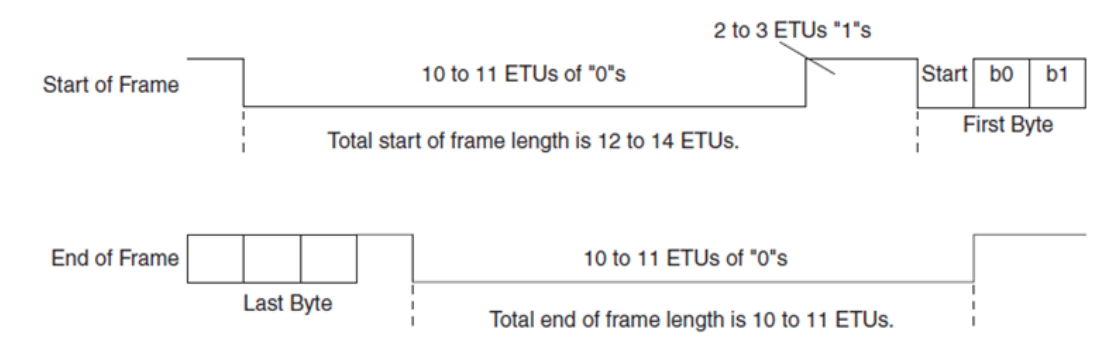

Figura 14 – Requerimentos do SOF e EOF(ISO/IEC, 2005).

3.1.2 Transmissão de Dados do leitor

O sinal da portadora não modulada que é transmitido quando o leitor está inativo é definido como lógica "1", enquanto o nível do sinal modulado é definido como lógica "0". Um frame transmitido pelo leitor consiste em SOF, vários caracteres de dados seguidos por dois bytes CRC_B (Verificador Cíclico de Redundância para o tipo B), e pelo EOF (Figura 15) (ISO/IEC, 2005).

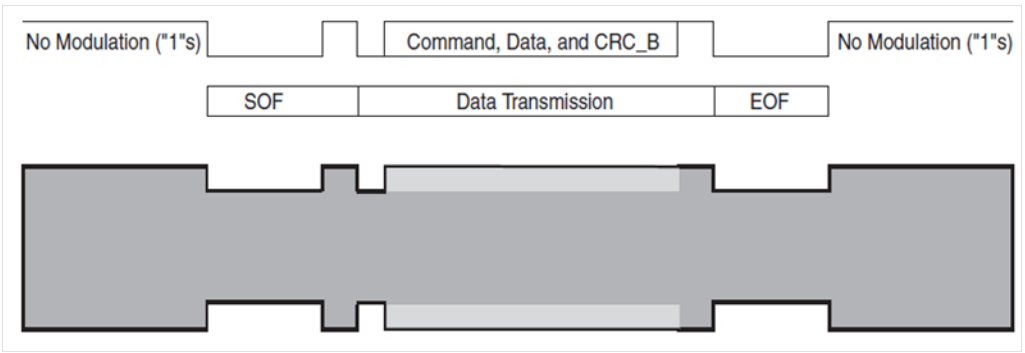

Figura 15 – Frame de comunicação do leitor(ISO/IEC, 2005)

3.1.3 Transmissão de dados da tag

A parte 2 da norma ISO/IEC 14443 especifica que a tag aguarde em silêncio por um comando do leitor depois de ser ativado pelo campo RF. Depois de receber um comando válido do leitor, a tag ligará a subportadora somente se ela tiver a intenção de transmitir uma resposta. A resposta da tag consiste em TR1(*Synchronization Time*), SOF, vários caracteres de dados seguido por um CRC_B de dois bytes, e pelo EOF. A subportadora deve ser desligada o mais tardar em 2 ETUs após a EOF (Figura 16).

A subportadora é ligada e mantém-se não modulada por um período de tempo conhecido como o tempo de sincronização(TR1). A fase da subportadora durante TR1 define lógica "1"e permite que o demodulador do leitor trave o sinal da subportadora. A subportadora deve permanecer ligada até depois da transmissão EOF esteja completa (ISO/IEC, 2005).

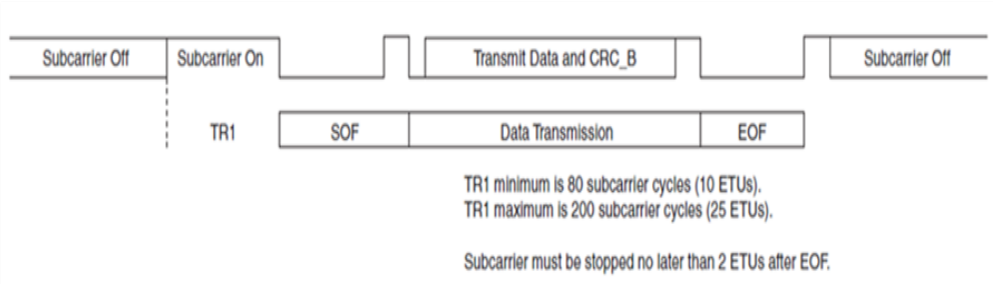

Figura 16 – Frame de comunicação da tag(ISO/IEC, 2005)

3.1.4 Tempo de resposta

Após a tag receber um comando a partir do leitor, não é permitido transmitir uma subportadora durante o tempo de guarda (TR0), conforme mostra a Figura 17. O tempo mínimo de tempo de espera é de oito ETUs para todos os comandos de resposta. O tempo máximo de guarda é definido pelo tempo de espera do frame (FWT), exceto para a resposta ATQB (*Answer to Request, Type B*), que tem um máximo TR0 de 32 ETUs (Figura 19).

Nas comunicações eletrônicas, *polling* é a verificação contínua de outros programas ou dispositivos por um programa ou dispositivo para ver o estado em que estão, geralmente para ver se eles ainda estão ligados ou querem se comunicar. Um dispositivo de controle com múltiplos dispositivos conectados compartilham a mesma linha, e o dispositivo de controle envia uma mensagem para cada dispositivo, um de cada vez, verificando se cada um tem alguma coisa a comunicar. Em outras palavras, se o dispositivo quer usar a linha de comunicação.

O FWT é o tempo máximo que uma tag exige para começar uma resposta. A tag transmite um parâmetro na resposta ATQB ao comando *polling*, que diz ao leitor o pior

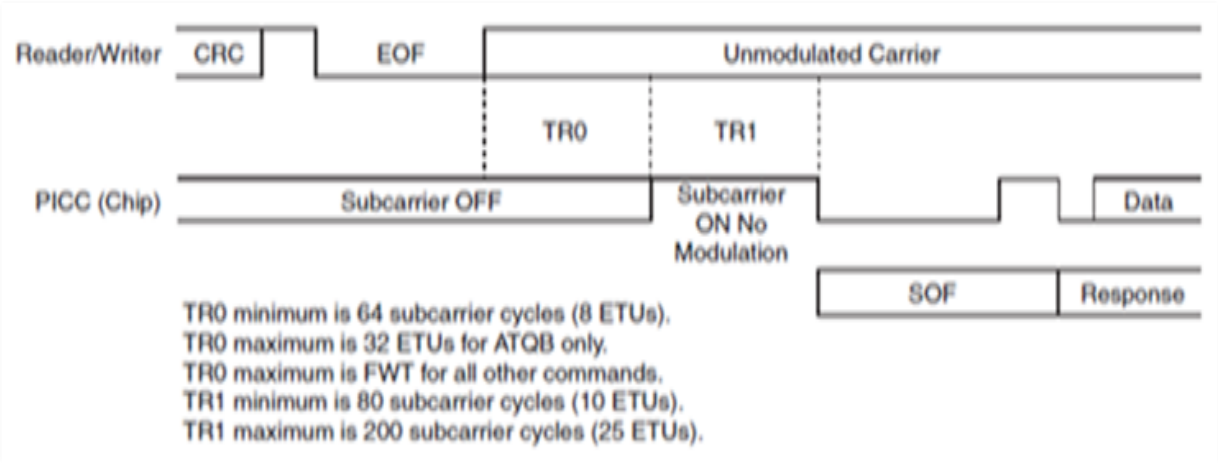

Figura 17 – tempo de guarda TR0(ISO/IEC, 2005).

caso FWT. Após a resposta da tag, o leitor é obrigado a esperar o tempo de atraso do frame (TR2) antes da transmissão do próximo comando. O tempo mínimo de atraso do frame necessário para todos os comandos são 14 ETUs, como é mostrado na Figura 18 (ISO/IEC, 2005).

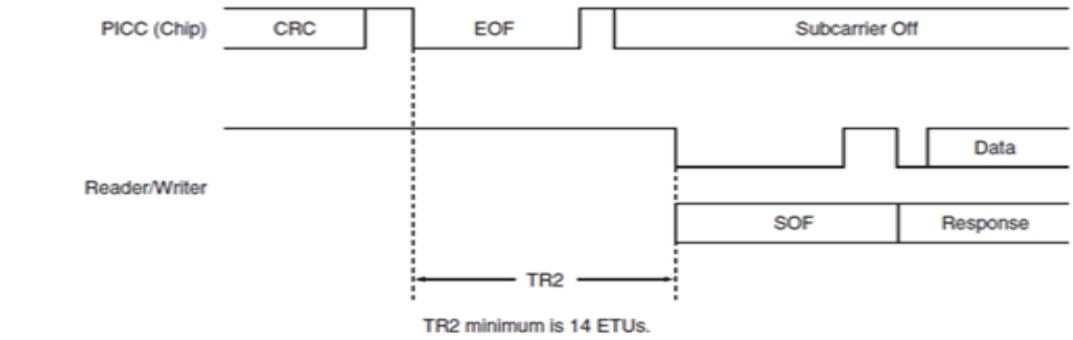

Figura 18 – Tempo de atraso do frame TR2(ISO/IEC, 2005).

3.1.5 Detecção de erros CRC

O CRC_B de 2 bytes é necessário em cada frame transmitido pela tag ou pelo leitor para permitir a detecção de erros de transmissão. O CRC_B é calculado em todos os comandos e bytes de dados no frame. A SOF, EOF, bits de inicio, bits de parada e EGT não estão incluídos no cálculo do CRC_B. O CRC_B de 2 bytes segue os bytes de dados do frame (Figura 19).

No exemplo apresentado acima, o CRC_B é calculado sobre os "k"bytes de dados e, em seguida, anexado aos dados. CRC1 é o byte menos significativo e CRC2, o byte mais significativo do CRC_B. Cada byte de dados e de CRC é transmitido como *LSB-first* (ISO/IEC, 2005).

Figura 19 – Ordem do byte CRC_B(ISO/IEC, 2005).

O valor inicial do registro utilizado para o cálculo CRC_B é $FFFF. Em hardware, a codificação e decodificação CRC_B é realizada por registradores (flip-flops D), com deslocamento cíclico de 16 etapas e com portas de realimentação apropriadas (portas XOR), como mostra a Figura 20.

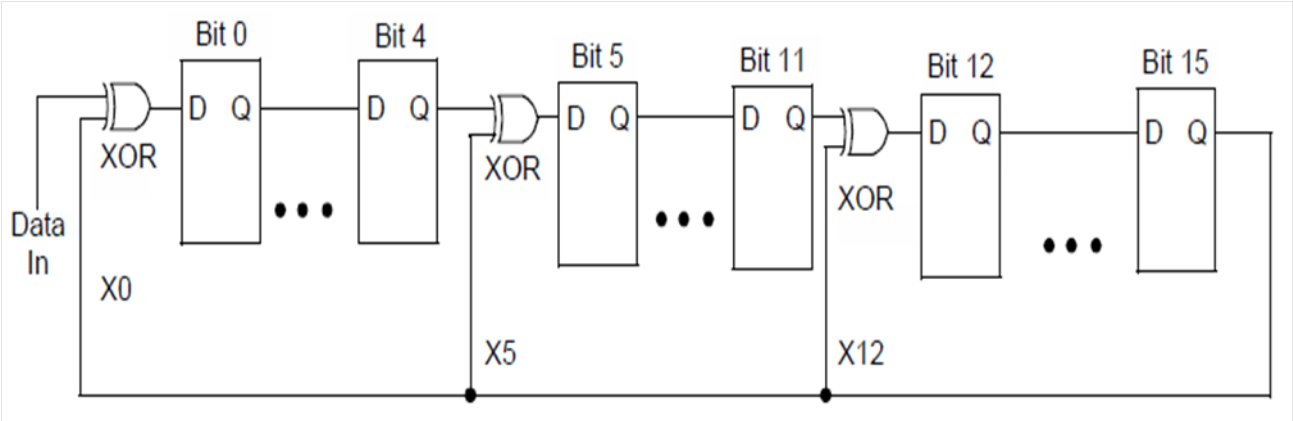

Figura 20 – Hardware do gerador CRC_B(BOMMENA, 2008).

Para a detecção de erros, uma soma de verificação é calculada e enviada ao final da mensagem que deve ser transmitida. Essa soma é binária de módulo 2, onde a soma é executada sem "carry", o que equivale a uma operação XOR (OU-Exclusivo). A Tabela 3 mostra a tabela-verdade da porta XOR de duas entradas.

A	**B**	**Y**
0	0	1
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 3 – Tabela Verdade porta XOR

O CRC_B representa o restante da divisão dos dados contidos no frame pelo polinômio $x^{16}+x^{12}+x^5+1$, que em binário é representado pelos bits **1 0001 0000 0010 0001**. Esta divisão é equivalente a uma soma binária de módulo 2. Portanto, os bits de dados são divididos por um número binário do polinômio. O resto da divisão é anexado na mensagem transmitida.

Um exemplo do cálculo para gerar um CRC é mostrado na Figura 21. A mensagem a ser enviada é igual a **110101**. O polinômio utilizado neste exemplo será $x^2 + x^0$, ou seja, **101** em binário. O resto desta divisão será o CRC.

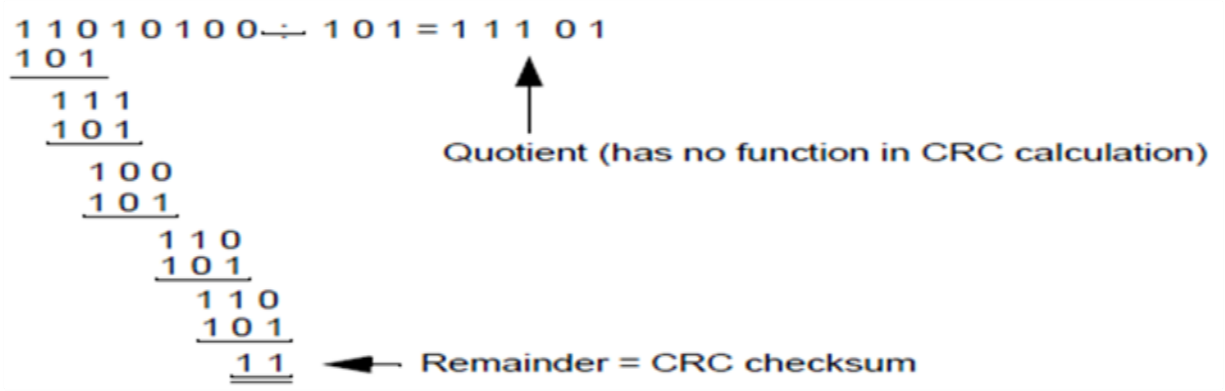

Figura 21 – Cálculo para gerar um CRC(SCHMIDT, 2000).

Portanto, a divisão é realizada bit a bit e então o divisor é deslocado um bit para a direita. Esse processo é repetido até o divisor chegar na extremidade direita do bit da mensagem. Para o receptor checar a mensagem por um erro CRC, deve-se dividir a mensagem junto com o CRC pelo mesmo polinômio que o transmissor usou, como mostra a Figura 22. Se o resultado da divisão é zero, então a transmissão foi realizada com sucesso. Se o resultado não for zero, um erro ocorreu durante a transmissão.

