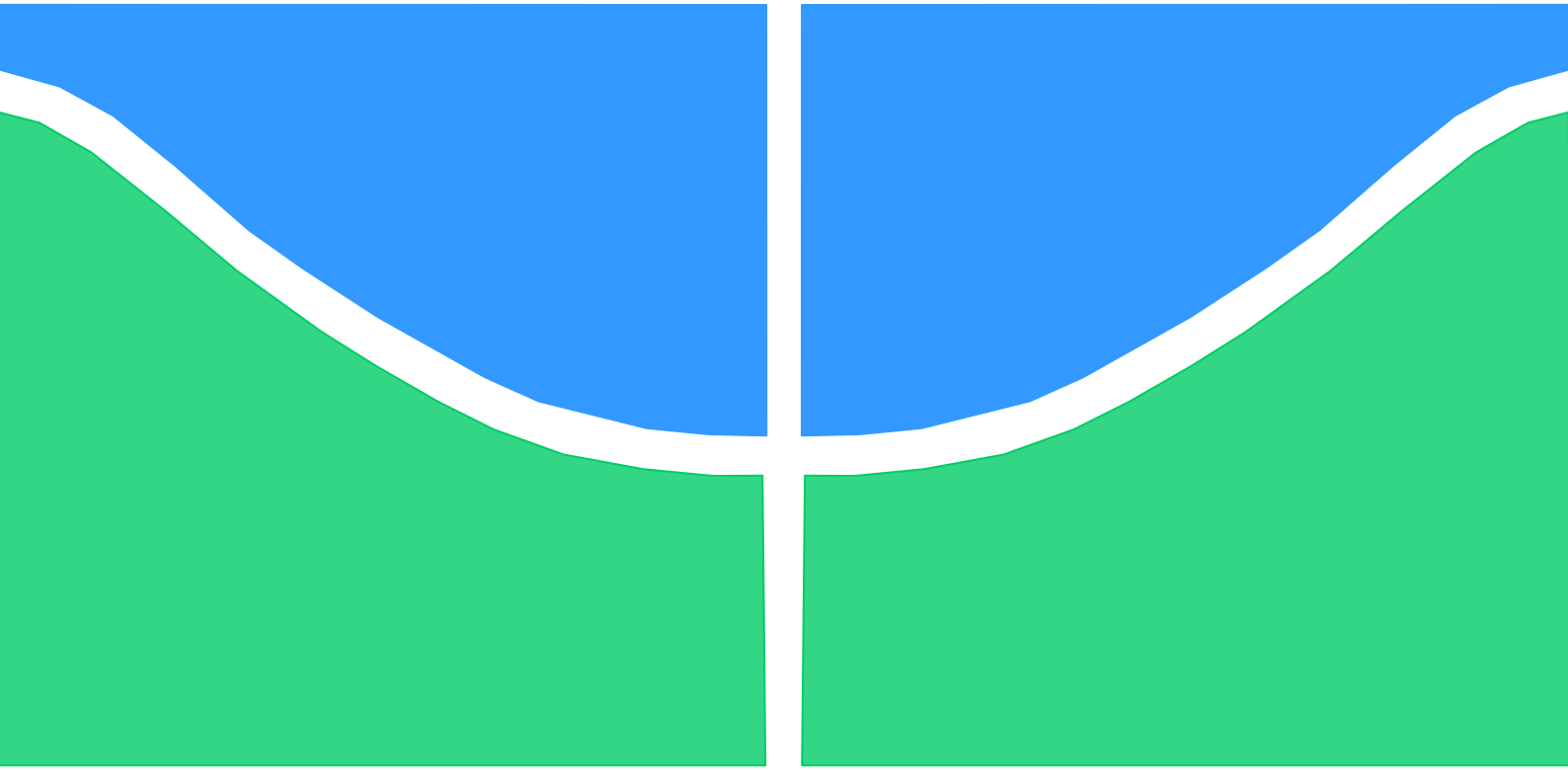


TRABALHO DE GRADUAÇÃO

CARACTERIZAÇÃO DE BLOCOS DE MEMÓRIA SRAM PARA IMPLEMENTAÇÃO DE FUNÇÕES FISICAMENTE INCLONÁVEIS

Arthur Morales Sampaio

Brasília, 23 de Agosto de 2013

UNIVERSIDADE DE BRASILIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

CARACTERIZAÇÃO DE BLOCOS DE MEMÓRIA SRAM PARA IMPLEMENTAÇÃO DE FUNÇÕES FISICAMENTE INCLONÁVEIS

Arthur Morales Sampaio

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro Eletricista

Banca Examinadora

Prof. José Edil G. de Medeiros, UnB/Departamento de Engenharia Elétrica (Orientador) _____

Prof. Sandro A. P. Haddad, UnB/ Faculdade do Gama _____

Prof. Diego F. Aranha, UnB/ Departamento de Ciência da Computação _____

Dedico este trabalho aos meus pais, por me incentivarem a persistir na conquista de meus sonhos e sobretudo me apoiarem nas decisões que tomei para torná-los realidade.

Agradecimentos

À minha mãe, a única pessoa no mundo que me conhece desde o meu primeiro suspiro. Me alimentou quando tive fome, cuidou-me quando estive enfermo, acolheu-me quando chorei, sorriu quando sorri, e sobretudo me ensinou a ser a pessoa que me tornei.

Ao meu pai, por todas as lições de vida e por ser o principal responsável pela formação de meu caráter. Pelo apoio e amor incondicionais em todos os momentos. Meu melhor amigo.

À minha companheira, Cristina, que prova a todos os dias que para o amor não existem barreiras.

Ao meu grande amigo Carlos Alberto Pacheco, por ter dividido momentos importantes comigo durante os meus estudos. Me ajudou sempre que precisei.

Aos meus tios Natan e Hilda, por terem se tornado minha segunda família no momento em que me acolheram com tanto amor e carinho nesta cidade. Serei eternamente grato.

A todos em minha família, que nunca me deixaram sentir sozinho durante esta jornada.

Aos meus amigos Rafael Martins, Ramon Martinelli, Henrique Cleto, Alan Andrade, Lucas Teixeira, Érico, e tantos outros por citar, por todas as noites que passamos em claro para tornar esse momento realidade.

Ao meu amigo, José Carlos de Oliveira, que proporcionou grande auxílio durante o desenvolvimento deste trabalho e vêm me apoiando em meu desenvolvimento profissional.

Aos Professores José Edil e José Camargo, por me guiarem desde o início da graduação. Me acompanharam na luta pela conquista do intercâmbio, me repreenderam, me ensinaram, e agora me auxiliaram a tornar um sonho realidade: Tornar-me engenheiro.

Arthur Morales Sampaio

RESUMO

Este projeto aborda a caracterização de blocos de memória SRAM para implementação de PUFS. Avaliou-se a idéia de considerar o estado power-up de células SRAM para identificar suas propriedades devido a incompatibilidades físicas. Também explorou-se o comportamento da memória através da realização de vários *power-on resets* e coleta de *dumps* de memória através de uma interface serial assíncrona. Assim, apresentou-se uma estrutura de aquisição e processamento dessas informações e mostrou-se alguns casos típicos em que o estágio de inicialização aparenta ser um forte candidato para gerar uma PUF utilizando células de memória estática em um circuito *off-the-shelf*.