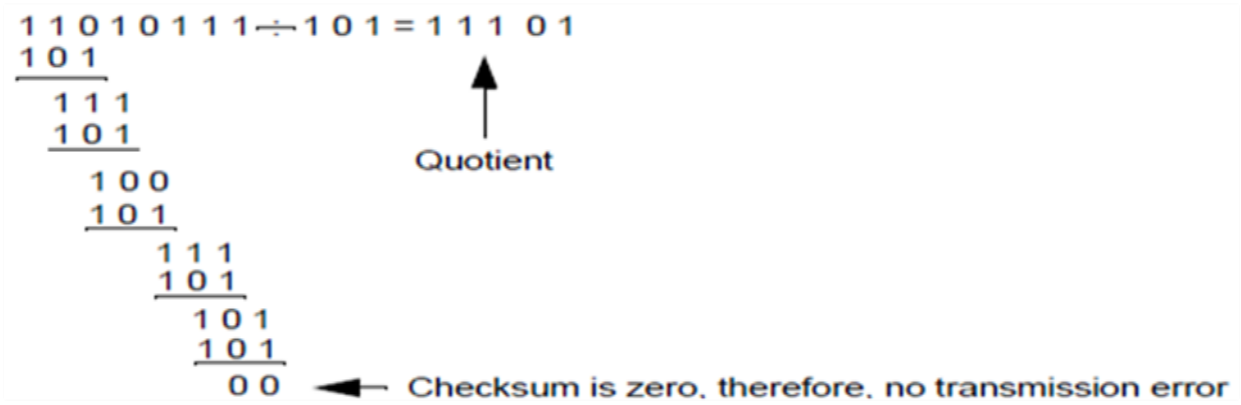

Figura 22 – Checando a mensagem por um erro CRC(SCHMIDT, 2000).

3.2 Anticolisão

Durante a sequência de anticolisão, pode acontecer que duas ou mais tags respondam simultaneamente. O leitor pode repetir o procedimento da anticolisão até encontrar

todas as tags próximas ao leitor. Após ter completado a sequência de anticolisão, a comunicação da tag estará sob o controle do leitor, permitindo que apenas uma tag de cada vez se comunique com o leitor. O esquema de anticolisão é baseado na definição de intervalos de tempo (*timeslots*) em que as tags são convidadas a responder com dados mínimos de identificação. As tags estão autorizadas a responder apenas uma vez a seqüência de anticolisão.

Conseqüentemente, mesmo no caso de várias tags presentes no campo da antena, haverá provavelmente um slot na qual apenas uma tag responde e onde o leitor é capaz de capturar os dados de identificação. Com base nos dados de identificação da antena, o leitor é capaz de estabelecer um canal de comunicação com a tag identificada. A ISO/IEC 14443-3 descreve dois tipos de anticolisão para a tag do tipo B: o procedimento *timeslot* e o procedimento probabilistico. A tag deste projeto será projetada utilizando o procedimento *timeslot*.

Quando a tag entra no campo RF do leitor, executa *Power on Reset* e espera silenciosamente por um comando *polling* tipo B válido. A tag deve ser capaz de aceitar o comando *polling* em 5 ms após ser ativada pelo campo. Se o leitor foi projetado para multiprotocolos, ou seja, para os tipos A e B, então a tag deve ser capaz de aceitar o comando *polling* em 5 ms depois do leitor parar a modulação do tipo A. Uma simples versão do diagrama do fluxo de estados da tag é mostrada na Figura 23. Este fluxograma será explicado detalhadamente no próximo tópico.

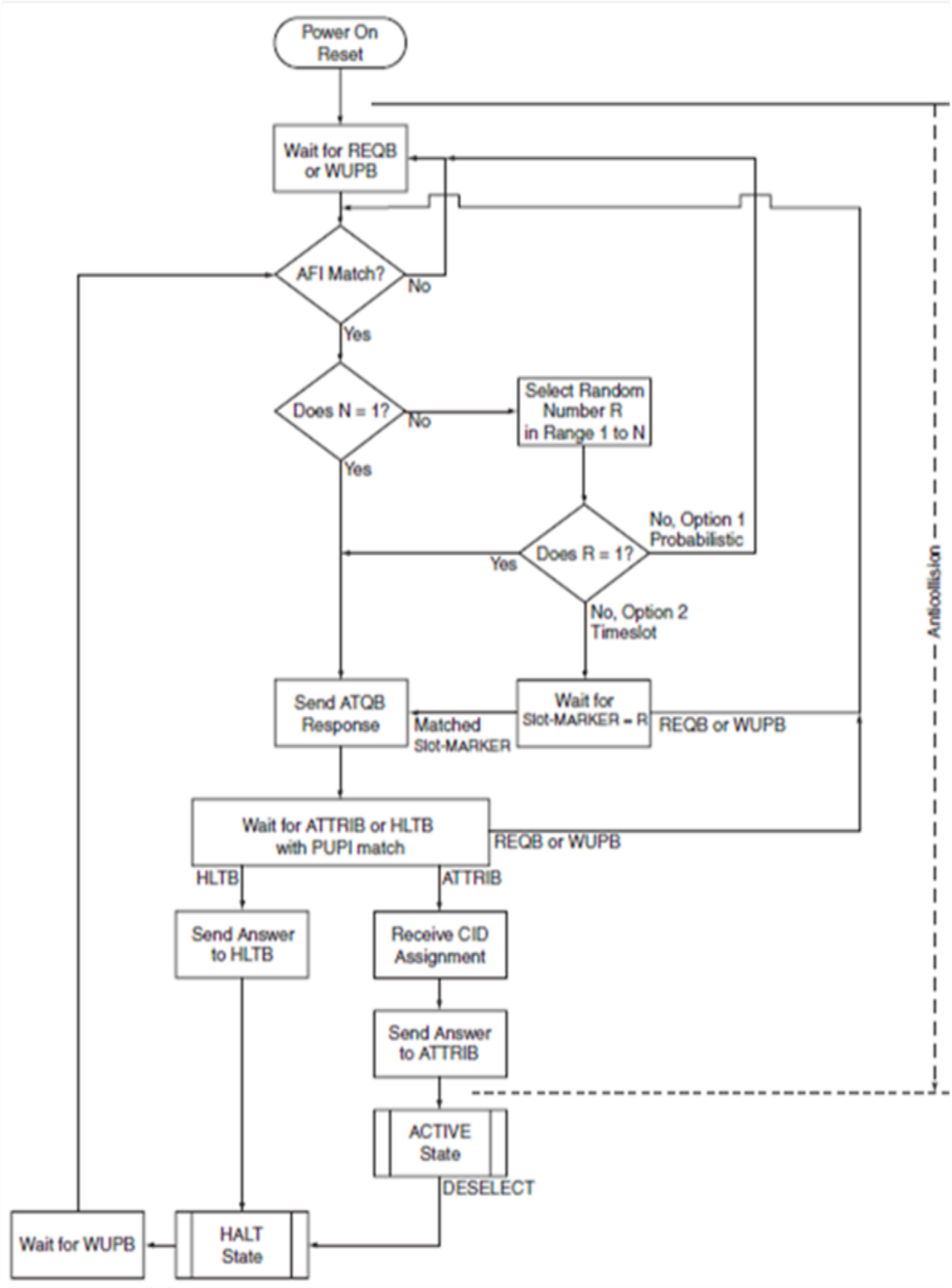

Figura 23 – Diagrama de transição dos estados da tag(ISO/IEC, 2005).

3.2.1 Anticolisão *Timeslot*

O leitor inicia o processo da anticolisão enviando um comando polling REQB (*Request Command, Type B*) ou WUPB (*Wake Up Command, Type B*). O comando WUPB ativa qualquer tag no campo com um código AFI (*Application Family Identifier*) correspondente. O comando REQB executa a mesma função, mas não afeta a tag no estado *Halt*. Os comandos REQB ou WUPB contém um inteiro "N"indicando o número de slots atribuídos ao processo de anticolisão para as tags.

Se "N"=1, então todas as tags respondem com ATQB (*Answer to Request, Type B*). Se "N"é maior que um, então a tag seleciona um número aleatório "R", em um alcance de 1 a "N". Se "R"=1, então a tag responde com ATQB. Se "R"é maior que 1, então a tag espera silenciosamente por um comando *Slot-Marker*, onde o número de slot "S"é igual a "R", e então responde com ATQB. O leitor verifica todos os slots periodicamente para determinar se alguma tag está presente no campo. A tag é permitida responder somente em um intervalo de "N"slots.

A resposta ATQB contém um número de identificação do cartão PUPI (*Pseudo Unique PICC Identifier*), que é usado para comandos diretos para uma tag específica durante o processo de anticolisão. Quando o leitor receber uma resposta ATQB, ele pode responder com um HLTB (*Halt Command, Type B*) para colocar a tag no estado *Halt*, ou pode responder com um comando ATTRIB (*PICC Selection Command, Type B*) para atribuir um número do ID do cartão (CID) e colocar a tag no estado ativo. Se a tag não suportar CID (*Card Identifier*), então o código CID $0 é enviado. Uma vez colocado no estado ativo, a tag está pronta para transações usando os comandos do estado ativo. A tag no estado ativo ignora os comandos REQB, WUPB, Slot MARKER, ATTRIB e HLTB.

A tag no estado ativo que suporta CID ignora comandos que não contêm o número CID que corresponde ao CID atribuido pelo comando ATTRIB. Até 15 tags que suportam CID pode ser ativadas simultaneamente. Se a tag não suporta a CID, então o leitor irá colocar uma única tag no estado ativo e completar a transação com o cartão antes de colocar no estado *Halt* e continuar com o procedimento de anticolisão.

Quando o leitor recebe uma resposta ATQB com um erro CRC, uma colisão é considerada como tendo ocorrido. Tipicamente, o leitor irá completar a transação com qualquer outra tag no campo e então colocá-la no estado *Halt*. O leitor então irá emitir um novo comando REQB, fazendo com que cada tag no campo que não esteja no estado Halt selecione um novo número aleatório "R". Esse procedimento resolve o conflito entre as tags colididas, permitindo que o leitor se comunique com elas.

O processo de anticolisão continua desta maneira até que todas as tags no campo tenham concluído as suas transações. Qualquer comando recebido pela a tag durante o processo de anticolisão com um erro CRC ou erro no formato do frame é ignorado.

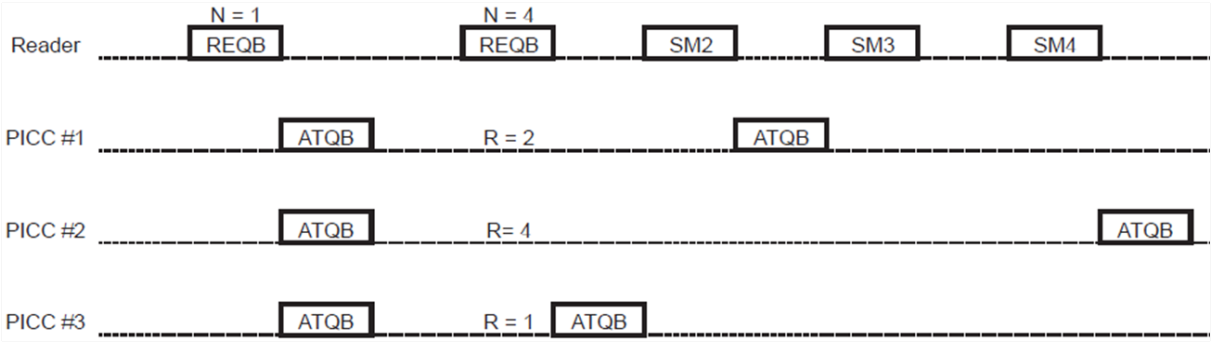

Figura 24 – Exemplo da anticolisão *Timeslot*(ISO/IEC, 2005).

Um exemplo da anticolisão *timeslot* é mostrada na Figura 24. Depois de transmitir REQB com "N"=1, todas as três tags no campo respondem, resultando em uma colisão. Enviando REQB com "N"=4 faz com que cada tag selecione "R"usando um gerador de número aleatório interno. A tag responde somente para o *Slot-Marker* correspondente ao "R". Note que o comando *Slot-Marker* pode ser transmitido pelo leitor em qualquer ordem. A parte 3 da norma define os comandos e respostas para a inicialização e anticolisão da tag do tipo B. A codificação do comando e resposta completa dos frames são mostrados a seguir.

3.2.2 Comando REQB/WUPB

Os comandos REQB e WUPB, como mostra a Figura 25, são usados como o primeiro passo do processo de anticolisão para tags do tipo B. Para responder aos comandos REQB ou WUPB, o comando ATQB é utilizado.

O AFI(*Application Family Identifier*) é usado para selecionar a família e subfamília de cartões que o leitor tem como alvo. Somente tags com código correspondente AFI podem responder os comandos REQB ou WUPB. A Tabela 4 descreve os critérios de correspondência AFI. Os valores de x e y da Tabela 4 podem variar entre $1 e $F.

AFI High Bits	**AFI Low Bits**	**REQB/WUPB Polling Produces a PICC Response From:**
$0	$0	All Families and Sub-Families
X	$0	All Sub-Families of Family X
X	Y	Only Sub-Family Y of Family X
$0	Y	Proprietary Sub-Family Y only

Tabela 4 – Critérios de correspondência AFI (ISO/IEC, 2005)

Utilizando os critérios de correspondência, o código AFI transmitido pelo leitor é comparado ao código AFI da tag. Por exemplo, se o registro AFI da tag contém $3B

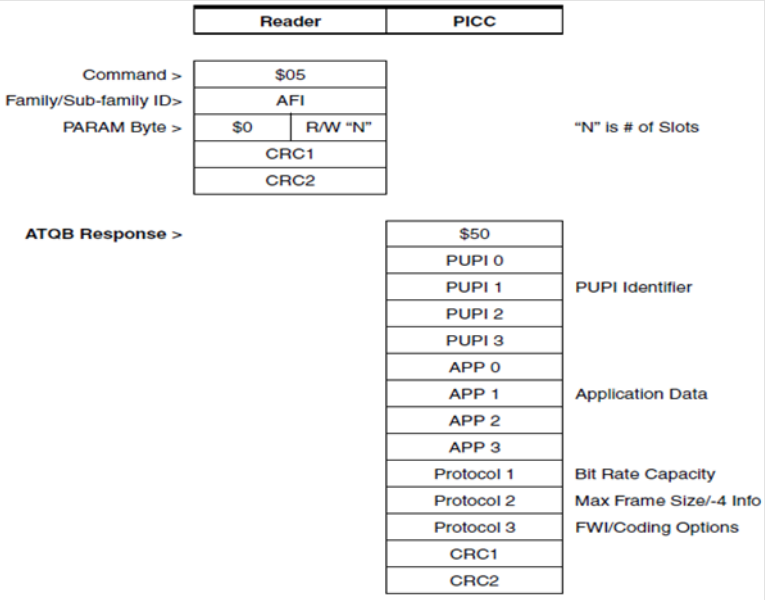

Figura 25 – Exemplo da anti-colisão *Timeslot*(ISO/IEC, 2005).

(familia 3, sub-familia B), então o AFI correspondente deve ocorrer somente se o leitor transmitir um AFI de $3B, ou $30 ou $00. Um AFI de $00 ativa todas as tags do tipo B. A definição das famílias do código AFI é mostrada na Tabela 5. O AFI da tag projetada será $3B, ou seja, com uma aplicação de identificação, um exemplo seria um controle de acesso que permite o acesso a uma sala apenas por uma pessoa autorizada.

AFI High Bits	**AFI Low Bits**	**Application Family**	**Examples**
$0	Y	Proprietary	
$1	Y	Transport	Mass Transit, Bus, Airline
$2	Y	Financial	Banking, Retail, Electronic Purse
$3	Y	Identification	Access Control
$4	Y	Telecommunication	Telephony, GSM
$5	Y	Medical	
$6	Y	Multimedia	Internet Services
$7	Y	Gaming	
$8	Y	Data Storage	Portable Files
$9-$F	Y	RFU	Not Currently Defined by 14443-3

Tabela 5 – Definição da família do código AFI(ISO/IEC, 2005)

Os comandos REQB e WUPB contêm um parâmetro "N", que atribui o número de slots disponíveis para o processo de anticolisão. A codificação de "N"é mostrada na Tabela 6. Valores de "N"que são reservados para uso futuro (RFU) são proibidos.

Bit 2	Bit 1	Bit 0	N
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	RFU
1	1	x	RFU

Tabela 6 – Codificação dos "N"Parâmentros da anti-colisão(ISO/IEC, 2005)

A selecão dos comandos REQB ou WUPB é determinada pelo valor do bit 3 do byte PARAM. Se o bit 3 for zero, representa o comando REQB. Se o bit 3 for 1, representa o comando WUPB. O REQB ativa a tag para os estados *Idle* ou *Ready*. O WUPB ativa a tag nos estados *Idle* ou *Ready* e acorda tags no estado *Halt*.

3.2.3 Comando Slot-Marker

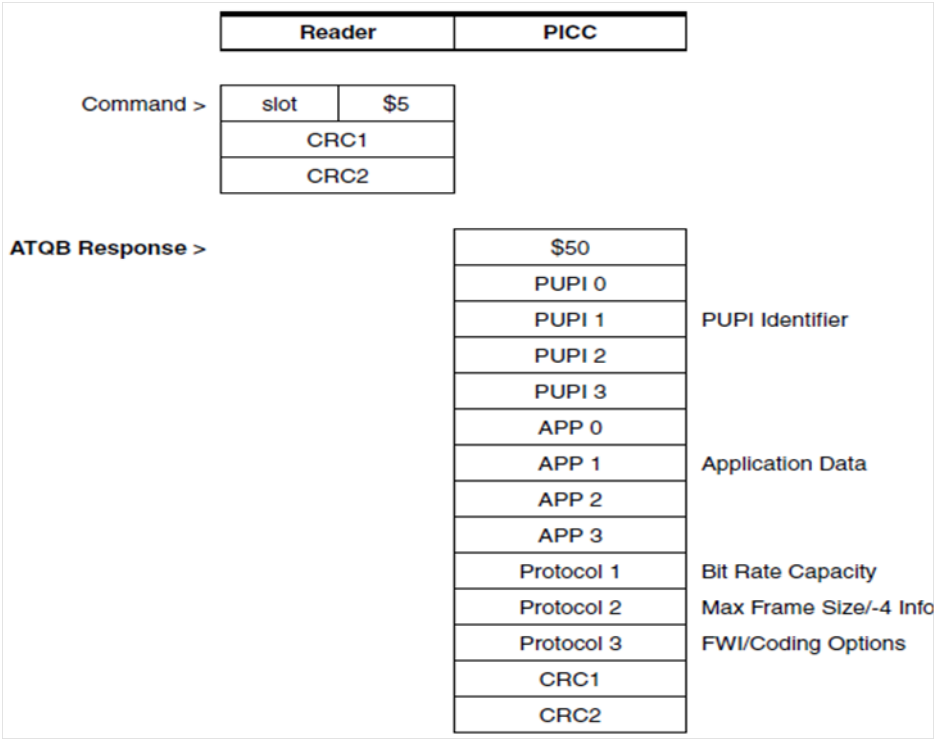

Figura 26 – Comando e Resposta Slot-Marker(ISO/IEC, 2005).

Depois dos comandos REQB ou WUPB com "N"maior que 1, a resposta ATQB é recebida e o leitor irá transmitir comandos *Slot-Marker* com valor do slot "S"de 2 até "N"para definir o começo de cada *timeslot* para a anticolisão. Se um número aleatório "R"selecionado pela tag corresponde a "S", então a tag responde com ATQB. Os comandos

Slot-MARKER, mostrados na Figura 26, não são obrigados a serem emitidos em uma ordem particular.

A codificação do número de slot do byte do comando é mostrada na Tabela 7.

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Slot**
0	0	0	0	Not Supported
0	0	0	1	2
0	0	1	0	3
0	0	1	1	4
0	1	0	0	5
0	1	0	1	6
0	1	1	0	7
0	1	1	1	8
1	0	0	0	9
1	0	0	1	10
1	0	1	0	11
1	0	1	1	12
1	1	0	0	13
1	1	0	1	14
1	1	1	0	15
1	1	1	1	16

Tabela 7 – Codificação do número do Slot(ISO/IEC, 2005)

3.2.4 Resposta ATQB

A resposta ATQB (*Answer to Request B*) serve para os comandos REQB, WUPB e *Slot-Marker*, e transmite o identificador PUPI e informações importantes do protocolo para o leitor. O formato da resposta é mostrado na Figura 27. Note que o PUPI (*Pseudo Unique PICC Identifier*) não é necessariamente um valor fixo. A tag pode gerar valores aleatórios PUPI.

Os três bytes do protocolo, mostrados na Figura 28, comunicam ao leitor se a tag suporta recursos de comunicação ou funcionalidades opcionais. O byte 1 do protocolo é $00 se a tag comunica apenas a taxa padrão de dados de 106 kbps em cada direção.

O byte 2 do protocolo contém o código de compatibilidade da parte 4 da norma e o tamanho máximo do frame suportado pela tag. Um valor de $0 nos quatro primeiros bits do Protocolo 2 indica que a tag não é compatível com a ISO/IEC 14443-4, enquanto o valor $1 indica a compatibilidade da parte 4. A codificação do tamanho máximo do frame da tag é mostrada na Tabela 8.

O byte 3 do protocolo contém o bit *Frame Waiting Time Integer* (FWI), no qual define o *Frame Waiting Time* (FWT), que é a máxima quantidade de tempo que o leitor

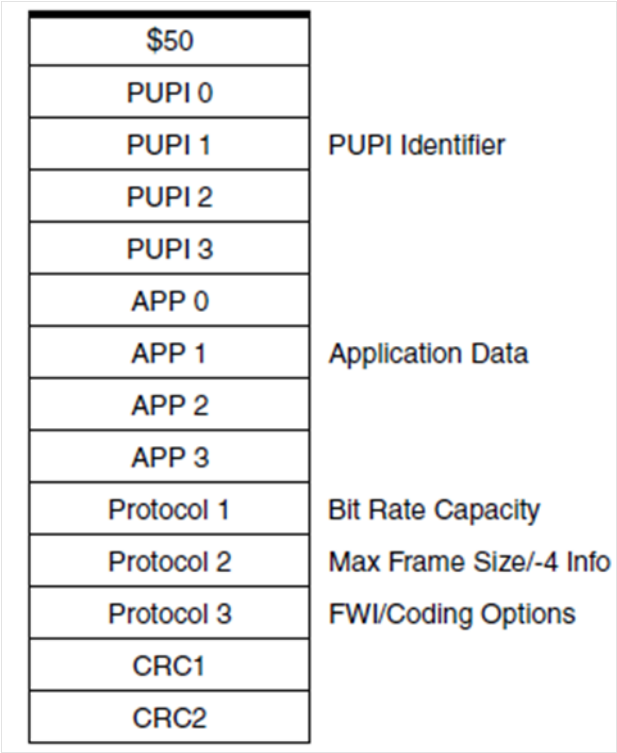

Figura 27 – Formato da resposta ATQB(ISO/IEC, 2005).

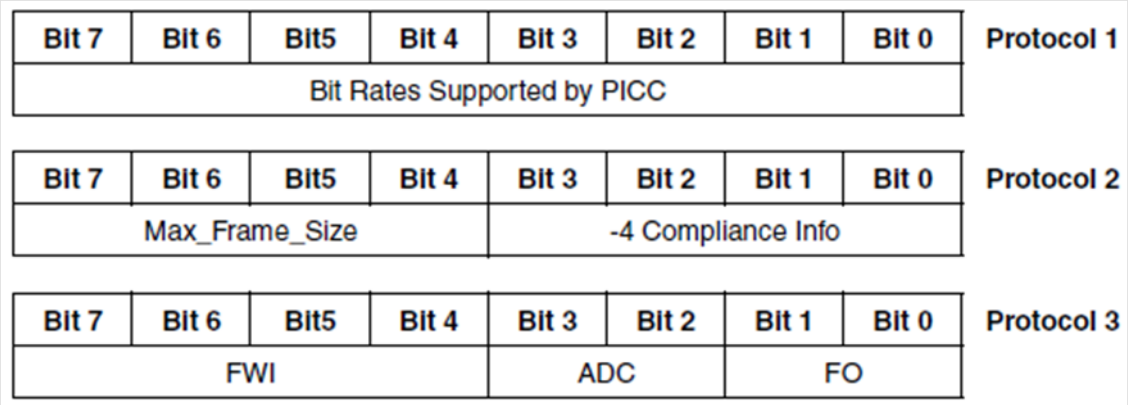

Figura 28 – Definição do byte do Procoloco ATQB(ISO/IEC, 2005).

deve esperar pela resposta da tag. A Tabela 9 mostra a codificação FWI e FWT em tempos de ETU e microssegundos, usando a fórmula da parte 3 da norma.

O *frame option* (FO) e o *Application Data Coding* (ADC) são também definidos no byte 3 do protocolo. Os bits do FO apresentam o suporte do CID ou NAD pela tag. O CID é usado para identificação de múltiplas tags no estado ativo. Se o bit 0 é 1, o CID é suportado pela tag. O NAD é usado para definir conexões lógicas para comunicações compatíveis com a parte 4. Se o bit 1 for igual a 1, o NAD é suportado pela tag. Se os bits ACD iniciarem uma aplicação proprietária, então os 4 bytes de dados de aplicação na resposta ATQB devem conter qualquer dado de aplicação.Se o bit 3 é 0 e o bit 2 é 0,

Bit 7	Bit 6	Bit 5	Bit 4	Max Frame
0	0	0	0	16 Bytes
0	0	0	1	24 Bytes
0	0	1	0	32 Bytes
0	0	1	1	40 Bytes
0	1	0	0	48 Bytes
0	1	0	1	64 Bytes
0	1	1	0	96 Bytes
0	1	1	1	128 Bytes
1	0	0	0	256 Bytes
1	x	x	x	RFU

Tabela 8 – Codificação do tamanho máximo do Frame da tag no byte 2 do Protocolo(ISO/IEC, 2005)

Bit 7	Bit 6	Bit 5	Bit 4	FWT	FWT Time
0	0	0	0	32 ETUs	302.1 μs
0	0	0	1	64 ETUs	604.1 μs
0	0	1	0	128 ETUs	1208.3 μs
0	0	1	1	256 ETUs	2416.5 μs
0	1	0	0	512 ETUs	4833.0 μs
0	1	0	1	1024 ETUs	9666.1 μs
0	1	1	0	2048 ETUs	19332.2 μs
0	1	1	1	4096 ETUs	38664.3 μs
1	0	0	0	8192 ETUs	77328.6 μs
1	0	0	1	16384 ETUs	154657.2 μs
1	0	1	0	32768 ETUs	309314.5 μs
1	0	1	1	65536 ETUs	618628.9 μs
1	1	0	0	131072 ETUs	1237257.8 μs
1	1	0	1	262144 ETUs	2474515.6 μs
1	1	1	0	524288 ETUs	4949031.3 μs
1	1	1	1	RFU	RFU

Tabela 9 – Codificação do FWI no byte do Protocolo no ATQB(ISO/IEC, 2005)

a aplicação é proprietária.

Se a aplicação não é proprietária, bit 3 é 0 e o bit 2 é 1. Os bytes de aplicação são definidos a seguir: o primeiro byte (APP 0) é o AFI da tag, o quarto byte (APP 3) contém o número de aplicações da tag, e o segundo e o terceiro byte contêm o CRC_B do AID, como definido na ISO/IEC 7816-5. O AID (*Application Identifier Code*) é um código identificador de aplicação multibyte que identifica uma aplicação fornecida ou emitida e indica se a aplicação fornecida é registrada com ISO ou com algum órgão de norma nacional.

3.2.5 Comando ATTRIB

O envio do comando ATTRIB (com um correspondente PUPI) depois de uma resposta ATQB seleciona a tag e a coloca no estado ativo. Ele Também atribui o CID para a tag e define os parâmetros de comunicação opcionais. O comando ATTRIB, como é mostrado na Figura 29, pode também conter um comando embutido de alta camada (nos bytes INF), se a tag suportar a parte 4.

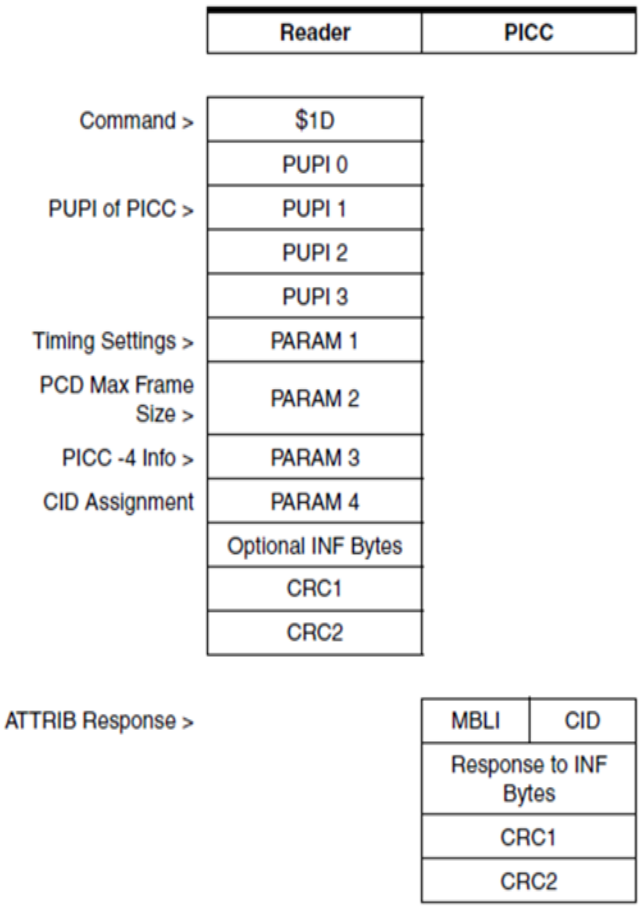

Figura 29 – Comando e resposta ATTRIB(ISO/IEC, 2005).

Se a tag não suporta a parte 4 ou o comando ATTRIB não contém bytes INF, então a resposta ATTRIB não pode conter os bytes INF. Os quatro bits inferiores retornam o valor CID atribuído pelo ATTRIB. Os quatro bits superiores da resposta ATTRIB representam o maior índice de comprimento do buffer (MBLI), que comunica ao leitor quantos bytes da tag são capazes de receber frames encadeados. Se a tag não suportar encadeamento de frames, então o parâmetro é $0.

O byte 1 do Param (Figura 30) contém $00 para tags que suportam somente as configurações padrão para mínimos TR e TR1, e exige ambos os campos SOF e EOF. Para a taxa de dados padrão de 106 Kbps, os bits de 4 a 7 do Param 2 são $0. O tamanho máximo dos bits do frame do leitor do Param 2 são codificados (Tabela 10).