ABSTRACT

This project discusses the characterization of SRAM PUFs through the development of a framework for assessing the use of SoC SRAMs as Physical Unclonable Functions. We evaluate the idea of considering the power-up state of a SRAM cell to identify its properties due to physical mismatches. We also explore the behavior of the memory by performing several power-on resets and gathering memory dumps through an asynchronous serial interface. We then provide a framework to retrieve and process this information and show a typical case where the startup stage shows itself to be a strong candidate to generate a Physical Unclonable Function using static memory cells on an off-the-shelf circuit.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ALGORITMOS

1 INTRODUÇÃO

Physically Unclonable Functions (PUFs) ou Funções Fisicamente Inclonáveis, representam um campo do conhecimento científico que emergiu recentemente [1] do desenvolvimento de protocolos criptográficos e do avanço de arquiteturas de segurança. São funções intrínsicas ao dispositivo mapeadas por pares *challenge-response*, isto é, quando solicitada com um desafio, a PUF gera uma resposta aleatória que depende das propriedades físicas do hardware em que está implementada. Como essas propriedades são sensíveis a variações típicas das condições de operação, como tensão de alimentação e temperatura, a PUF deverá retornar uma resposta ligeiramente diferente a cada vez que for estimulada, o que distancia o caso real do cenário ideal. Entretanto, por meio de uma análise estatística e implementação de códigos de erro já foi demonstrado na literatura que PUFs em SRAM [2] [3] [4] podem ser utilizadas de forma a identificar o dispositivo tornando complicado o processo de clonagem ou reprodução do seu comportamento.

Ainda que os recursos de segurança relacionados às PUFs estejam em estágio de investigação e pareçam incipientes há uma tendência de que os resultados sejam promissores. Avaliações já publicadas na literatura são difíceis de reproduzir devido às condições de teste serem bastante específicas e variarem expressivamente quanto aos métodos de análise.

1.1 DESCRIÇÃO DO PROBLEMA

Há muito se pensou que o conteúdo de memórias SRAM após a energização do circuito era puramente aleatório, mas recentemente descobriu-se que o comportamento de blocos de memória estática poderiam ser relacionados a pequenas diferenças físicas introduzidas durante o processo de fabricação do sistema [2]. O problema a ser abordado neste trabalho refere-se à caracterização de blocos de memória SRAM quanto às suas propriedades de forma a determinar sua utilidade para implementação de *Physically Unclonable Functions* (PUFs).

1.2 OBJETIVOS

1 – Caracterizar o funcionamento de blocos de memória SRAM para implementação de PUFs.

2 – Desenvolver um framework que possa ser reutilizado para caracterização de outras figuras de mérito associadas ao funcionamento de PUFs em SRAM.

2 REVISÃO BIBLIOGRÁFICA

Esse capítulo apresenta uma revisão bibliográfica dos principais conceitos utilizados na realização deste projeto, bem como, define termos que serão empregados ao longo deste trabalho de maneira a facilitar a compreensão e contextualizar o que já foi publicado na literatura a respeito dos tópicos de interesse.

2.1 PUFs – PHYSICALLY UNCLONABLE FUNCTIONS

PUFs consistem em sistemas intrinsicamente inclonáveis que fornecem uma resposta única baseada em um desafio apresentado a ela e à estrutura física do circuito utilizado para implementá-la [5]. O objetivo é adquirir uma função que seja fácil de avaliar mas difícil de previr antes de aplicar desafios a ela, o que é definido pela comunidade científica por autenticação challenge-response. Esta arquitetura consiste em uma família de protocolos onde uma das partes envolvidas na autenticação apresenta uma pergunta ("challenge") e a outra deve ser capaz de prover uma resposta válida ("response") a ser autenticada. O exemplo mais simples deste tipo de autenticação é a validação de uma senha, em que o desafio está na solicitação da senha e a resposta é provida pelo usuário inserindo a senha correta. No caso específico das PUFs quando um estímulo físico é aplicado à estrutura ela reage de forma imprevisível, mas que pode ser repetida, devido à complexa interação do estímulo com a microestrutura física do dispositivo. Esta estrutura depende de fatores físicos introduzidos durante a construção de cada dispositivo em particular.

Uma PUF quando solicitada com uma certa entrada deve ser capaz de produzir uma saída mensurável. E não é uma função em sentido matemático, já que uma entrada pode assumir mais de uma saída válida, o que teoricamente violaria a definição formal de uma função matemática. Sendo assim, é mais apropriado considerar uma PUF como uma função num sentido amplo aplicado à engenharia, por exemplo, um processo executado por, ou agindo sobre, um sistema físico particular. Tipicamente, uma entrada para uma PUF é chamada de desafio (challenge) e a saída uma resposta (response). Um desafio aplicado e a sua saída correspondente recebem, portanto o nome de CRP, do inglês, *challenge-response pair*. Em uma aplicação típica, uma PUF é utilizada em duas fases distintas, a primeira geralmente chamada de registro (*enrollment*) é quando um número de CRPs é adquirido de um dispositivo a ser utilizado como PUF e armazenado no que definimos como CRP database, uma espécie de banco de dados onde ficam armazenados os pares desafio-resposta característicos de um determinado dispositivo. Na segunda fase típica de utilização, que denominamos de verificação, um desafio proveniente do CRP database é aplicado à PUF e a resposta produzida é então comparada com a resposta contida no CRP database. Apenas a título de referência é importante mencionar que em algumas PUFs a funcionalidade challenge-response é inferida naturalmente à partir de sua construção, enquanto para outras é menos óbvio e certos parâmetros devem ser explicitamente indicados para agir como desafio [2].

Idealmente uma PUF deve apresentar resposta constante para um mesmo dispositivo e resposta necessariamente diferente para dispositivos diferentes. E desse comportamento deve-se derivar a capacidade de identificação do dispositivo por meio da PUF. Assim, é possível definir duas métricas de interesse para a análise de PUFs: medidas inter- e intra-classe [2].

- Inter-classe: Para um desafio em particular, a distância inter-classe associada a duas PUFs diferentes representa uma medida da distância entre duas respostas a um mesmo desafio a ambas as PUFs.
- Intra-classe: Para um desafio em particular, a distância intra-classe entre duas medidas em um mesmo dispositivo que esteja sendo utilizado como PUF representa a distância entre as duas respostas resultantes da aplicação deste mesmo desafio duas vezes nessa mesma PUF.

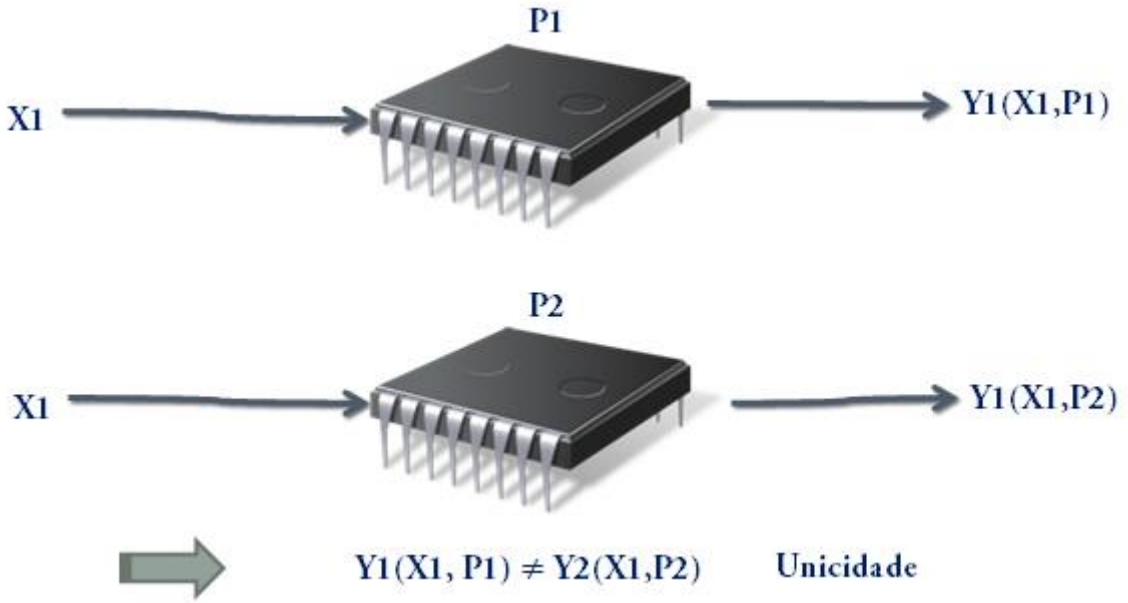


Fig. 1 – Exemplo gráfico do comportamento inter-classe de uma PUF

Tanto as distâncias inter- como intra-classe são medidas com base em um par de respostas quando estimulados por um mesmo desafio. Em muitos casos onde a resposta a um desafio é um *array* de bits, utiliza-se o conceito de distância de Hamming para mensurar quão diferente uma amostra é da outra. A distância de Hamming entre dois *arrays* de bits de mesmo tamanho é o número de posições nas quais eles diferem entre si. É bastante comum também utilizarmos uma representação percentual desta distância em relação à quantidade total de bits que a amostra possui. [6]
Exemplo: A distância de Hamming entre 10**1**1**1**010 e 10**0**1**0**010 é 2, enquanto que a distância fracional de hamming entre as duas amostras apresentadas aqui seria:

$$HD_{\%} = {}^{HD}/_{Total} = {}^{2}/_{8} = 0.25 = 25\%$$

O valor associado às distâncias inter- e intra-classe pode variar em função do desafio aplicado e da PUF envolvida. Para um tipo particular de PUF as distâncias inter- e intra-classe são geralmente caracterizadas por meio da apresentação de histogramas que mostram a ocorrência de ambos os tipos de distância, observados sob um número de diferentes desafios e um número de diferentes PUFs. Em muitos casos, ambos os histogramas (inter- e intra-classe) podem ser aproximados por uma distribuição gaussiana e são caracterizados em termos de suas médias (μ_{intra} *e* μ_{inter}), ou quando disponíveis seus desvios padrão (σ_{intra} *e* σ_{inter}). É importante notar que μ_{intra} denota uma a noção de **ruído médio** nas respostas, pois estaríamos interessados que amostras em uma mesma instância de uma PUF quando submetida a um desafio específico responda de forma imprevisível, mas repetidamente, o que em tese significaria no caso ideal que estaríamos interessados que todas as amostras aplicadas a uma mesma PUF para um mesmo desafio fossem idênticas. Sendo assim, deve-se alcançar μ_{intra} tão pequeno quanto se possa alcançar de forma a produzir uma PUF cuja resposta seja extremamente confiável. A grandeza μ_{inter} expressa uma noção de **unicidade** ao ponto que ele mensura quão diferentes são duas PUFs particulares quando submetidas a um mesmo desafio. Nesse caso é desejável que este parâmetro seja máximo, já que estaria expressando que duas PUFs diferentes quando submetidas a um desafio específico produziriam respostas distintas a ponto de possibilitar a

identificação do dispositivo em questão. É interessante perceber que a noção de minimizar as distâncias intra-classe e maximizar as métricas inter-classe traz por definição um *trade-off*, o que caracteriza portanto, uma solução de compromisso ao encontrar um ponto ótimo que satisfaça as condições necessárias à implementação de um dispositivo que responda de maneira segura e confiável e que se aproxime do caso ideal.

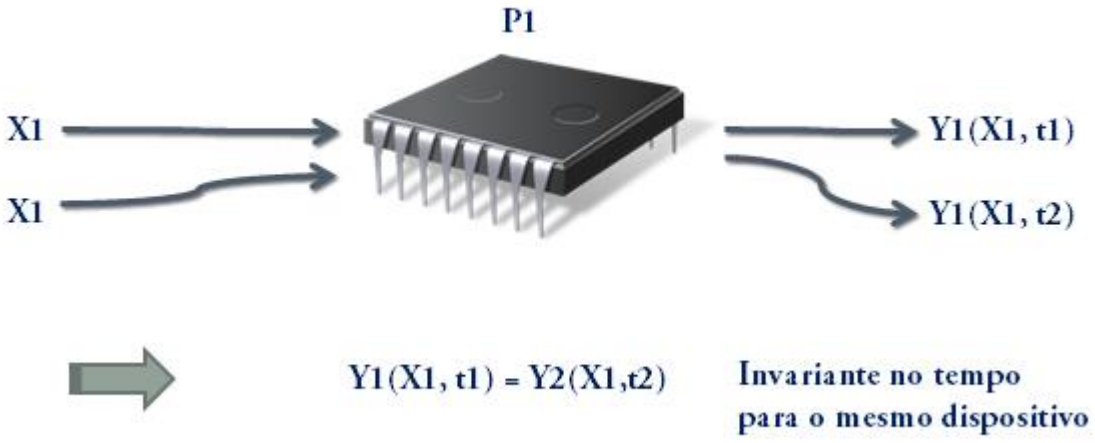


Fig. 2 – Exemplo gráfico do comportamento intra-classe

A avaliação de PUFs deve considerar a determinação da influência de parâmetros associados ao ambiente. Já foi mencionado que o mesmo desafio aplicado a uma PUF não necessariamente produzirá a mesma resposta, o que dá margem à aplicação da chamada distância intra-classe. Em uma tentativa de avaliar quais fatores podem influenciar a divergência entre valores obtidos de um mesmo dispositivo quando submetido a um estímulo particular é proposto que ruídos puramente aleatórios são introduzidos durante as medidas. Entretanto, alguns fatores relacionados ao ambiente também apresentam um efeito sistemático na medida das respostas, por exemplo, temperatura ou tensão de alimentação no caso de uma PUF em um circuito integrado. As distâncias intra-classe médias irão provavelmente aumentar quando medidas são consideradas em um ambiente variável quanto às condições ambientais. Para possibilitar uma avaliação justa entre os resultados deve-se estabelecer um *framework* capaz de controlar essas variáveis que podem influenciar nas medidas.