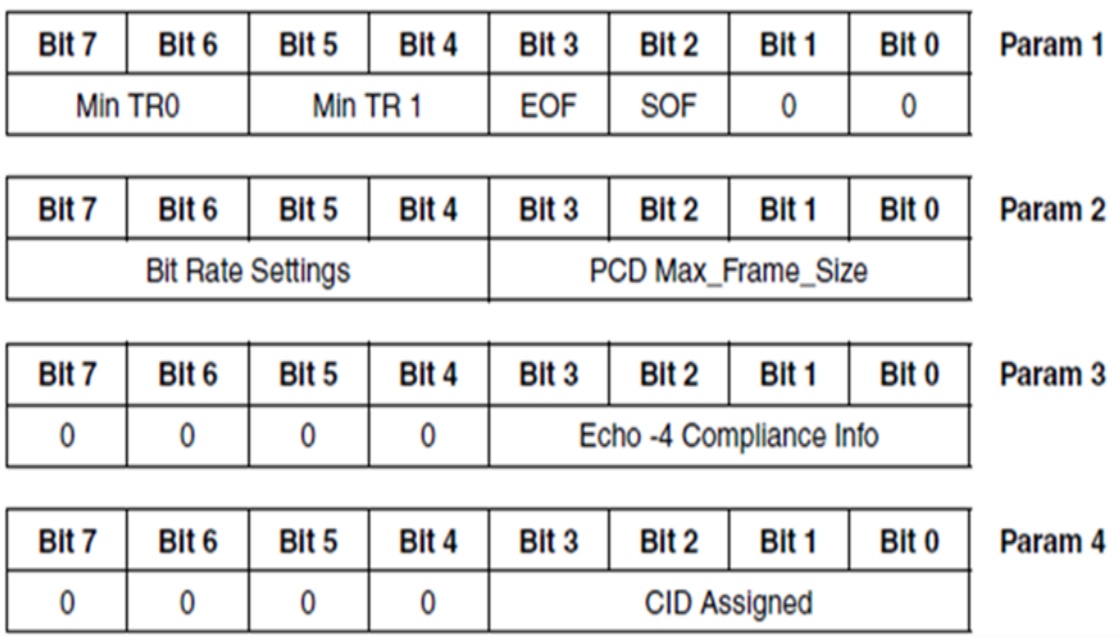

Figura 30 – Definição do byte Param do comando ATTRIB(ISO/IEC, 2005).

Bit 3	Bit 2	Bit 1	Bit 0	Max Frame
0	0	0	0	16 Bytes
0	0	0	1	24 Bytes
0	0	1	0	32 Bytes
0	0	1	1	40 Bytes
0	1	0	0	48 Bytes
0	1	0	1	64 Bytes
0	1	1	0	96 Bytes
0	1	1	1	128 Bytes
1	0	0	0	256 Bytes
1	x	x	x	RFU

Tabela 10 – Codificação do tamanho máximo do Frame do leitor do Param 2(ISO/IEC, 2005)

Os quatro bits superiores do Param 3 são $0. Os quatro bits inferiores do Param 3 são $0 se a tag não é compatível com a parte 4, enquanto o valor $1 indica compatibilidade. Os quatro bits superiores do Param 4 também são $0. Os quatro bits inferiores são usados para atribuir um CID exclusivo para a tag. Se a tag não suportar CIDs, então $0 é enviado. O CID no ATTRIB Param 4 e a resposta ATTRIB são codificados como mostra a Tabela 11.

Bit 3	Bit 2	Bit 1	Bit 0	CID
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	RFU

Tabela 11 – Codificação do CID no Param 4 e resposta do ATTRIB(ISO/IEC, 2005)

3.2.6 Comando HLTB

Enviar o comando *Halt B* (HLTB) com um PUPI correspondente após uma resposta ATQB coloca a tag no estado *Halt*. A resposta HLTB é $00. Depois de responder a HLTB, a tag irá ignorar todos os comandos, exceto WUPB. A Figura 31 mostra o comando e a resposta HLTB.

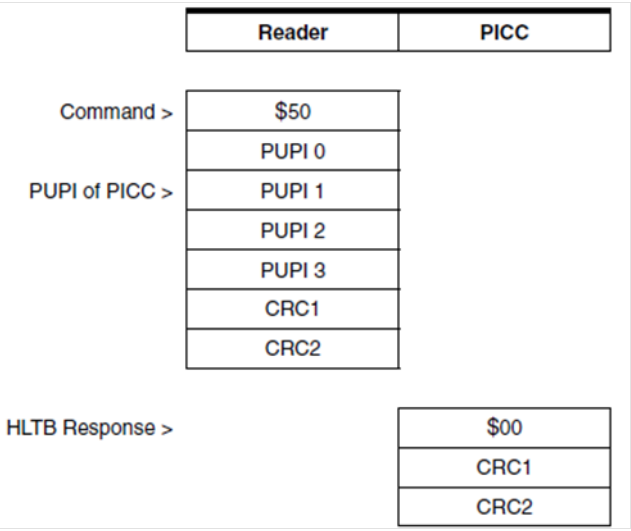

Figura 31 – Comando e Resposta HLTB(ISO/IEC, 2005).

3.3 Definição do bloco a ser implementado

Um bloco que implementa o protocolo anticolisão será construído com base no diagrama de transição dos estados da tag mostrado na Figura 23. Esta arquitetura foi escolhida porque técnicas de criptografia poderiam ser usadas para proteger a privacidade, no entanto, apresentam custo elevado para as tags RFID de baixo custo. Portanto, será considerado inicialmente que a tag passiva de RFID enviará apenas o número do ID para o leitor. Isto também faz com que a tag reduza o consumo de energia, que é de extrema importância neste projeto. Quando a tag transmite os dados armazenados na memória, a energia consumida pelo processador banda base deve ser reduzida para compensar a energia consumida pelo bloco do *front end* analógico. Com a redução do consumo de energia, é possível otimizar o tempo de operação e reduzir os custos da fabricação da tag.

Como apresentado também no item referente a anticolisão, as tags são identificadas por um número de identificação do cartão PUPI, que é usado para comandos diretos para uma tag específica durante o processo de anticolisão. As principais tarefas para a parte digital da tag são armazenar o ID da tag e transmiti-lo quando a tag estiver ligada. Quando a tag não é ligada, ela permanece no estado ocioso. Quando a energia suficiente é induzida na tag, ela entra em estado de execução. Neste estado, para cada clock, os dados armazenados na própria tag são recuperados e enviados para o leitor. Quando os dados são enviados totalmente, a tag entra em estado ocioso.

4 Resultados e Discussão

Este capítulo apresenta a descrição e as simulações em Verilog do bloco anticolisão da tag de RFID. Os códigos foram escritos e testados no *ISE Design Suite*, uma ferramenta da *Xilinx ISE*. A arquitetura que será utilizada para realizar o bloco anticolisão da tag será modelada com base no diagrama de transição dos estados da tag mostrado na Figura 23, que se encontra no item referente à anticolisão entre tags.

4.1 Descrição em Verilog

Com base no diagrama de transição dos estados da tag mostrado na Figura 23, o protocolo anticolisão foi descrito em Verilog. O código completo se encontra listado no anexo A.

O módulo necessitaria inicialmente das entradas e saídas clk", "receber"e "enviar". Contudo, foram adicionadas diversas variáveis para que fosse possível mostrar os sinais internos presentes no bloco para validar o mesmo, já que não foi possível mostrar os sinais internos no simulador ISim.

```
47    always @(rst or posedge clk or negedge clk) begin
48
49    if( rst ) begin // se reset=1
50        state <= 4'b0000; // Vai para o estado "wait for REQB or WUPB"
51    end
52    else  begin
53        case( state )
54
55        4'b0000: begin  //espera por REQB ou WUPB
56
57             if( comando == 8'h05 ) begin //se for $05 (reqb/wupb)
58                state <= 4'b0001; //o estado vai para afi_match
59             end
60             else begin //senao continua no mesmo estado
61                 state <= 4'b0000;
62             end
63        end
64
65        4'b0001: begin // estado "AFI match" (AFI = $3B ou $30 ou $00)
66
67             if( (afi == 8'h3B) || (afi == 8'h30) || (afi == 8'h00) ) begin // $30 3b ou 00
68                state <= 4'b0010; //o estado vai para "does n=1"
69             end
70             else begin //senao volta para o estado anterior
71                 state <= 4'b0000; //wait for reqb/wupb
72             end
```

Figura 32 – Bloco always da maquina de estados e os estados 0000 e 0001

4.1.1 Máquina de estado

Após as declarações das portas de entrada e saída do módulo, começa o bloco "always"(linha 47 da Figura 32), que será executado sempre que o reset for acionado ou

quando ocorrer a borda de subida ou de descida do clock.

Neste processo, cada bloco do diagrama de transição dos estados da tag é um estado. Por exemplo, o bloco "wait for REQB or WUPB"representa o estado 0000 em binário, e o bloco "AFI Match?", o estado 0001 em binário. Os blocos foram numerados para descrever a tag por meio de uma máquina de estados. A Figura 33 mostra essa numeração em decimal.

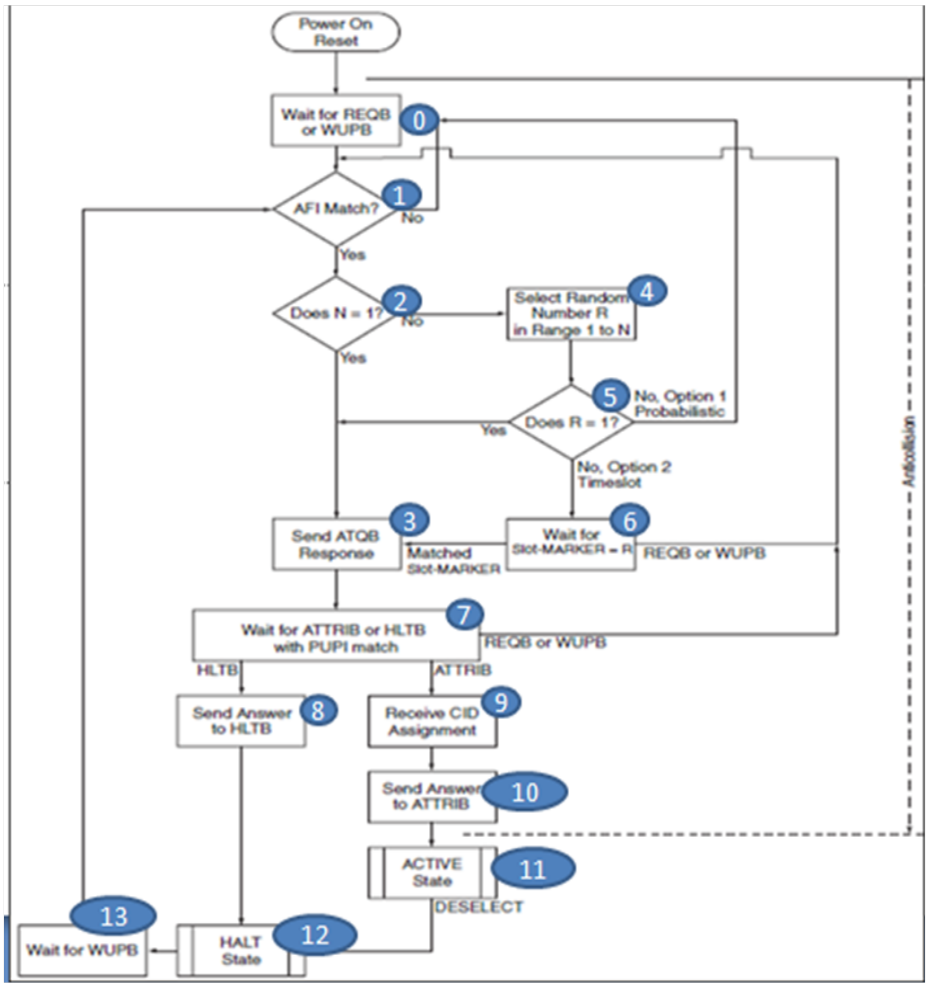

Figura 33 – Numeração em decimal dos blocos do diagrama de transição dos estados da tag

Para descrever a máquina de estados, foi utilizada a declaração "case". Portanto, foi descrita em Verilog uma lógica para que o código se comportasse como o diagrama de transição dos estados da tag. A lógica de cada estado é explicada nos itens a seguir.

1. **Estado 0000 (Wait for REQB or WUPB)**:

   Este estado espera pelos comandos REQB ou WUPB, ou seja, espera que o comando seja igual a $05 em hexadecimal. Se o comando for $05, entra no proximo estado (linha 57). Senão, o estado continua o mesmo, e a tag espera o leitor enviar o comando correto (linha 60).

2. **Estado 0001 (AFI match)**:

   Este estado confere se o AFI do leitor corresponde ao AFI da tag. Como o AFI da tag é $3B, então o AFI correspondente deve ocorrer somente se o leitor transmitir um AFI de $3B, ou $30 ou $00 (linha 67). Se o leitor não enviar o AFI correto, o estado volta para o estado inicial 0000.

```
75        4'b0010: begin //estado "does n=1"
76
77             if( n == 3'b000 ) begin //ou seja 000:n=1
78                state <= 4'b0011; //estado "send atqb response"
79             end
80             else begin //se n!=1
81                state <= 4'b0100; //select random number in range 1 to n
82             end //else begin
83         end //end 4'b0010
84
85          4'b0011: begin //estado "send atqb response"
86             //deve enviar uma resposta ATQB
87             //$50 + PUPI0,1,2,3 + APP0,1,2,3 + Protocol1,2,3 + CRC1,2 (14)
88                //PUPI0,1,2,3 = $11 $22 $33 $44
89                //APP0,1,2,3 = $3B $00 $00 $01
90                //protocol1,2,3=$00 $00 $00
91             enviar <= 112'h50112233443B0000010000000000;
92             //apos enviar, vai para o estado "wait for attrib or hltb with pupi match"
93             state <= 4'b0111;
94          end
```

Figura 34 – Estados 0010 e 0011

3. **Estado 0010 (does n=1)**:

   Os comandos REQB e WUPB contêm um parâmetro "N", que atribui o número de slots disponíveis para o processo de anticolisão. A codificação de "N"é mostrada na Tabela 6. Ou seja, se n=1, a máquina entra no estado 0011 (linha 77 da Figura 34). Senão, entra no estado 0100 (linha 82 da figura 34).

4. **Estado 0011 (send atqb response)**:

   Neste estado, a tag responde ao leitor. Portanto, é enviada uma resposta através do vetor "enviar"(linha 91 da Figura 34) e depois a máquina entra no estado 0111 (linha 93 da Figura 34).

5. **Estado 0100 (select random number in range 1 to n)**:

   O valor gerado pelo gerador do inteiro N é armazenado na variável "r"(linha 98 da Figura 35). A geração deste valor é explicada posteriormente neste documento. O valor gerado é armazenado na variável "r", uma vez que o valor gerado é alterado a cada descida e subida de clock, ou seja, o valor gerado muda constantemente. Por segurança, se o valor aleatório for maior que o valor do slot "S", o valor de "r"será 1 porque o contador começa a contar do 1 (linha 100 da figura 35). Depois, a máquina entra no estado 0101 (linha 104 da figura 35).