Fig. 3 – Diagrama exemplo da imprevisibilidade de uma PUF

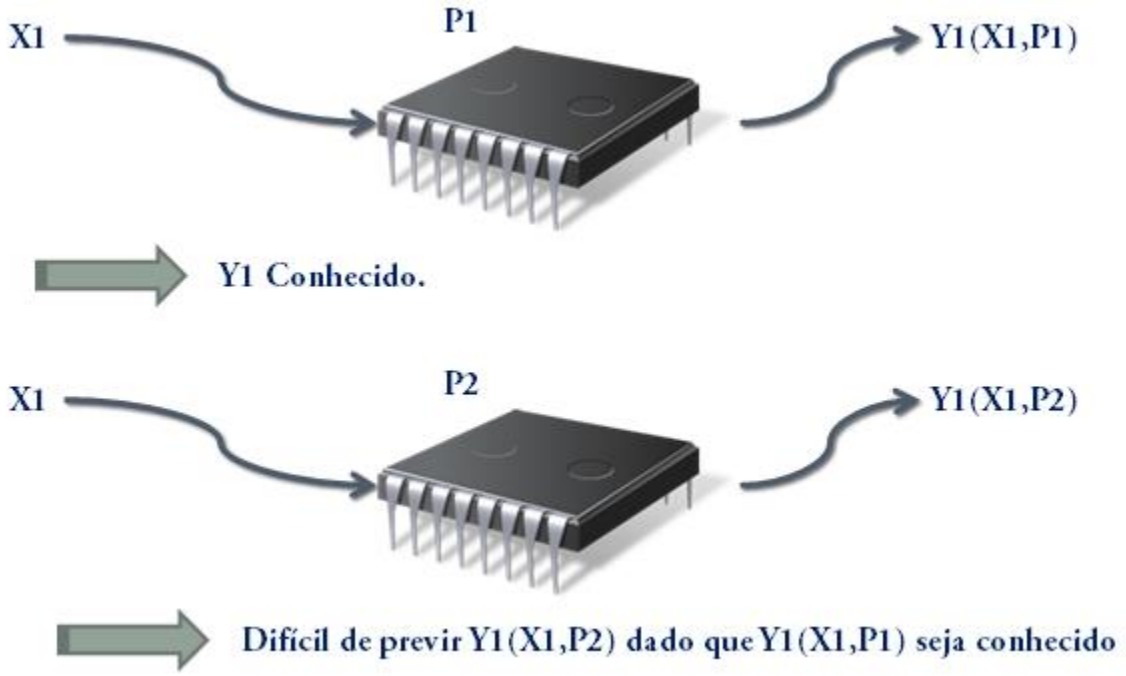


Fig. 4 – Diagrama exemplo da imprevisibilidade de uma PUF

O fato de os efeitos do ambiente serem sistemáticos possibilita a proposição de técnicas para reduzir sua influência na resposta das PUFs. Se os efeitos são parcialmente lineares e afetam o dispositivo inteiro como um todo uma proposta diferencial pode ser adotada. Considerar a relação entre duas medidas simultâneas, por exemplo a diferença ou a razão, pode proporcionar uma medida robusta o suficiente para isolar o efeito do ambiente em algumas medidas. Essa técnica foi introduzida em [7] [8] e é denominada compensação. Outra estratégia que pode ser adotada é admitir que o impacto de fatores do ambiente dependem principalmente de detalhes de implementação física de cada PUF, e dessa forma deve ser possível minimizar esses efeitos por meio da construção de um sistema robusto em termos de arquitetura do próprio hardware ou estrutura física. Essa idéia foi introduzida em [9] [10]. E por fim, outra abordagem que pode ser considerada é isolar os parâmetros de interesse por meio da inserção de um sensor que monitora as variáveis que podem ser responsáveis pela introdução de ruído aleatório no sistema e o estabelecimento de intervalos de variação que podem ser considerados aceitáveis para a validade de medidas. Essa será a estratégia utilizada neste trabalho, pois utilizaremos sensores de temperatura e de tensão para isolar essas variáveis durante as medidas.

2.2 IMPLEMENTAÇÕES DE PUFs

Existem na literatura, algumas abordagens sobre os tipos de arquiteturas que poderiam ser utilizadas como PUFs. Serão apresentadas, nessa seção, algumas dessas implementações e uma breve discussão sobre elas.

2.2.1 PUFs NÃO-ELETRÔNICAS

Em [1] [11] são propostas aplicações do que se denomina **PUFs ópticas**, que lançam mão de um dispositivo óptico que contém microestrutura construída por meio da mistura de esferas de vidro refratário microscópico (500 μm) numa superfície baseada em um polímero. Esse dispositivo é irradiado com um laser e a frente de onda emergente se torna bastante irregular devido à dispersão na superfície do material. O padrão gerado pela forma de onda é então captado por uma câmera CCD e processado digitalmente. Esse padrão da frente de onda gera um código binário que representa a

estrutura física construída. A implementação da PUF propriamente dita se dá no momento em que é definido como desafio a posição de irradiação do laser e a resposta o código gerado computacionalmente para o padrão adquirido. Nos trabalhos mencionados acima ficou demonstrado que o uso de dispositivos ópticos para a criação de uma PUF é trabalhoso e o *framework* de teste possui elevado custo de implementação. Outra implementação considerada são as PUFs em papel, que consistem em escanear uma estrutura única e aleatória de uma folha de papel. Similarmente ao caso da PUF óptica mencionada anteriormente, houve algumas abordagens na literatura como em [12] [13] até mesmo antes da introdução do conceito de PUF ser desenvolvido. Em [14] a reflexão de um feixe de laser emitido sobre uma estrutura de fibra de um documento de papel foi utilizado como uma espécie de impressão digital do documento para prevenir fraudes, o que não deixa de ser uma proposta de identificação por meio de uma propriedade física intrínseca ao material.

Em [15] foi proposto que os comprimentos de estruturas gravadas em CDs (compact disks) comuns apresentariam um desvio aleatório dos comprimentos pretendidos pelo fabricante durante a manufatura. Foi assim, empregada uma técnica para monitorar o sinal elétrico do fotodetector que lê o CD de forma a captar essas discrepâncias devidas ao processo de fabricação. Em [16] utiliza-se a unicidade inerente dos padrões de partículas em mídias magnéticas, como cartões com trilha magnética. Em [17] esses padrões são utilizados de maneira a prevenir fraudes de cartões de crédito. Existe também na literatura, [18], uma aplicação para implementação de uma PUF por meio de componentes acústicos chamados linhas de atraso, que baseiam-se na conversão de um sinal elétrico para uma vibração mecânica e a sua captação e conversão para o universo elétrico novamente. Isso introduz um atraso característico no sistema. Essa propriedade é utilizada de forma a gerar um *array* de bits que identifica a estrutura física.