6. **Estado 0101 (does r=1)**:

```
96          4'b0100: begin //estado "select random number in range 1 to n"
97
98             r <= r_ale;  //r recebe um valor do contador
99                //por segurança
100            if(r > num) begin //valor R pode conter um valor > num
101               r = 5'b00001; //o contador começa a contar do 1
102            end
103            //depois vai para o estado "does r=1"
104            state <= 4'b0101;
105
106         end
107
108         4'b0101: begin //estado "does r=1"
109
110            if(r == 5'b00001) begin// se r=1 vai para o estado "send atqb response"
111               state <= 4'b0011;
112            end
113            else begin //senao vai para o estado wait for slot-marker = R
114                state <= 4'b0110;
115            end
116         end
```

Figura 35 – Estados 0100 e 0101

Se r=1, a máquina entra no estado 0011, onde será enviada uma resposta ATQB (linha 110 da Figura 35). Senão, a máquina entra no estado 0110, onde a tag espera pelo slot-marker=R (linha 113 da Figura 35).

```
118         4'b0110: begin //estado "wait for slot-marker = R"
119
120            //slot-marker = R : vai para o estado "send atqb response"
121            if(r == slot) begin
122               state <= 4'b0011;
123            end
124            //senao espera por um slot-marker aonde S = R
125            else begin
126               state <= 4'b0110; //continua no mesmo estado
127            end
128         end
129
130         4'b0111: begin //estado "wait for attrib or hltb with pupi match"
131            //espera pelo comando ATTRIB($1D) ou HLTB($50)
132            if( comando == 8'h50 ) begin //$50 = Comando HLTB
133               state <= 4'b1000; //o estado vai para "send answer to HLTB"
134            end
135
136            else if( comando == 8'h1D ) begin //$1D = Comando ATTRIB
137               state <= 4'b1001; //o estado vai para "receive CID assignment"
138            end
139
140            else begin
141               state <= 4'b0111; //continua no mesmo estado
142            end
143         end
```

Figura 36 – Estados 0110 e 0111

7. **Estado 0110 (wait for slot-marker = R)**:

   Neste estado, a tag espera pelo slot-marker=R. Se for igual, entra no estado 0011 (linha 121 da Figura 36). Senão, continua no mesmo estado até que o slot-marker enviado pelo leitor seja igual à variável "R"(linha 125 da Figura 36).

8. **Estado 0111 (wait for ATTRIB or HLTB with PUPI match)**:

   Neste estado, a tag espera pelo comando HLTB ou pelo comando ATTRIB. Se o leitor enviou o comando HLTB, a tag vai para o estado 1000 (linha 132 da Figura

36), onde será enviada uma resposta ao comando HLTB. Caso o leitor tenha enviado o comando ATTRIB, a tag vai para o estado 1001 (linha 136 da Figura 36). Caso nenhum comando tenha sido enviado pelo leitor, o estado continua o mesmo (linha 140 da Figura 36).

```
145         4'b1000: begin //estado "send answer to hltb"
146               //resposta HLTB: $00 + CRC1 + CRC2
147               enviar <= 112'h00123400000000000000000000000;
148               //apos enviar a resposta, vai para o estado "halt state"
149               state <= 4'b1100;
150         end
151
152         4'b1001: begin //estado "receive cid assignment"
153            //comando attrib: $1D + PUPI0,1,2,3 + PARAM1,2,3,4 + CRC1,2 , onde PARAM 4 = CID atribuido
154            cid[7] = receber[23];cid[6] = receber[22];cid[5] = receber[21];cid[4] = receber[20];
155            cid[3] = receber[19];cid[2] = receber[18];cid[1] = receber[17];cid[0] = receber[16];
156            //apos receber o CID, vai para o estado "send answer to attrib"
157            state <= 4'b1010;
158
159         end
160
161         4'b1010: begin //estado "send answer to attrib"
162            //resposta attrib: MBLI=$0 e CID=$0 + CRC1,2 (3bytes)
163            enviar <= 112'h01200000000000000000000000000;
164            //apos enviar, vai para o estado "active state"
165            state <= 4'b1011;
166         end
```

Figura 37 – Estados 1000, 1001 e 1010

9. **Estado 1000 (send answer to HLTB)**:

   Neste estado, é enviada um resposta ao comando HLTB (linha 147 da figura 37). Depois, a tag entra no estado 1100 (linha 149 da figura 37).

10. **Estado 1001 (receive cid assignment)**:

    Neste estado, o CID enviado pelo leitor através do comando ATTRIB (linha 154 da figura 37) é armazenado. Após armazenar o CID, a tag vai para o estado 1010, onde será enviada uma resposta ao comando ATTRIB.

11. **Estado 1010 (send answer to ATTRIB)**:

    Neste estado, é enviada uma resposta ao comando ATTRIB (linha 163 da figura 37). Após enviar a resposta, a máquina entra no estado 1011, onde a tag entra no estado ativo.

12. **Estado 1011 (active state)**:

    Uma vez colocada no estado ativo, a tag está pronta para transações usando comandos do estado ativo. Uma tag no estado ativo que suporta CIDs ignora comandos que não possuem um CID que corresponde ao CID atribuído pelo comando ATTRIB. Até 15 tags que suportam CIDs podem ser ativas simultaneamente. Se a tag não suporta CID, então o leitor colocará uma única tag no estado ativo e completará a transação com o cartão antes de colocá-lo no estado Halt e continuar o procedimento

```
168          4'b1011: begin //estado "active state"
169
170             //nao sei o que faz neste estado
171
172             //depois vai para o estado "halt state"
173             state <= 4'b1100;
174          end
175
176          4'b1100: begin //estado "halt state"
177
178             //zerando as variaveis
179             comando = 8'h00;
180             cid = 8'h00;
181             s = 4'b0000;
182             //depois vai para o estado "wait for wupb"
183             state <= 4'b1101;
184          end
```

Figura 38 – Estados 1011 e 1100

da anticolisão. A tag sai do estado ativo quando um comando DESELECT válido é recebido(o comando DESELECT é definido na ISO/IEC 14443-4). Como a tag não suporta a parte da ISO/IEC 14443-4, entra no estado ativo e depois entra no estado 1100 (linha 173 da figura 38).

13. **Estado 1100 (halt state)**:

    Neste estado, as variáveis armazenadas na tag são zeradas para que a tag consiga realizar novamente uma futura conexão com o leitor, fazendo com que o processo da anti colisão se repita sem nenhum erro. Após zerar algumas variáveis, a tag entra no estado 1101 (linha 183 da figura 38).

```
186          4'b1101: begin //estado "wait for wupb"
187
188             if( comando == 8'h05 ) begin //se for $05 (reqb/wupb)
189
190                if( r_w == 1) begin  //r_w = 0:reqb 1:wupb
191                   //o estado vai para afi_match
192                   state <= 4'b0001;
193                end
194             end
195
196             else begin
197                state <= 4'b1101; //continua no mesmo estado
198             end
199          end
200
201       default: begin //outros casos volta para o estado power on reset
202             state <= 4'b0000;
203       end
204
205     endcase //fecha o case
206
207   end // fecha o else da linha 22
208
209 end //fecha a task always da linha 17
```

Figura 39 – Estado 1101

14. **Estado 1101 (wait for wupb)**:

    Neste estado, a tag espera pelo comando WUPB (linha 188 da figura 39). Se o comando for WUPB (linha 190 da figura 39), a tag volta para o estado 0001 para

começar novamente o processo da anticolisão. Senão, continua no mesmo estado até o leitor enviar o comando WUPB.

Para outros estados não declarados, a tag volta para o estado 0000 (linha 201 da figura 39).

```
210
211 always @(receber) begin  //ler o comando e o AFI sempre que a entrada muda!!!
212
213       //pegando o comando
214       comando[7] = receber[87];comando[6] = receber[86];comando[5] = receber[85];comando[4] = receber[84];
215       comando[3] = receber[83];comando[2] = receber[82];comando[1] = receber[81];comando[0] = receber[80];
216
217       //pegando o afi        afi = receber[79:72]
218       afi[7] = receber[79];afi[6] = receber[78];afi[5] = receber[77];afi[4] = receber[76];
219       afi[3] = receber[75];afi[2] = receber[74];afi[1] = receber[73];afi[0] = receber[72];
220
221       //pegando r/w + N      receber[71]=receber[70]=receber[69]=receber[68]= 0
222       r_w = receber[67]; //0:reqb 1:wupb
223       n[2] = receber[66];n[1] = receber[65];n[0] = receber[64]; //000:n=1 001:n=2 010:n=4 011:n=8 100:n=16
224
225       case(n) //001:n=2 ou 010:n=4 ou 011:n=8 ou 100:n=16
226          3'b001:  num <= 5'b00010; //2 em binario
227          3'b010:  num <= 5'b00100; //4 em binario
228          3'b011:  num <= 5'b01000; //8 em binario
229          3'b100:  num <= 5'b10000; //16 em binario
230       endcase
```

Figura 40 – Bloco always para armazenar as variáveis

4.1.2 Armazenando as variáveis

Simultaneamente ao bloco always que descreve a maquina de estados, existe outro bloco always (linha 211 da Figura 40) que será executado quando o leitor enviar uma mensagem para a tag, ou seja, quando a variável "receber"for modificada. Como a variável "receber"apresenta diversas informações como o comando, o AFI, o número do slot, entre outros, é necessário dividir estas informações para facilitar a construção da máquina de estados da tag. Portanto, neste bloco always serão armazenadas informações encontradas no vetor "receber"em outras variáveis ("comando","afi","r_w","n", entre outros).

A selecão dos comandos REQB ou WUPB são determinados pelo valor do bit 3 do byte PARAM. Se o bit 3 for zero, representa o comando REQB. Se o bit 3 for 1, representa o comando WUPB. Este valor é armazenado na variável "r_w"(linha 222 da Figura 40).

Os comandos REQB e WUPB contêm um parâmetro "N", que atribui o número de slots disponíveis para o processo de anticolisão. A codificação de "N"é mostrada na Tabela 6. Este parâmetro é armazenado na variável "n". Como n é codificado, é utilizada uma declaração "case"(linha 225 da Figura 40) para decodificar o parâmetro "N"com base na Tabela 6.

O leitor irá transmitir comandos Slot-Marker com valor do slot "S"de 2 até "N"para definir o começo de cada timeslot para a anticolisão. A codificação do número de slot do

```
232      //pegando o Slot number   $slot e 5 onde 0001 <= slot <= 1111
233      if(comando == 8'h15 || comando == 8'h25 || comando == 8'h35 || comando == 8'h45 || comando == 8'h55 ||
234         comando == 8'h65 || comando == 8'h75 || comando == 8'h85 || comando == 8'h95 || comando == 8'ha5 ||
235         comando == 8'hb5 || comando == 8'hc5 || comando == 8'hd5 || comando == 8'he5 || comando == 8'hf5) begin
236
237            s[3] = receber[87];s[2] = receber[86];s[1] = receber[85];s[0] = receber[84];
238
239            case(s) //s=0001: slot=2, s=0010: slot=3 , s=0011: slot=4...
240               4'b0000: state <= 4'b0110; //continua no mesmo estado
241               4'b0001: slot <= 5'b00010; //2 em binario
242               4'b0010: slot <= 5'b00011; //3 em binario
243               4'b0011: slot <= 5'b00100; //4 em binario
244               4'b0100: slot <= 5'b00101; //5 em binario
245               4'b0101: slot <= 5'b00110; //6 em binario
246               4'b0110: slot <= 5'b00111; //7 em binario
247               4'b0111: slot <= 5'b01000; //8 em binario
248               4'b1000: slot <= 5'b01001; //9 em binario
249               4'b1001: slot <= 5'b01010; //10 em binario
250               4'b1010: slot <= 5'b01011; //11 em binario
251               4'b1011: slot <= 5'b01100; //12 em binario
252               4'b1100: slot <= 5'b01101; //13 em binario
253               4'b1101: slot <= 5'b01110; //14 em binario
254               4'b1110: slot <= 5'b01111; //15 em binario
255               4'b1111: slot <= 5'b10000; //16 em binario
256            endcase
257      end
258   end
```

Figura 41 – Decodificação e armazenamento do número de solt

byte do comando é mostrada na Tabela 7. A decodificação do número de slot "s"é realizada utilizando uma declaração "case"(linha 239 da Figura 41). O bloco always é terminado (linha 258 a Figura 41) e as informações do vetor "receber"são armazenadas em outras variáveis para facilitar a construção do diagrama de transição dos estados da tag.

4.1.3 Gerador do Inteiro N

Os comandos REQB ou WUPB contêm um inteiro "N"indicando o número de slots atribuídos ao processo de anticolisão para as tags. Se "N"é maior que um, então a tag seleciona um número aleatório "R"em um alcance de 1 a "N".

```
260   always @(posedge clk or negedge clk) begin //Gerador de numeros aleatorios (contador)
261
262      if (r_ale <= num) begin //R vai de 1 a N
263          r_ale <= r_ale + 5'b00001; // r = r + 1
264      end
265
266      else begin  // reseta o contador
267         r_ale <= 5'b00001 ;  //reseta, começa do 1
268      end
269
270   end
271
272   endmodule
273
```

Figura 42 – Gerador do Inteiro N

Para simular um gerador de número aleatorio (RNG) usou-se um contador crescente que incrementa tanto na borda de subida como na de descida do clock (linha 260 da figura 42). Este contador não pode ser considerado um gerador de número aleatório visto

que, se duas ou mais tags entrarem no campo eletromagnético do leitor ao mesmo tempo, as *tags* teriam o mesmo número aleatório. Este contador incrementa de 1 a 16 (linha 263 da Figura 42), que é o valor máximo de "N". Quando o valor aleatório chega ao seu valor máximo, no próximo incremento o valor aleatório é zerado, voltando ao seu valor mínimo 1 (linha 267 da figura 42).

4.1.4 *Testbench* do Bloco Anticolisão

Após o término do código em Verilog do bloco digital da tag, foi realizado um *testbench* para validar o projeto. O *testbench* completo se encontra no anexo A. A análise do funcionamento da tag pode ser feita observando-se o código em Verilog que se encontra no anexo A e o item anterior, que explica detalhadamente o funcionamento de cada estado da tag. A parte principal do *testbench* do bloco é mostrada na figura 43.

```
44    initial begin
45       // Initialize Inputs
46       rst = 1;
47       clk = 0;
48
49       #4;
50       rst = 0;
51       receber = 88'h000000000000000000000;
52       #5;
53       receber = 88'h051100000000000000000;
54       #5;
55       receber = 88'h053B040000000000000000; //0001:n=16
56       #5;
57       //slot marker: slot e $5 + crc1,2 onde slot:0000=nao suporta 0001=2 0010=3 0011=4 ... 1111=16
58       receber = 88'h250000000000000000000; //0010=3
59       #5;
60       receber = 88'h950000000000000000000; //1001=10
61       #5;
62       //receber = 88'h500000000000000000000;  //hltb
63       //comando attrib: $1D + PUPI0,1,2,3 + PARAM1,2,3,4 + CRC1,2 , onde PARAM 4 = CID atribuido
64       receber = 88'h1d11223344000000330000;  //attrib
65       #10;
66       //receber = 88'h051109000000000000000; //wupb
67    end
68
69    always begin
70          #2  clk = ~clk; // Toggle clock every 5 ticks
71    end
```

Figura 43 – Test bench do Bloco Digital

O *testbench* (figura 43) possui valores iniciais rst=1 e clk=0 (linhas 46 e 47, respectivamente). Os atrasos especificam a hora em que os valores atribuídos se propagam através das entradas para as saídas das portas. Na linha 49, colocou-se um delay "#4;". Portanto, o atraso será de 4 nanosegundos para que o rst=0 e o leitor envie o seu primeiro comando para a tag (linhas 50 e 51 respectivamente). Com o reset acionado, a tag entra no estado 0000, lembrando que o vetor "receber"armazena os dados enviados do leitor para a tag.