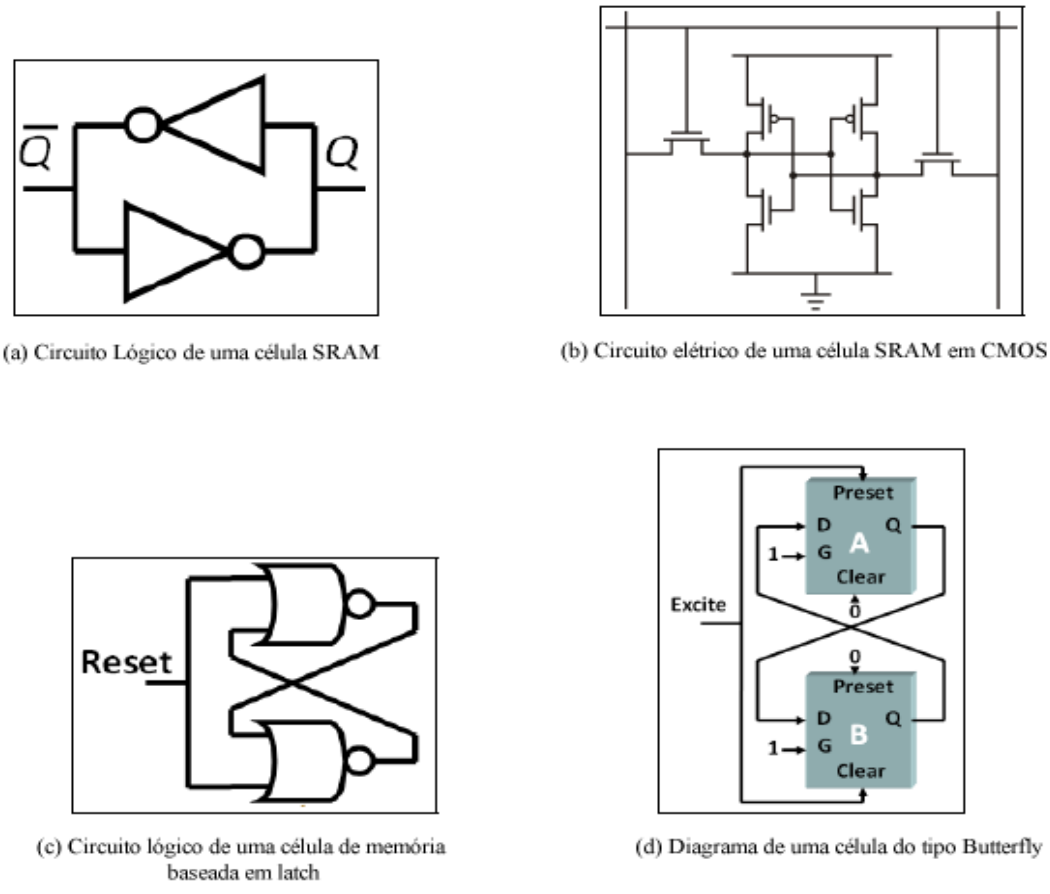
2.2.1 PUFs ELETRÔNICAS

Uma construção chamada RF-DNA (Radio-frequency-DNA) foi proposta em [19]. Foi construída uma pequena estrutura, onde condutores são posicionados de maneira aleatória sobre um isolante de silicone. Neste caso, os autores se concentraram em avaliar a distribuição de ondas eletromagnéticas sobre os condutores de cobre em frequências da ordem de 5 – 6 GHz. Os efeitos aleatórios de espalhamento do campo sobre a superfície são medidos por meio de antenas RF. Esses dados são então processados de forma a implementar uma proposta para o que seria uma PUF baseada na característica de espalhamento de campo magnético sobre essa estrutura. É conhecido também que um campo promissor é avaliar propriedades intrínsecas de circuitos analógicos como a tensão Vt de transistores, a distribuição de potência em circuitos integrados, frequência de ressonância de circuitos LC, atraso característico de estruturas implementadas em silício, de forma a identificar a variação aleatória destes parâmetros durante o processo de manufatura para que a estrutura possa atuar como uma PUF. Trabalhos de referência nesse sentido podem ser encontrados em [20] [21] [22] [23] [24] [25] [26].

O trabalho [7] introduz o princípio de utilizar osciladores em anel como PUFs. É sabido que por mais preciso que seja o processo de fabricação destes componentes sempre há uma variação aleatória entre as respostas de circuitos osciladores. A vantagem desta abordagem se encontra no fato de que medidas de frequência são relativamente fáceis de ser implementadas por meio do uso de componentes digitais como um detector de bordas de um sinal que oscila e um contador para definir quantas vezes o sinal mudou de estado durante um certo período de tempo. Essas variações na frequência de saída do oscilador podem ser interpretadas como um parâmetro que auxilia na identificação do dispositivo quanto às suas propriedades físicas e consequentemente representa uma estratégia de implementação de uma PUF.

Outra implementação abordada para PUFs utiliza uma célula de memória digital, que é tipicamente um circuito com mais de um estado lógico possível. Ao passo que a célula mantém-se em um estado estável ela pode armazenar informação digital. Todavia, se o elemento é levado ao um estado instável não é tão claro qual comportamento será verificado. A célula pode oscilar entre estados instáveis ou pode convergir para um dos estados possíveis. Essa noção de variabilidade é explicada

não somente pela implementação da célula, mas também por discrepâncias físicas introduzidas nos circuitos durante o processo de manufatura. Por este motivo, admite-se que o estado inicial de uma memória logo após a energização do circuito é um bom candidato a fornecer uma resposta que satisfaça às condições requisitadas para que o sistema opere como uma PUF. Em [5] é apresentada uma proposta de implementação de PUF por meio de um circuito que utiliza uma célula SRAM (Static Random Access Memory). Em [27] é apresentada uma arquitetura chamada de Butterfly PUF cell, que é uma arquitetura diferente do proposto anteriormente, pois em FPGAs a implementação de células de memória não deve ser uma estratégia muito promissora, já que em FPGAs a célula de memória é geralmente resetada para zero logo após o estágio de power-up, o que eliminaria a variabilidade estatística presente no estado inicial dessas células de memória. Essa arquitetura Butterfly lança mão do acoplamento cruzado de dois latches de forma a introduzir dois estados estáveis possíveis na lógica da célula de memória. Isso se compara ao avaliado em uma célula SRAM comum, todavia sem a necessidade de desligar o circuito e liga-lo novamente para verificar o estado estável para o qual a célula será levada. Em [28] também é mencionada uma estrutura para uma PUF utilizando uma célula de memória, todavia baseada em um latch digital, o que é bastante similar às PUFs em SRAM e Butterfly, todavia lançando mão de uma estrutura digital simples e avaliando o estado inicial para o qual a célula caminha durante o estágio power-up.



(a) Circuito Lógico de uma célula SRAM

(b) Circuito elétrico de uma célula SRAM em CMOS

(c) Circuito lógico de uma célula de memória baseada em latch

(d) Diagrama de uma célula do tipo Butterfly

Fig. 5 – Comparação entre diferentes implementações de PUFs baseadas em células de memória

2.3 PUFs EM SRAM

Como mencionado anteriormente, uma estratégia promissora no desenvolvimento de PUFs é a implementação em células de memória. PUFs baseadas em SRAM oferecem uma solução bastante flexível para implementação, pois elas podem ser reprogramadas em campo [5]. Esta alternativa já tem ganhado força no mercado de circuitos integrados. Recentemente o fabricante NXP semiconductors, sediada em Eindhoven, que surgiu da divisão de semicondutores da Philips publicou um documento [29] que direciona a utilização de PUFs em SRAM em seus produtos baseados em Smart Card e RFID. Existe ainda, outra companhia que investiu no conceito chamada Verayo, que já utiliza PUFs para implementar o que ela define por Silicon DNA™. Outro exemplo de companhia que já utiliza este conceito no mercado é a, também baseada em Eindhoven, Intrisic-Id.

O presente trabalho surgiu da iniciativa da Intel Labs University Research Office em desenvolver este campo do conhecimento em parceria com a Universidade de Brasília. A estratégia de abordagem de PUFs em SRAM vem sido estressada por institutos de pesquisa, universidades e até mesmo empresas do setor privado. Em [3] são testados diferentes tipos de implementações de PUFs e é concluído que as estratégias mais promissoras seriam a implementação em SRAM ou osciladores em anel. É de interesse deste trabalho, portanto, validar as estratégias de verificação, caracterização e implementação de PUFs em circuitos que são utilizados em grande escala no mercado atual. Dessa forma foi escolhido como escopo principal para este trabalho o estudo, implementação e verificação de PUFs em SRAM.

Uma SRAM consiste em um tipo de memória implementada em material semicondutor utilizando arquitetura CMOS. Diferentemente da RAM dinâmica (DRAM) que possui por característica preponderante a necessidade de atualização periódica dos valores ali armazenados a SRAM (Static Random-Access Memory) é baseada em latches bi-estáveis que manterão seu valor enquanto o circuito permaneça energizado. Cada bit armazenado consiste em 6 transistores organizados em duas topologias inversoras acopladas como mostrado na Fig. 6.

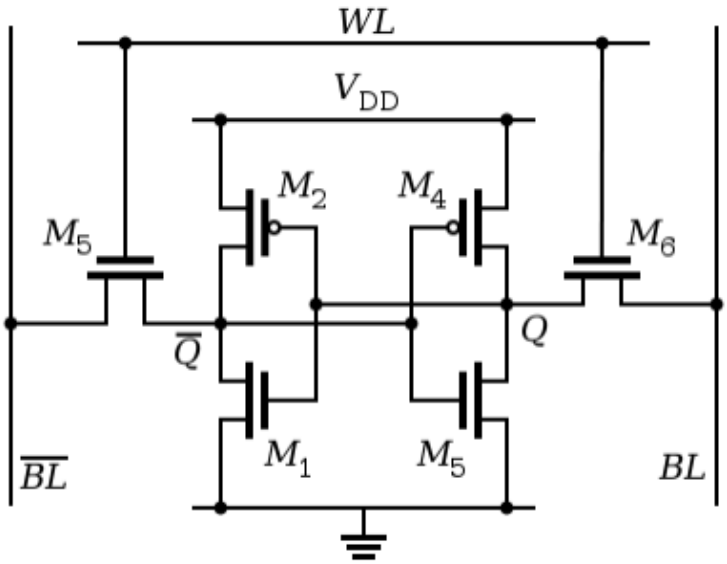


Fig. 6 – Célula de Memória SRAM utilizando 6 transistores, comumente denominada 6T-SRAM **[30]**

O bit armazenado na memória possui dois estados estáveis, os quais representam o zero lógico ou o um lógico. O estado para o qual a célula caminha logo após a energização do circuito depende de pequenas diferenças físicas durante a construção do circuito em silício, pois já é sabido que é praticamente impossível construir dois transistores fisicamente idênticos. Quanto mais uniforme for a distribuição das tensões nos componentes do circuito maior influência deste "ruído" físico será percebido durante medições efetuadas logo após o circuito ser ligado. Essas considerações conduzem à hipótese de que deve ser possível avaliar o comportamento de um bloco de memória SRAM que contém, no mínimo, milhares de células como a exemplificada Gquanto ao seu comportante durante o *startup*. De fato, essa é uma técnica já discutida na literatura, e é chamada de *Physical Fingerprinting*, pois representa uma espécie de impressão digital do circuito [31]. Variações no processo de manufatura ocorrem de muitas formas diferentes, incluindo imprecisões durante o processo de litografia do circuito integrado, o que termina por produzir variações no tamanho dos transistores e das tensões de limiar V_T (Threshold) características de cada semicondutor. Em [31] nota-se a preferência pela caracterização dos circuitos quanto à identificação por meio de divergências nas tensões de limiar (V_T) pelo fato de variações na litografia do CI serem espacialmente correlacionadas, ou seja, células vizinhas tenderiam a apresentar variações litográficas similares, o que não ocorre com as tensões, que via de regra apresentam variações aleatórias em um circuito dessa natureza.

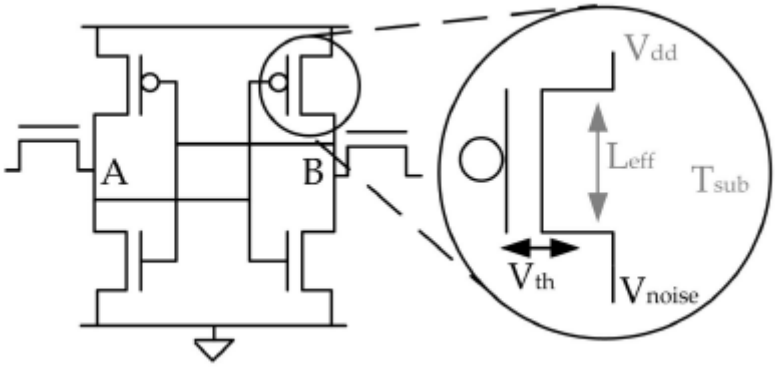


Fig. 7 – Célula SRAM apresentando variações e ruído durante o processo de fabricação

De posse do conceito da introdução de ruído durante o processo de fabricação de células SRAM é importante mencionar que cada um dos inversores é responsável por alimentar um dos nós A ou B na Fig. 7. Quando o circuito encontra-se desenergizado ambos os nós estão descarregados (AB="00"). Quando o sistema é ligado, esse estado instável irá transitar para um dos estados estáveis possíveis, seja "0" (AB = 01) ou "1" (AB = 10). O estado AB = 11 é instável e inalcançável [31]. Essa característica durante o *startup* de memórias SRAM sugere que esta estratégia possa representar uma forte candidata para a implementação de uma PUF. É importante, todavia, que sejamos capazes de demonstrar que existe aleatoriedade suficiente para identificar o dispositivo e ainda que o comportamento de um bloco de memória logo após o sistema ser energizado não seja previsível, mas que ele de alguma forma possa ser reproduzido de maneira confiável.

3 METODOLOGIA

Para desenvolvimento deste trabalho realizou-se uma pesquisa sobre a definição de PUFs e suas principais aplicações, o que levou à escolha de um tipo específico de PUF para ser avaliado neste trabalho. A avaliação das propriedades de blocos de memória SRAM de forma a caracterizá-los quanto à possibilidade de utilização como PUFs é parte integrante do foco deste manuscrito. Vale lembrar que este projeto representa a primeira iniciativa do Departamento de Engenharia Elétrica da Universidade de Brasília em pesquisar a utilização de PUFs e por ser parte integrante de um trabalho de pesquisa que objetiva a implementação de PUFs em SoCs (*System-On-Chip*) julgou-se pertinente utilizar uma estrutura que pudesse reproduzir de certa forma as condições de utilização em ambiente de produção de um SoC.

Decidiu-se isolar os problemas em partes mais simples e para tal, implementou-se o conceito em uma SRAM desenvolvida em um FPGA (*Field Programmable Gate Array*), todavia percebeu-se que em um ambiente real onde SoCs são realmente utilizados o hardware não apresenta as mesmas caracterísitcas que em um FPGA, onde as interfaces podem ser totalmente reconfiguradas de modo a atribuir nova funcionalidade ao bloco, muito pelo contrário, as interfaces em um SoC são bem definidas e padronizadas e visam otimização de custo, potência e processamento. Isso motivou a escolha de um circuito *off-the-shelf* de forma a avaliar a implementação de PUFs em SRAM, o que representaria um cenário similar ao de um SoC. O microcontrolador MSP430 da Texas Instruments foi escolhido, pois representa uma arquitetura simples que possui um bloco de SRAM de 128 Bytes e instrumentos de controle das variáveis de ambiente mencionadas anteriormente por meio de um conversor analógico-digital Sigma-Delta de 16 bits.

Dessas considerações surge a necessidade de um arranjo experimental capaz de proporcionar informações sobre a viabilidade de aplicação destes dispositivos para implementar PUFs. Será proposto neste trabalho como objetivo secundário o desenvolvimento de um *framework* capaz de mensurar o comportamento de um dispositivo em particular quando submetido a um mesmo desafio diversas vezes, o que chama-se de distância intra-classe entre as medidas, e também capaz de fornecer informações de variação característica de diferentes dispositivos quando submetidos a um mesmo *set* de experimentos, o que é conhecido por distância inter-classe entre as medidas. Embora o *framework* aqui proposto seja capaz de validar o comportamento de dispositivos quanto às variabilidades intra- e inter-classe especifica-se aqui que o objeto principal de estudo deste trabalho é a caracterização intra-classe de uma série específica de memórias SRAM presente na família de microcontroladores MSP430 da Texas Instruments. E para tal será desenvolvido aqui um sistema de aquisição de dados de memória logo após a energização do circuito. A plataforma apresentada é capaz de repetir o processo quantas vezes forem necessárias e controlar as variáveis externas, consideradas pertinentes – temperatura e tensão de alimentação – de modo a produzir dados estatísticos que demonstrem e caracterizem as propriedades necessárias para uma PUF em SRAM.

Por fim, experimentos foram realizados por meio da variação dos parâmetros mencionados anteriormente e repetições da ordem de 1000 leituras de memória consecutivas, temperatura e tensão de alimentação (Vcc). A escolha deste valor (1000) para as leituras consecutivas de memória é justificada na seção 5.1. Os resultados destes ensaios são apresentados em gráficos que expõem a distribuição estatística das distâncias de Hamming, de modo a validar a aplicação de um modelo estatístico que represente o comportamento de memórias SRAM quando aplicadas ao contexto de *PUFs*.