Na linha 51, o leitor envia um comando inválido para a tag(comando=$00). Neste caso, a tag deve permanecer no estado 0000. Após 5 ns (linha 52), o leitor envia outro

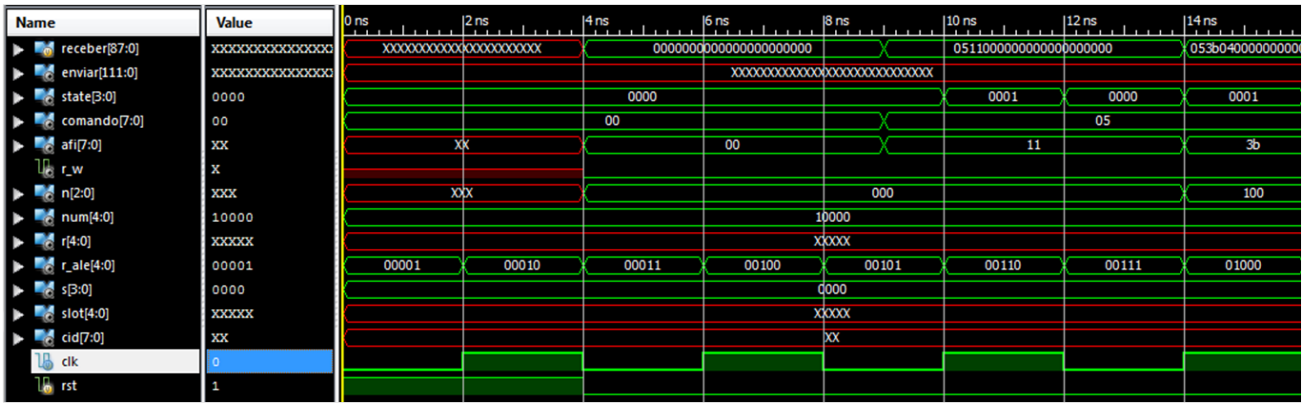

Figura 44 – Simulação do Bloco Digital no simulador ISim Parte 01

comando. Portanto, no tempo 9 ns é enviado outro comando. Isto porque o tempo continua acumulando, somando os 4ns de atraso anterior com os 5 ns do novo atraso.

Comando=$05 representa o comando REQB/WUPB. Logo, verificando o vetor "receber"da linha 53, é possível observar que o leitor enviou: comando=$05, AFI=$11, r_w=0, n=000. Como o comando=$05, a tag entra no estado 0001. Como o AFI=$11(AFI do leitor), o AFI da tag não corresponde. Como o AFI da tag é igual a $3B, o AFI correspondente deve ocorrer somente se o leitor transmitir um AFI de $3B, ou $30 ou $00. Consequentemente, a tag permanece no estado 0001(estado AFI match).

No tempo 14 ns, é enviado outro comando (atraso de 5ns na linha 54). O leitor envia comando=$05, AFI=$3B, r_w=0, n=100 (linha 55). Sendo assim, o AFI da tag corresponde ao AFI do leitor e a tag entra no estado 0010. Como o leitor enviou a mensagem no tempo 14 ns, somente na próxima subida ou descida de clock o estado da tag vai mudar. Desta maneira, a tag entra no estado 0010 no tempo 16 ns, como mostra a figura 44.

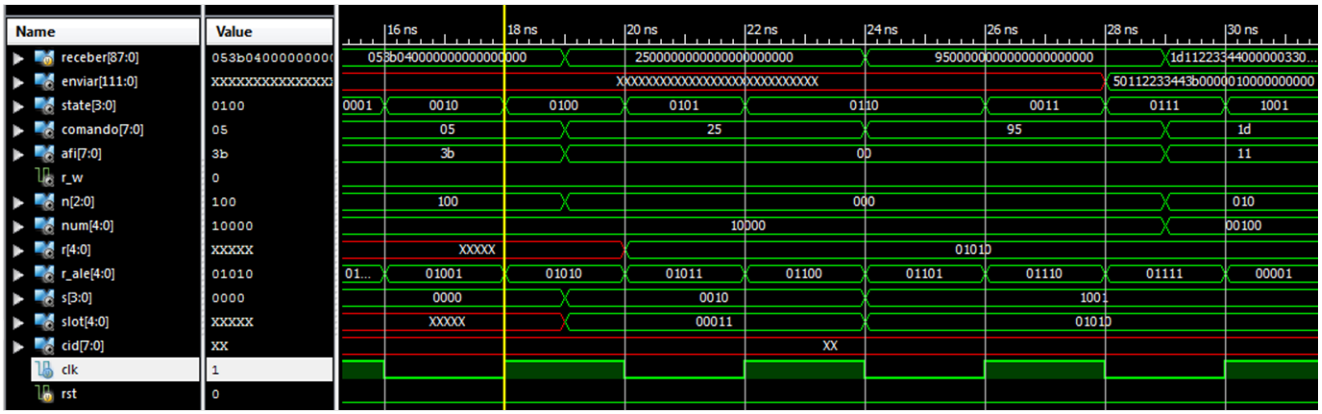

Figura 45 – Simulação do Bloco Digital no simulador ISim Parte 02

Vale observar que r_w=0 representa o comando REQB e que r_w=1 representa o comando WUPB. O parâmetro "n"é um valor codificado e atribui o número de slots disponíveis para o processo de anticolisão. A Tabela 6 mostra o valor de "n"codificado. O valor de "n"decodificado é armazenado no parâmetro "num". Observando a Tabela 6, n=100 representa o número 16. Deste modo, num=10000 (16 em binário). Como o gerador de número aleatório conta de 1 a "n", inicializou-se num=10000(n=16 em decimal). Consequentemente, o valor de "num"não foi alterado.

Em vista de n=100, a tag, que estava no estado 0010, vai para o estado 0100 em sequência (tempo 18 ns). No estado 0100, a tag seleciona um valor do gerador do Inteiro N. Este valor, que fica armazenado no parâmetro "r_ale", é armazenado no parâmetro "r". Isto ocorre porque o gerador do Inteiro N gera números a cada descida e subida de clock e o valor de "r_ale"muda constantemente, como foi explicado anteriormente. Em vista disso, r=01010 (tempo 20 ns). Depois, a tag entra no estado 0101.

Como o valor de "r"é diferente de 1, a tag entra no estado 0110 (tempo 22 ns). Neste estado, a tag espera pelo Comando Slot-Marker. Este comando é mostrado na Figura 45. Neste estado, a tag espera pelo slot-marker=r. No tempo 19 ns, é enviado outro comando pelo leitor (atraso de 5ns na linha 56). Logo, comparando o vetor "receber"da linha 58 com a Figura 45, é possível ver que o leitor enviou o s=0010. A Tabela 7 mostra o valor de "s"codificado. O valor de "s"decodificado é armazenado no parâmetro "slot". Observando a Tabela 7, s=0010 representa o número 3. Deste modo, slot=00011 (3 em binário). Como "r"é diferente do parâmetro "slot", a tag continua no mesmo estado até que slot-marker=r.

No tempo 24 ns, é enviado outro comando pelo leitor (atraso de 5ns na linha 59). Logo, comparando o vetor "receber"da linha 60 com a Figura 45, é possível observar que o leitor enviou s=1001. Observando a Tabela 7, s=1001 representa o número 10. Deste modo, slot=01010 (10 em binário). Consequentemente, slot-marker=R e a tag entra no estado 0011. Neste estado, a tag responde ao leitor. Portanto, é enviada uma resposta através do vetor "enviar"e depois a tag entra no estado 0111 (tempo 28 ns).

Neste estado, a tag espera pelo comando HLTB ou pelo comando ATTRIB. No tempo 29 ns, é enviado outro comando pelo leitor (atraso de 5ns na linha 61). O comando enviado foi igual a $1D (linha 64), que representa o comando ATTRIB. Portanto, a tag vai para o estado 1001. Neste estado, é armazenado o CID enviado pelo leitor através do comando ATTRIB (tempo 32 ns), como mostra a figura 46. Após armazenar o CID, a tag vai para o estado 1010. Neste estado, é enviada uma resposta ao comando ATTRIB (tempo 34 ns). Esta resposta é mostrada na figura 46. Após enviar a resposta, a tag entra no estado 1011 e depois entra no estado 1100 (tempo 36 ns). Neste estado, as variáveis armazenadas na tag são zeradas para que a tag possa realizar novamente uma futura conexão com o leitor, fazendo com que o processo da anticolisão se repita sem nenhum erro.

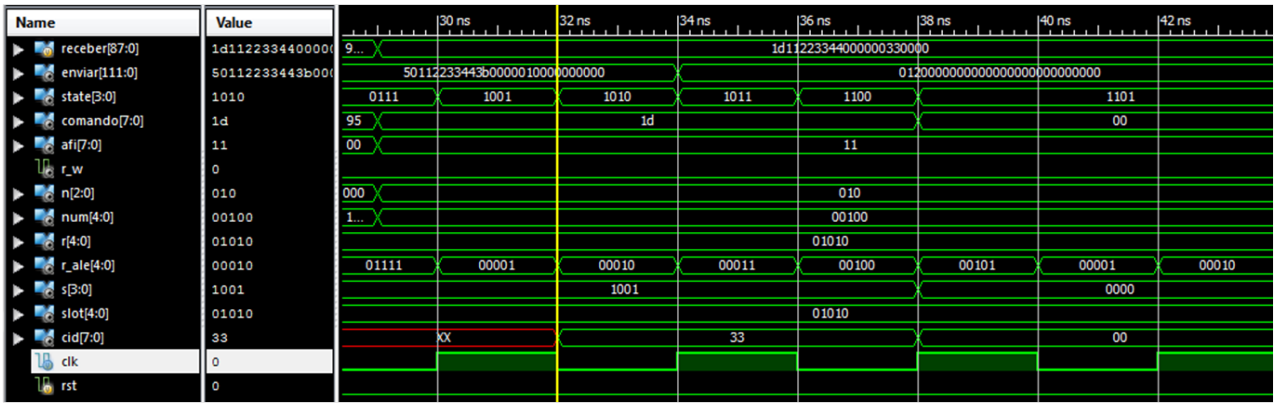

Figura 46 – Simulação do Bloco Digital no simulador ISim Parte 03

Após zerar algumas variáveis, a tag entra no estado 1101(tempo 38 ns). Neste estado, a tag espera pelo comando WUPB. Se o comando for enviado, a tag volta para o estado 0001, para começar novamente o processo da anticolisão. Senão, continua no mesmo estado até que o leitor envie o comando WUPB.

4.2 Código CRC

O CRC foi explicado na seção referente à detecção de erros CRC. Um código simples do CRC em Verilog foi elaborado utilizando o modelo encontrado no site Asic World(TALA, 2014) como referência.

Em hardware, a codificação e decodificação CRC_B é realizada por registradores (flip-flops D) com deslocamento cíclico de 16 etapas e com portas de realimentação apropriadas (portas XOR). O CRC_B representa o restante da divisão dos dados contidos no frame pelo polinômio 16 + 12 + 5 + 1. Portanto, este código representa o CRC serial, de 16 bits, e com o valor inicial $FFFF dos registradores (flip-flops D). O código é mostrado na figura 47.

O módulo apresenta uma entrada chamada "data_in", onde os dados serão colocados serialmente. A saída será um vetor de 16 bits ("crc_out"), que armazena o cálculo do CRC. Na linha 10, o registrador possui esse valor porque o valor inicial dos registradores é $FFFF, como foi explicado anteriormente. A cada subida de clock, o CRC é calculado (linha 12). A descrição lógica é apresentada entre as linhas 14 e 29. Portanto, se o dado a ser calculado for de 8 bits, o clock deve possuir 8 subidas de clock para calcular o CRC corretamente. Se o dado de entrada tiver 16 bits, o clock deve possuir 16 subidas de clock, e assim sucessivamente. Para a simulação deste código, utilizou-se o *testbench* mostrado na figura 48 .

Neste *testbench*, será calculado o CRC para um dado de entrada de 8 bits. Será

```
4  module crc_simp(clk, data_in, crc_out);
5  //-----------Input Ports---------------
6  input clk     ;
7  input data_in ;
8  //-----------Output Ports---------------
9  output [15:0] crc_out;
10 reg   [15:0] crc_out = 16'hFFFF;
11 // Logic to CRC Calculation
12 always @ (posedge clk) begin
13
14       crc_out[0]  <= data_in ^ crc_out[15];
15       crc_out[1]  <= crc_out[0];
16       crc_out[2]  <= crc_out[1];
17       crc_out[3]  <= crc_out[2];
18       crc_out[4]  <= crc_out[3];
19       crc_out[5]  <= crc_out[4] ^ data_in ^ crc_out[15];
20       crc_out[6]  <= crc_out[5];
21       crc_out[7]  <= crc_out[6];
22       crc_out[8]  <= crc_out[7];
23       crc_out[9]  <= crc_out[8];
24       crc_out[10] <= crc_out[9];
25       crc_out[11] <= crc_out[10];
26       crc_out[12] <= crc_out[11] ^ data_in ^ crc_out[15];
27       crc_out[13] <= crc_out[12];
28       crc_out[14] <= crc_out[13];
29       crc_out[15] <= crc_out[14];
```

Figura 47 – Código verilog CRC

```
5     // Inputs
6     reg clk;
7     reg data_in;
8
9     // Outputs
10    wire [15:0] crc_out;
11
12    // Instantiate the Unit Under Test (UUT)
13    crc_simp uut (
14       .clk(clk),
15       .data_in(data_in),
16       .crc_out(crc_out)
17    );
18
19    initial begin
20       // Initialize Inputs
21       clk = 0;
22       data_in = 0;
23
24    end
25
26    always begin
27          #1  clk = ~clk; // Toggle clock every 5 ticks
28    end
29
30 endmodule
```

Figura 48 – Testbench CRC

colocado data_in = 0 (linha 22), ou seja, o vetor de entrada é nulo (data_in[8] = 0,0,0,0,0,0,0,0). O simulador ISim mostra as formas de ondas de sinais de entrada e de saída do módulo (figura 49).

O valor correto do CRC é o valor encontrado após 8 subidas de clock. Portanto, o CRC calculado foi $e1f0 em hexadecimal. Para validar este valor, utilizou-se duas calculadoras de CRC online (BIES, 2015), (ANALYTICS; METRIKA, 2015). Elas calculam vários valores de CRC. Isto ocorre devido à existência de diversos CRCs para a detecção de um erro de transmissão. Os valores encontrados nessas calculadoras foram os mesmos encontrados no simulador ISim. Foram testados outros valores de entrada e mesmo assim

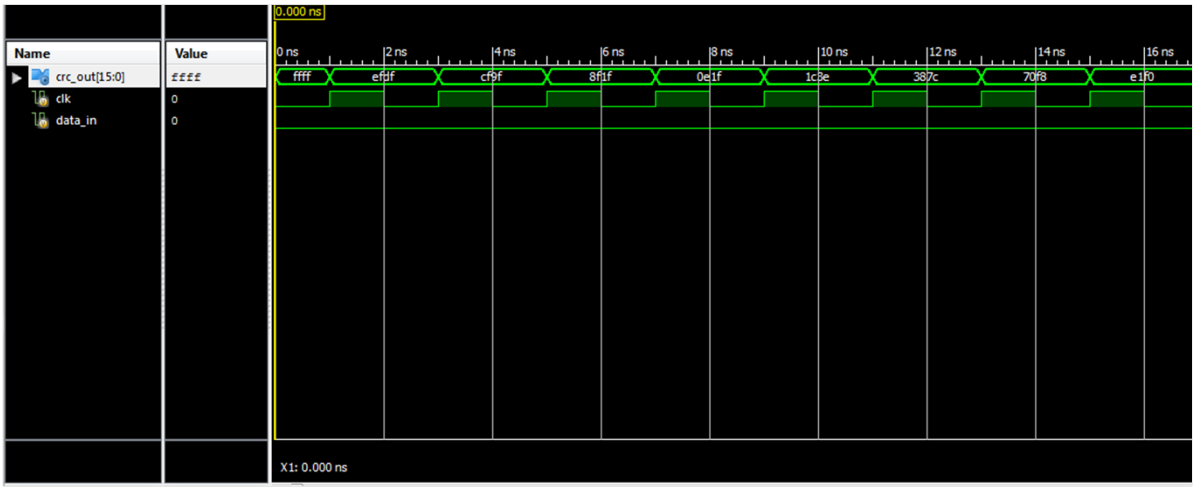

Figura 49 – Simulação CRC no simulador ISim

os valores encontrados permaneceram idênticos. Deste modo, foi possível validar o cálculo do CRC.

Não foi possível implementar o código CRC juntamente com o código principal da tag, que contém a máquina de estados. Era necessário uma mudança no código, onde seria trocado o clock por um loop for. Isto porque o índice do loop incrementado teria a função do clock, ou seja, teria o valor do tamanho dos bits de entrada do dado para o cálculo do CRC, visto que o tamanho das mensagens calculadas podem variar dependendo do comando enviado pelo leitor. Todavia, utilizando o loop for, o CRC não era calculado. Modificando o código anterior colocando um loop for (figura 50), onde o índice incrementa de 0 a 7, ou seja, incrementa 8 vezes, o valor do CRC era incorreto, resultando em um valor incoerente $xfxx (figura 51).

```
14  initial begin
15
16     for(i=0; i < 8; i = i + 1 ) begin
17
18         crc_out[0]  <= data_in ^ crc_out[15];
19         crc_out[1]  <= crc_out[0];
20         crc_out[2]  <= crc_out[1];
21         crc_out[3]  <= crc_out[2];
22         crc_out[4]  <= crc_out[3];
23         crc_out[5]  <= crc_out[4] ^ data_in ^ crc_out[15];
24         crc_out[6]  <= crc_out[5];
25         crc_out[7]  <= crc_out[6];
26         crc_out[8]  <= crc_out[7];
27         crc_out[9]  <= crc_out[8];
28         crc_out[10] <= crc_out[9];
29         crc_out[11] <= crc_out[10];
30         crc_out[12] <= crc_out[11] ^ data_in ^ crc_out[15];
31         crc_out[13] <= crc_out[12];
32         crc_out[14] <= crc_out[13];
33         crc_out[15] <= crc_out[14];
34
35     end  //for
36
37 end //always
38
39 endmodule
```

Figura 50 – Código verilog CRC utilizando o loop for

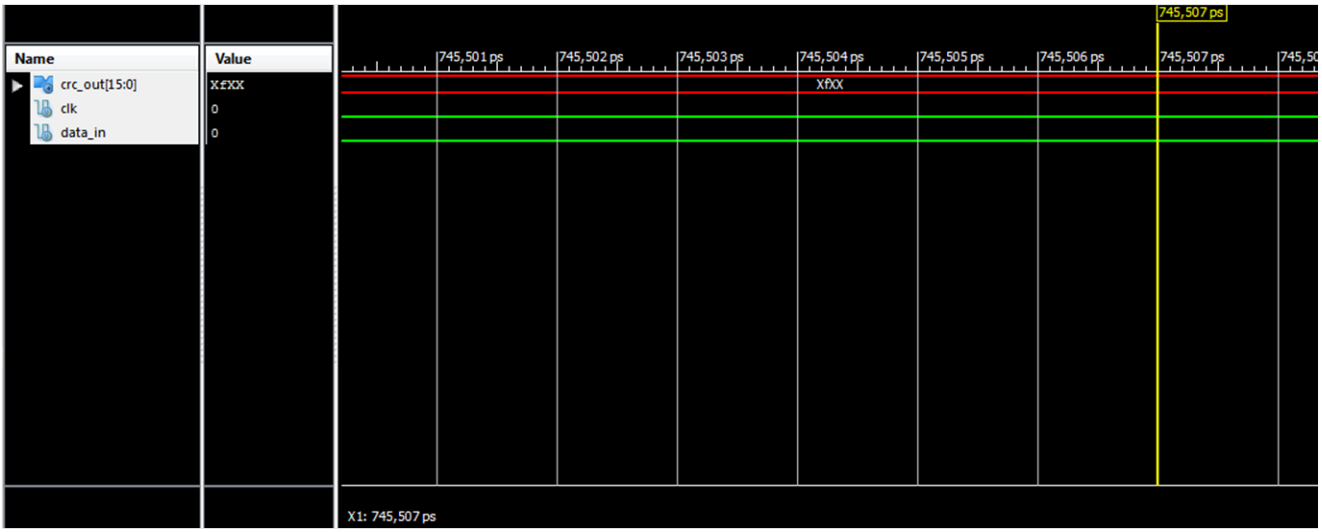

Figura 51 – Simulação CRC utilizando o loop for no simulador ISim

5 Conclusão

Este trabalho tinha como proposta inicial o projeto e implementação em ASIC dos blocos do processador banda base da tag de RFID utilizando o fluxo de projeto para circuitos integrados digitais. Contudo, devido à indisponibilidade da licença do software necessário para realizar o projeto, o foco do trabalho foi alterado para a comunicação entre o leitor e a tag, mais especificamente para a definição, descrição e simulação em Verilog do protocolo anticolisão. Sendo assim, a tag pode realizar o processo de anticolisão garantindo que apenas uma tag de cada vez se comunique com o leitor. Foi considerado que a tag passiva de RFID mandava apenas o número do ID para o leitor, e que as principais tarefas para a parte digital da tag são armazenar e transmitir o ID da tag.

O bloco foi modelado com base no diagrama de transição dos estados da tag mostrado na figura 23. A validação foi realizada utilizando um *testbench* contendo os estímulos configurados de acordo com os campos definidos no protocolo ISO/IEC 14443, que é um padrão internacional para Smart Cards sem contato, operando a 13,56 MHz em estreita proximidade de aproximadamente 10 cm entre o leitor e a tag. Considerou-se que as tags são identificadas por um número de identificação do cartão PUPI, que é usado para comandos diretos para uma tag específica durante o processo de anticolisão. Sendo assim, o ID da tag é representado pelo PUPI que é enviado para o leitor na resposta ATQB, que é representado por um dos estados da máquina. Portanto, não foi necessário modelar uma memória ROM.

Um gerador de números aleatórios (RNG), necessário para efetuar o processo da anticolisão, foi simulado e modelado como um contador crescente, que incrementa de 1 a 16. Este contador não pode ser considerado um gerador de número aleatório visto que, se duas ou mais tags entrarem no campo eletromagnético do leitor ao mesmo tempo, as *tags* teriam o mesmo número aleatório. O código CRC, necessário para a detecção de erros na transmissão, foi elaborado separadamente e validado por meio de simulação, tendo como referência os valores calculados por duas calculadoras CRC online(BIES, 2015), (ANALYTICS; METRIKA, 2015). Não foi possível implementar o código CRC juntamente com o código principal da tag, que contém a máquina de estados. Era necessária uma mudança no código, trocando-se o clock por um loop *for* que, quando incrementado, teria a função do clock, ou seja, teria o valor do tamanho dos bits de entrada do dado para o cálculo do CRC, visto que o tamanho das mensagens calculadas pode variar, dependendo do comando enviado pelo leitor. Todavia, o resultado encontrado era incorreto e gerava um valor incoerente.

Outra limitação deste trabalho foi o fato de adotar, para a simulação, formato

ideal de dados, visto que a validação do projeto seria muito complexa devido ao grande tamanho dos dados tanto de entrada (vetor "receber", que tem comprimento máximo de 88 bits) como de saída (vetor "enviar", que tem comprimento máximo de 112 bits). Além disso, a frequência das simulações não estão de acordo com as especificações da ISO, validou somente a máquina de estado em alto nível. Sendo assim, o formato de dados apresentado e a frequência adequada devem ser adotadas para futuros testes entre a tag e o leitor. Para futuros trabalhos, deve-se obter mais detalhes referente ao funcionamento do estado ativo e juntar o código CRC com o código principal da tag, que se encontra no anexo A. Assim, será possível realizar a simulação mista juntamente com os blocos do *front end* análogico da tag. É recomendável que a máquina de estado funcione somente na borda de subida ou de descida. Portanto, deve-se fazer esta alteração para projetos futuros.

Referências

ANALYTICS, G.; METRIKA, Y. Online crc calculator. 2015. Disponível em: <http://crccalc.com/>. Citado 2 vezes nas páginas 75 e 78.

ATMEL CORPORATION. *Understanding the Requirements of ISO/IEC 14443 for Type B Proximity Contactless Identification Cards*: Requirements of iso/iec 14443 type b proximity contactless identification cards. San José, Califórnia, 2005. Citado 19 vezes nas páginas 15, 17, 39, 40, 44, 45, 46, 47, 48, 51, 53, 54, 55, 56, 57, 58, 59, 60 e 61.

BARAL1, J. S. R. N. Design of compliant passive digital block of read-only rfid tag. In: *International Journal of Engineering and Science*. [S.l.: s.n.], 2012. v. 1, p. 1–07. Citado 3 vezes nas páginas 15, 35 e 36.

BIES, L. On-line crc calculation and free library. 2015. Disponível em: <https://www.lammertbies.nl/comm/info/crc-calculation.html>. Citado 2 vezes nas páginas 75 e 78.

BOMMENA, M. T. I. S. Microchip an1148, cyclic redundancy check (crc). Microchip Technology Inc., 2008. Citado 2 vezes nas páginas 15 e 48.

COE, C. d. E. e. R. R. O que É rfid. 2005. Disponível em: <http://www.rfid-coe.com.br/_Portugues/OqueERFID.aspx>. Citado 3 vezes nas páginas 24, 25 e 28.

FILHO, M. C. P. Modelagem em verilog-ams de uma tag passiva de rfid e projeto elétrico do demodulador ask. Universidade de Brasília - UnB, 2014. Citado 5 vezes nas páginas 11, 13, 15, 25 e 26.

GUIMARAES, R. R. do N. R. Modelagem de sistemas heterogêneos utilizando frameworks baseados em moc. Universidade de Brasília - UnB, 2013. Citado na página 25.

KOTHAMASU, R. P. M. Power feeding and digital block design of multi-standard passive rfid tag). Department of ECE, Indian Institute of Technology Guwahati,India, 2012. Citado 3 vezes nas páginas 15, 36 e 37.

MARKHAM. Understanding rfid (radio frequency identification). Canadá, 2012. Citado 2 vezes nas páginas 17 e 38.

MARTINS, V. A. Rfid, identificação por radiofrequência, inovações tecnológicas fitec, inteligência em telecomunicações teleco. 2005. Disponível em: <http://www.teleco.com.br/pdfs/tutorialrfid.pdf>. Citado 3 vezes nas páginas 17, 33 e 38.

NETO, J. A. R. de S. Teste, projeto e verificação funcional de uma tag de rfid de 13,56 mhz. Universidade de Brasília - UnB, 2015. Citado 3 vezes nas páginas 11, 13 e 26.

PINHEIRO, J. M. S. Rfid - identificação por radiofreqüência. 2004. Disponível em: <http://www.projetoderedes.com.br/artigos/artigo_identificacao_por_radiofrequencia.php>. Citado na página 29.

PORTO, T. Entendendo um pouco sobre rfid. 2005. Disponível em: <http://imasters.com.br/artigo/3731/tendencias/entendendo-um-pouco-sobre-rfid/>. Citado na página 29.

PUHLMANN, H. Introdução à tecnologia de identificação rfid. 2015. Disponível em: <http://www.embarcados.com.br/introducao-a-tecnologia-de-identificacao-rfid/>. Citado 4 vezes nas páginas 15, 30, 33 e 34.

SANGREMAN, T. C. A. O que é rfid?,utilidades práticas,como funciona. Universidade Federal do Rio de Janeiro, 2003. Disponível em: <http://www.gta.ufrj.br/grad/07_1/rfid/RFID_arquivos/Index.htm>. Citado 7 vezes nas páginas 15, 24, 28, 29, 30, 31 e 32.

SCHMIDT, M. T. I. T. Microchip an730, crc generating and checking. Microchip Technology Inc., 2000. Citado 2 vezes nas páginas 15 e 49.

TALA, D. K. Serial crc. 2014. Disponível em: <http://www.asic-world.com/examples/verilog/serial_crc.html>. Citado na página 74.

TUTORIALSPOINT.COM. Dcn - digital transmission. 2015. Disponível em: <http://www.tutorialspoint.com/data_communication_computer_network/digital_transmission.htm>. Citado 2 vezes nas páginas 15 e 41.

ZHANG WEIPING JING, B. J. W. Implementation and design of digital system for high frequency rfid tag chip. In: *Sensors & Transducers*. [S.l.: s.n.], 2014. v. 178, p. 123–129. Citado 2 vezes nas páginas 15 e 35.