4 DESENVOLVIMENTO DO FRAMEWORK

Este capítulo descreve a construção do framework de testes utilizado neste trabalho. Apresenta um fluxograma completo sobre a estratégia de medidas e discorre sobre detalhes de implementação. É demonstrado também o desenvolvimento de um protocolo serial assíncrono para comunicação entre o hardware e o computador, bem como as implementações dos softwares nos periféricos envolvidos no teste.

4.1 O FRAMEWORK

A caracterização de PUFs em SRAM baseia-se no fato de que a memória após o *startup* do circuito deve ser lida de maneira segura, ou seja, sem que haja perda de informação e que os dados sejam confiáveis. Embora testes com outras famílias de dispositivos não tenham sido conduzidos neste estágio do projeto, em trabalhos futuros não se deve perceber grandes dificuldades em aplicar os mesmos conceitos aqui demonstrados em outras arquiteturas baseadas em outros processadores. É sabido também que alguns microcontroladores não possuem blocos de comunicação serial implementados em hardware e ainda que a maioria dos controladores presentes no mercado apresente esse recurso foi uma decisão de projeto utilizar um microcontrolador que não possuísse interface de comunicação UART nativa em hardware, o que obrigaria a implementação de um protocolo simples de comunicação entre a placa e um software de aquisição e processamento de dados que seria desenvolvido posteriormente. O benefício atrelado a essa escolha vem do fato de o protocolo ser implementado pelo software em execução na unidade controladora, o que garante total controle sobre todas as variáveis incluídas no processo, como por exemplo o envio de cada bit sem que haja necessidade de sincronização por meio de um pino externo, tampouco a utilização de alguma codificação digital (*e.g* Manchester) para que os dados sejam recebidos corretamente. A Fig. 8 demonstra as interfaces de comunicação entre o computador que executa um software que também foi desenvolvido de maneira a implementar o protocolo assíncrono para coletar e analisar as informações do bloco de memória.

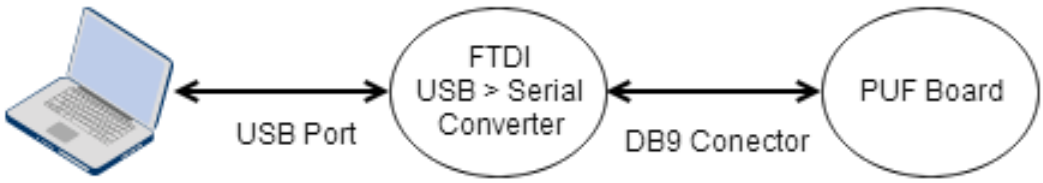


Fig. 8 – Interface de comunicação entre computador rodando software de análise e placa de avaliação da PUF em SRAM

Um microcontrolador atua como um computador em único chip. Contém processador, memória e periféricos de entrada e saída. É programado para funções específicas por meio de código que fica armazenado geralmente em memória não volátil e é comumente utilizado em produtos industrializados de maneira a controlar ações e funções. Neste projeto é empregado um microcontrolador específico da Texas Instruments: MSP430F2013. O processador desta unidade controladora é baseado na arquitetura RISC de 16 bits, do inglês **Reduced Instruction Set Computer** ou **Computador com um Conjunto Reduzido de Instruções (RISC)**, que é uma linha de arquitetura de processadores que favorece um conjunto simples e pequeno de instruções que levam aproximadamente a mesma quantidade de tempo para serem executadas. Esse microcontrolador possui um conjunto de 51 instruções (27 implementadas em hardware e 24 emuladas) e um total de 16 registradores de 16 bits. Uma característica importante a ser mencionada sobre este dispositivo microcontrolador é que a memória de dados, ou seja, a memória que o sistema pode armazenar variáveis e pilha do programa compilado possui 128 Bytes, o que significa, 1024 bits disponíveis para a utilização como SRAM-PUF caso o programa carregado na memória seja escrito de maneira otimizada para que não ocupe nenhuma posição de memória durante a avaliação da PUF. Isso significa que o software a ser embarcado no dispositivo deve fazer considerações de otimização de forma que nenhuma posição de memória seja utilizada seja por alguma variável, alguma chamada de função ou rotina de tratamento de interrupção, antes que toda informação de memória *startup* tenha sido lida e devidamente transmitida, pois assim o sistema é capaz de garantir confiabilidade às medidas a serem realizadas.

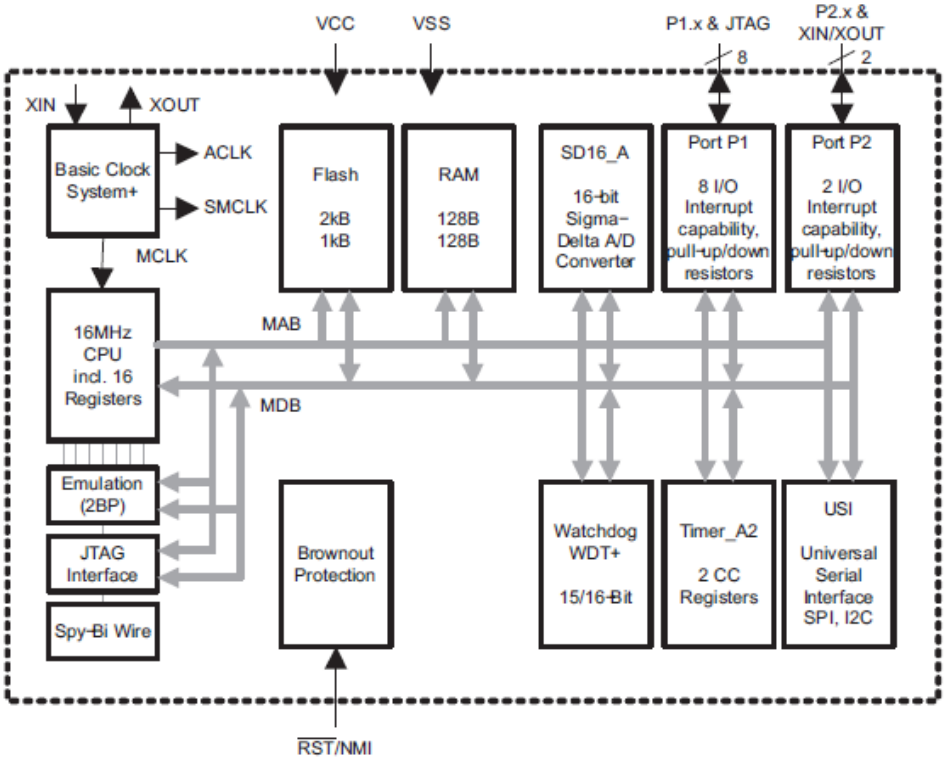


Fig. 9 – Diagrama Funcional do microcontrolador MSP430F2013 **[32]**

Na Fig. 9 apresenta-se o diagrama funcional desse microcontrolador. Um ponto importante a ser mencionado é a ausência de uma interface UART (Universal Asynchronous Receiver/Transmitter padrão implementada em Hardware. Há, entretanto, a interface USI – *Universal Serial Interface*, que também poderia ser utilizada para comunicação utilizando os protocolos *SPI* ou *I2C*, todavia decidiu-se utilizar um protocolo assíncrono baseado em *Handshaking* pela simplicidade de implementação e

controle. Uma vantagem imediata dessa escolha vem da transmissão de dados sem utilização de interrupções e isso auxiliará na programação utilizando o mínimo espaço possível da pilha (*_stack*) gerada pelo compilador. Para facilitar a compreensão desse conceito é interessante verificar a distribuição de memória neste microcontrolador em específico e então realizar as considerações devidas a respeito de suas limitações e vantagens [33] (**Fig. 10**).