Anexos

ANEXO A – Código Verilog do Bloco Digital

Neste anexo se encontra o código completo em verilog e o test bench do bloco digital da tag.

```
1  //comandos:
2  //reqb/wupb: $05
3  //slot marker: $slot e 5
4  //attrib: $1D
5  //hltb: $50
6
7  //PUPI nao é obrigado a ser um valor fixo, PUPI:Pseudo Unique PICC Identifier
8  //PUPI, valor definido: PUPI0,1,2,3 = $11 $22 $33 $44
9  //APP 0=$3B é o AFI da tag
10 //APP 1=$00 e APP2=$00 contém o CRC_B do AID como definido na ISO/IEC 7816-5
11 //APP 3=$01 contém o número de aplicações da tag
12
13 //Protocol1,2,3
14 //MBLI:quantos bytes da tag é capaz de receber frames encadeados
15
16 module mealy(rst, clk, receber, enviar, state, comando, afi, r_w, n, r,r_ale,num,s,slot,cid);
17
18    input rst;
19    input clk;
20    input [87:0] receber; //tag recebe do leitor
21    output [111:0] enviar; //tag envia para o leitor
22    output [3:0] state; // mostra o estado //retirar!!!!!!!!!!
23    output [7:0] comando; // mostra o estado //retirar!!!!!!!!
24    output [7:0] afi; // mostra o estado //retirar!!!!!!!!!!!!
25    output r_w; // 0:reqb 1:wupb
26    output [2:0] n; //valor de n
27    output [4:0] r; //numero aleatorio
```

```
28    output [4:0] r_ale; //numero aleatorio
29    output [4:0] num; // converter n
30    output [3:0] s; //pega o slot-marker
31    output [4:0] slot;
32    output [7:0] cid; //recebe o CID atribuido pelo leitor
33
34    reg [7:0] comando = 8'h00; // armazena o comando(8 bits)
35    reg [3:0] state; //estado vai de 0 a 15
36    reg [111:0] enviar;// envia para o leitor
37    reg [7:0] afi; // armazena o AFI da tag
38    reg r_w; //armazenar r_w
39    reg [2:0] n; //armazenar n
40    reg [4:0] r;
41    reg [4:0] r_ale = 5'b00000; //para o contador
42    reg [4:0] num = 5'b10000; //maior valor de N
43    reg [3:0] s = 4'b0000;
44    reg [4:0] slot;
45    reg [7:0] cid; //armazena cid
46
47    always @(rst or posedge clk or negedge clk) begin
48
49    if( rst ) begin // se reset=1
50        state <= 4'b0000; // Vai para o estado "wait for REQB or WUPB"
51    end
52    else  begin
53        case( state )
54
```

```
55          4'b0000: begin  //espera por REQB ou WUPB
56
57               if( comando == 8'h05 ) begin //se for $05 (reqb/wupb)
58                  state <= 4'b0001; //o estado vai para afi_match
59               end
60               else begin //senao continua no mesmo estado
61                   state <= 4'b0000;
62               end
63          end
64
65          4'b0001: begin // estado "AFI match" (AFI = $3B ou $30 ou $00)
66
67               if( (afi == 8'h3B) || (afi == 8'h30) || (afi == 8'h00) ) begin // $30 3b ou 00
68                  state <= 4'b0010; //o estado vai para "does n=1"
69               end
70               else begin //senao volta para o estado anterior
71                   state <= 4'b0000; //wait for reqb/wupb
72               end
73          end
74
75          4'b0010: begin //estado "does n=1"
76
77               if( n == 3'b000 ) begin //ou seja 000:n=1
78                  state <= 4'b0011; //estado "send atqb response"
79               end
80               else begin //se n!=1
81                  state <= 4'b0100; //select random number in range 1 to n
```

```
82               end //else begin
83           end //end 4'b0010
84
85            4'b0011: begin //estado "send atqb response"
86               //deve enviar uma resposta ATQB
87               //$50 + PUPI0,1,2,3 + APP0,1,2,3 + Protocol1,2,3 + CRC1,2 (14)
88                  //PUPI0,1,2,3 = $11 $22 $33 $44
89                  //APP0,1,2,3 = $3B $00 $00 $01
90                  //protocol1,2,3=$00 $00 $00
91               enviar <= 112'h50112233443B0000010000000000;
92               //apos enviar, vai para o estado "wait for attrib or hltb with pupi match"
93               state <= 4'b0111;
94            end
95
96            4'b0100: begin //estado "select random number in range 1 to n"
97
98               r <= r_ale;  //r recebe um valor do contador
99                  //por segurança
100              if(r > num) begin //valor R pode conter um valor > num
101                 r = 5'b00001; //o contador começa a contar do 1
102              end
103              //depois vai para o estado "does r=1"
104              state <= 4'b0101;
105
106           end
107
```

```
108           4'b0101: begin //estado "does r=1"
109
110              if(r == 5'b00001) begin// se r=1 vai para o estado "send atqb response"
111                 state <= 4'b0011;
112              end
113              else begin //senao vai para o estado wait for slot-marker = R
114                  state <= 4'b0110;
115              end
116           end
117
118           4'b0110: begin //estado "wait for slot-marker = R"
119
120              //slot-marker = R : vai para o estado "send atqb response"
121              if(r == slot) begin
122                 state <= 4'b0011;
123              end
124              //senao espera por um slot-marker aonde S = R
125              else begin
126                 state <= 4'b0110; //continua no mesmo estado
127              end
128           end
129
130           4'b0111: begin //estado "wait for attrib or hltb with pupi match"
131              //espera pelo comando ATTRIB($1D) ou HLTB($50)
132              if( comando == 8'h50 ) begin //$50 = Comando HLTB
133                 state <= 4'b1000; //o estado vai para "send answer to HLTB"
134              end
```

```
135
136             else if( comando == 8'h1D ) begin //$1D = Comando ATTRIB
137                state <= 4'b1001; //o estado vai para "receive CID assignment"
138             end
139
140             else begin
141                state <= 4'b0111; //continua no mesmo estado
142             end
143          end
144
145          4'b1000: begin //estado "send answer to hltb"
146                //resposta HLTB: $00 + CRC1 + CRC2
147                enviar <= 112'h00123400000000000000000000000;
148                //apos enviar a resposta, vai para o estado "halt state"
149                state <= 4'b1100;
150          end
151
152          4'b1001: begin //estado "receive cid assignment"
153             //comando attrib: $1D + PUPI0,1,2,3 + PARAM1,2,3,4 + CRC1,2 , onde PARAM 4 = CID atribuido
154             cid[7] = receber[23];cid[6] = receber[22];cid[5] = receber[21];cid[4] = receber[20];
155             cid[3] = receber[19];cid[2] = receber[18];cid[1] = receber[17];cid[0] = receber[16];
156             //apos receber o CID, vai para o estado "send answer to attrib"
157             state <= 4'b1010;
158
159          end
160
```

```
161          4'b1010: begin //estado "send answer to attrib"
162             //resposta attrib: MBLI=$0 e CID=$0 + CRC1,2 (3bytes)
163             enviar <= 112'h01200000000000000000000000000;
164             //apos enviar, vai para o estado "active state"
165             state <= 4'b1011;
166          end
167
168          4'b1011: begin //estado "active state"
169
170             //nao sei o que faz neste estado
171
172             //depois vai para o estado "halt state"
173             state <= 4'b1100;
174          end
175
176          4'b1100: begin //estado "halt state"
177
178             //zerando as variaveis
179             comando = 8'h00;
180             cid = 8'h00;
181             s = 4'b0000;
182             //depois vai para o estado "wait for wupb"
183             state <= 4'b1101;
184          end
185
```

```
186          4'b1101: begin //estado "wait for wupb"
187
188             if( comando == 8'h05 ) begin //se for $05 (reqb/wupb)
189
190                if( r_w == 1) begin  //r_w = 0:reqb 1:wupb
191                   //o estado vai para afi_match
192                   state <= 4'b0001;
193                end
194             end
195
196             else begin
197                state <= 4'b1101; //continua no mesmo estado
198             end
199          end
200
201        default: begin //outros casos volta para o estado power on reset
202             state <= 4'b0000;
203        end
204
205      endcase //fecha o case
206
207    end // fecha o else da linha 22
208
209 end //fecha a task always da linha 17
210
```

```
211 always @(receber) begin  //ler o comando e o AFI sempre que a entrada muda!!!
212
213       //pegando o comando
214       comando[7] = receber[87];comando[6] = receber[86];comando[5] = receber[85];comando[4] = receber[84];
215       comando[3] = receber[83];comando[2] = receber[82];comando[1] = receber[81];comando[0] = receber[80];
216
217       //pegando o afi        afi = receber[79:72]
218       afi[7] = receber[79];afi[6] = receber[78];afi[5] = receber[77];afi[4] = receber[76];
219       afi[3] = receber[75];afi[2] = receber[74];afi[1] = receber[73];afi[0] = receber[72];
220
221       //pegando r/w + N      receber[71]=receber[70]=receber[69]=receber[68]= 0
222       r_w = receber[67]; //0:reqb 1:wupb
223       n[2] = receber[66];n[1] = receber[65];n[0] = receber[64]; //000:n=1 001:n=2 010:n=4 011:n=8 100:n=16
224
225       case(n) //001:n=2 ou 010:n=4 ou 011:n=8 ou 100:n=16
226          3'b001:  num <= 5'b00010; //2 em binario
227          3'b010:  num <= 5'b00100; //4 em binario
228          3'b011:  num <= 5'b01000; //8 em binario
229          3'b100:  num <= 5'b10000; //16 em binario
230       endcase
231
232       //pegando o Slot number   $slot e 5 onde 0001 <= slot <= 1111
233       if(comando == 8'h15 || comando == 8'h25 || comando == 8'h35 || comando == 8'h45 || comando == 8'h55 ||
234          comando == 8'h65 || comando == 8'h75 || comando == 8'h85 || comando == 8'h95 || comando == 8'ha5 ||
235          comando == 8'hb5 || comando == 8'hc5 || comando == 8'hd5 || comando == 8'he5 || comando == 8'hf5) begin
236
237             s[3] = receber[87];s[2] = receber[86];s[1] = receber[85];s[0] = receber[84];
```

```
238
239             case(s) //s=0001: slot=2, s=0010: slot=3 , s=0011: slot=4...
240                4'b0000: state <= 4'b0110; //continua no mesmo estado
241                4'b0001: slot <= 5'b00010; //2 em binario
242                4'b0010: slot <= 5'b00011; //3 em binario
243                4'b0011: slot <= 5'b00100; //4 em binario
244                4'b0100: slot <= 5'b00101; //5 em binario
245                4'b0101: slot <= 5'b00110; //6 em binario
246                4'b0110: slot <= 5'b00111; //7 em binario
247                4'b0111: slot <= 5'b01000; //8 em binario
248                4'b1000: slot <= 5'b01001; //9 em binario
249                4'b1001: slot <= 5'b01010; //10 em binario
250                4'b1010: slot <= 5'b01011; //11 em binario
251                4'b1011: slot <= 5'b01100; //12 em binario
252                4'b1100: slot <= 5'b01101; //13 em binario
253                4'b1101: slot <= 5'b01110; //14 em binario
254                4'b1110: slot <= 5'b01111; //15 em binario
255                4'b1111: slot <= 5'b10000; //16 em binario
256             endcase
257       end
258 end
259
```

```
260 always @(posedge clk or negedge clk) begin //Gerador de numeros aleatorios (contador)
261
262    if (r_ale <= num) begin //R vai de 1 a N
263        r_ale <= r_ale + 5'b00001; // r = r + 1
264    end
265
266    else begin  // reseta o contador
267       r_ale <= 5'b00001 ;  //reseta, começa do 1
268    end
269
270 end
271
272 endmodule
273
```

Figura 52 – Código em verilog do Bloco Digital da tag

```
1  //`timescale 1ns / 1ps
2
3  module mealy_tb;
4
5     // Inputs  //entrada vira reg
6     reg rst;
7     reg clk;
8     reg [87:0] receber;
9
10    // Outputs
11    wire [111:0] enviar; //saida vira wire
12    wire [3:0] state;
13    wire [7:0] comando;
14    wire [7:0] afi;
15    wire r_w; // 0:reqb 1:wupb
16    wire [2:0] n; //valor de n
17    wire [4:0] r;
18    wire [4:0] r_ale;
19    wire [4:0] num;
20    wire [3:0] s;
21    wire [4:0] slot;
22    wire [7:0] cid;
23
24    // Instantiate the Unit Under Test (UUT)
25    mealy uut (
26       .rst(rst),
27       .clk(clk),
28       .receber(receber),
29       .enviar(enviar),
30       .state(state),
31       .comando(comando),
32       .afi(afi),
33       .r_w(r_w),
34       .n(n),
35       .r(r),
36       .r_ale(r_ale),
37       .num(num),
38       .s(s),
39       .slot(slot),
40       .cid(cid)
41    );
42
43
44    initial begin
45       // Initialize Inputs
46       rst = 1;
47       clk = 0;
48
49       #4;
50       rst = 0;
51       receber = 88'h000000000000000000000000;
52       #5;
53       receber = 88'h051100000000000000000000;
54       #5;
55       receber = 88'h053B040000000000000000; //0001:n=16
56       #5;
57       //slot marker: slot e $5 + crc1,2 onde slot:0000=nao suporta 0001=2 0010=3 0011=4 ... 1111=16
58       receber = 88'h250000000000000000000000; //0010=3
59       #5;
60       receber = 88'h950000000000000000000000; //1001=10
61       #5;
62       //receber = 88'h500000000000000000000000;  //hltb
63       //comando attrib: $1D + PUPI0,1,2,3 + PARAM1,2,3,4 + CRC1,2 , onde PARAM 4 = CID atribuido
64       receber = 88'h1d11223344000000330000;  //attrib
65       #10;
66       //receber = 88'h051109000000000000000000; //wupb
67
68    end
69
70    always begin
71          #2  clk = ~clk; // Toggle clock every 5 ticks
72    end
73
74 endmodule
75
```

Figura 53 – Test Bench do Bloco Digital da tag

ANEXO B – Código Verilog do CRC

Neste anexo se encontra o código completo em verilog e o test bench do CRC.

```
// File Name   : serial_crc.v
// Function    : CCITT Serial CRC
//-----------------------------------------------------
module crc_simp(clk, data_in, crc_out);
//-----------Input Ports---------------
input clk     ;
input data_in ;
```

```
//-----------Output Ports---------------
output [15:0] crc_out;
reg   [15:0] crc_out = 16'hFFFF;
// Logic to CRC Calculation
always @ (posedge clk) begin

      crc_out[0]  <= data_in ^ crc_out[15];
      crc_out[1]  <= crc_out[0];
      crc_out[2]  <= crc_out[1];
      crc_out[3]  <= crc_out[2];
      crc_out[4]  <= crc_out[3];
      crc_out[5]  <= crc_out[4] ^ data_in ^ crc_out[15];
      crc_out[6]  <= crc_out[5];
      crc_out[7]  <= crc_out[6];
      crc_out[8]  <= crc_out[7];
      crc_out[9]  <= crc_out[8];
      crc_out[10] <= crc_out[9];
      crc_out[11] <= crc_out[10];
      crc_out[12] <= crc_out[11] ^ data_in ^ crc_out[15];
      crc_out[13] <= crc_out[12];
      crc_out[14] <= crc_out[13];
      crc_out[15] <= crc_out[14];

end

endmodule
```

Figura 54 – Código verilog CRC

```
1  `timescale 1ns / 1ps
2
3  module crc_simp_tb;
4
5     // Inputs
6     reg clk;
7     reg data_in;
8
9     // Outputs
10    wire [15:0] crc_out;
11
12    // Instantiate the Unit Under Test (UUT)
13    crc_simp uut (
14       .clk(clk),
15       .data_in(data_in),
16       .crc_out(crc_out)
17    );
18
19    initial begin
20       // Initialize Inputs
21       clk = 0;
22       data_in = 0;
23
24    end
25
26    always begin
27          #1  clk = ~clk; // Toggle clock every 5 ticks
28    end
29
30 endmodule
31
```

Figura 55 – Test Bench CRC

```
1
2  module crc_mod(clk, data_in, crc_out);
3
4  input clk;
5
6  input data_in;
7
8  output [15:0] crc_out;
9
10 reg [15:0] crc_out = 16'hFFFF;
11
12 integer i ;
13
14 initial begin
15
16    for(i=0; i < 8; i = i + 1 ) begin
17
18       crc_out[0]  <= data_in ^ crc_out[15];
19       crc_out[1]  <= crc_out[0];
20       crc_out[2]  <= crc_out[1];
21       crc_out[3]  <= crc_out[2];
22       crc_out[4]  <= crc_out[3];
23       crc_out[5]  <= crc_out[4] ^ data_in ^ crc_out[15];
24       crc_out[6]  <= crc_out[5];
25       crc_out[7]  <= crc_out[6];
26       crc_out[8]  <= crc_out[7];
27       crc_out[9]  <= crc_out[8];
```

```
28          crc_out[10] <= crc_out[9];
29          crc_out[11] <= crc_out[10];
30          crc_out[12] <= crc_out[11] ^ data_in ^ crc_out[15];
31          crc_out[13] <= crc_out[12];
32          crc_out[14] <= crc_out[13];
33          crc_out[15] <= crc_out[14];
34
35       end  //for
36
37    end //always
38
39    endmodule
40
```

Figura 56 – Código verilog CRC utilizando o loop for

```
1   module crc_mod_tb;
2
3      // Inputs
4      reg clk;
5      reg data_in;
6
7      // Outputs
8      wire [15:0] crc_out;
9
10     // Instantiate the Unit Under Test (UUT)
11     crc_mod uut (
12        .clk(clk),
13        .data_in(data_in),
14        .crc_out(crc_out)
15     );
16
17     initial begin
18        // Initialize Inputs
19        data_in = 0;
20        clk=0;
21
22        //#8 clk=1;
23        //#8 clk=0;
24     end
25
26
27  endmodule
```

Figura 57 – Test Bench CRC utilizando o loop for