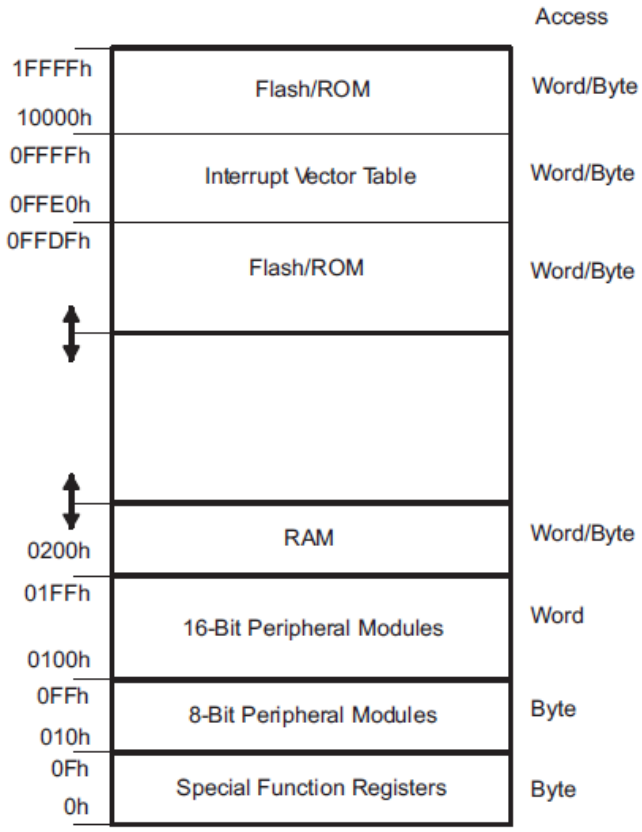


Fig. 10 – Mapeamento dos endereços de memória dos microcontroladores da família MSP430X2XXX **[33]**

É interessante verificar que nesta arquitetura a memória RAM é utilizada para gravação de dados em tempo de execução. Sendo assim, o programa que será executado não fica armazenado nessas posições de memória. Das informações obtidas dos *datasheets* fornecidos pelo fabricante dessa família de microcontroladores consta que o tipo de memória utilizado aí é SRAM e que nessa arquitetura especificamente a memória de dados inicia no endereço 0x200h e por possuir 128 Bytes infere-se que o último endereço a ser aproveitado na construção de uma PUF por meio deste dispositivo deverá ser o endereço *0x27Fh*.

Outro recurso que será utilizado aqui de forma a auxiliar na medição de algumas variáveis do ambiente é o conversor A/D Sigma-Delta de 16 bits também presente neste microcontrolador. O conversor presente no microcontrolador utilizado neste trabalho possui uma série de recursos interessantes que serão aproveitados de forma a controlar a medição da temperatura e da tensão de alimentação. É importante mencionar que para efetuar medições de temperatura, de acordo com a topologia do conversor exemplificada na Fig. 11, é necessário informar por meio do registrador SD16INCHx o valor binário 110 que representa a seleção do canal A6 para a entrada do conversor SD16. Já no caso em que o interesse seja medição de tensão de alimentação (Vcc), de acordo com o mesmo diagrama, é necessário informar por meio do registrador SD16INCHx o valor binário 101 que representa a seleção do canal A5 para a entrada do conversor.

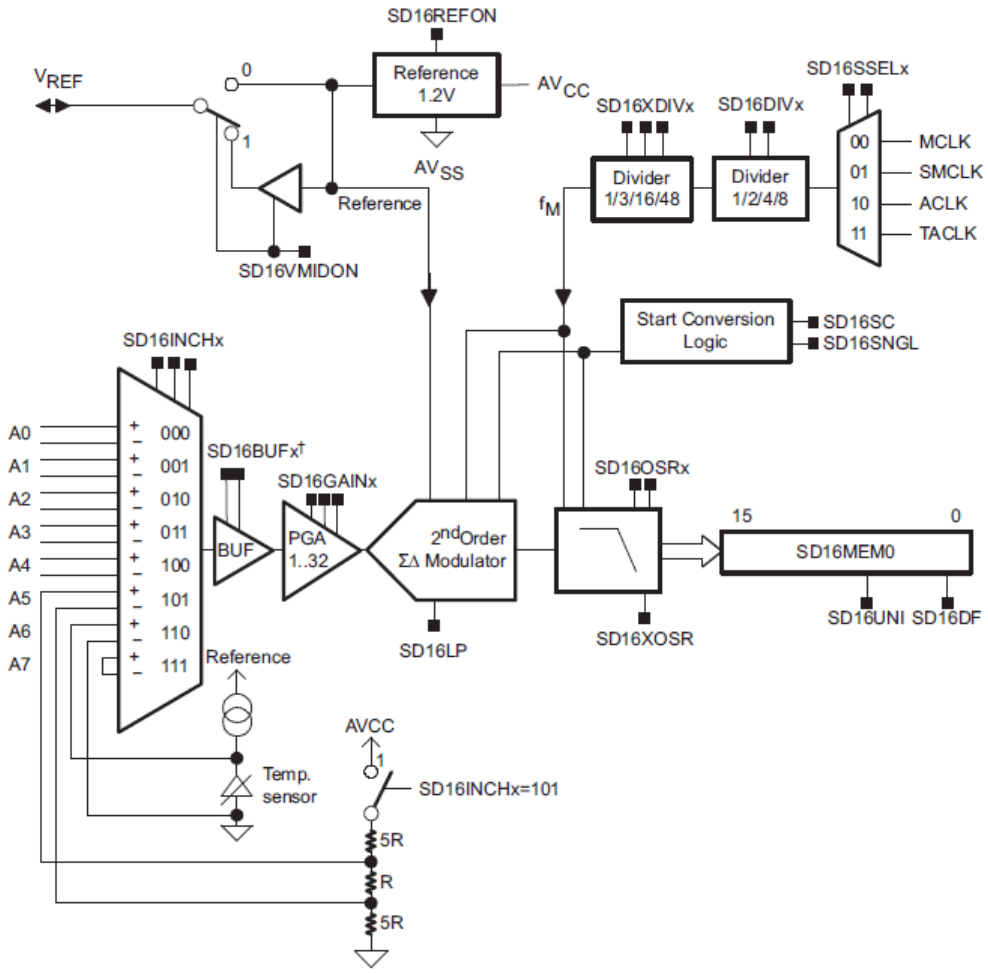


Fig. 11 – Conversor analógico-digital Sigma-Delta presente no microcontrolador MSP430F2013

De posse da descrição dos periféricos que serão utilizados durante as medidas e do entendimento da distribuição das informações de memória no hardware do microcontrolador é interessante apresentar um fluxograma de como o processo será conduzido (Fig. 12). Inicialmente a placa que contém a memória SRAM é energizada. O sistema identifica qual o endereço de início da memória de dados e efetua a leitura da memória *startup* do sistema. Esses dados são enviados para o computador conectado à placa por meio de uma interface serial assíncrona. O sistema então aciona o sensor de temperatura, envia esse valor em dois ciclos de comunicação, pois sua representação é um inteiro de 16 bits e um ciclo de comunicação é capaz de enviar um byte apenas. Só então o divisor de tensão é acionado para aferir a tensão de alimentação (Vcc) da placa e envia o seu valor da mesma forma por meio da inferface serial.

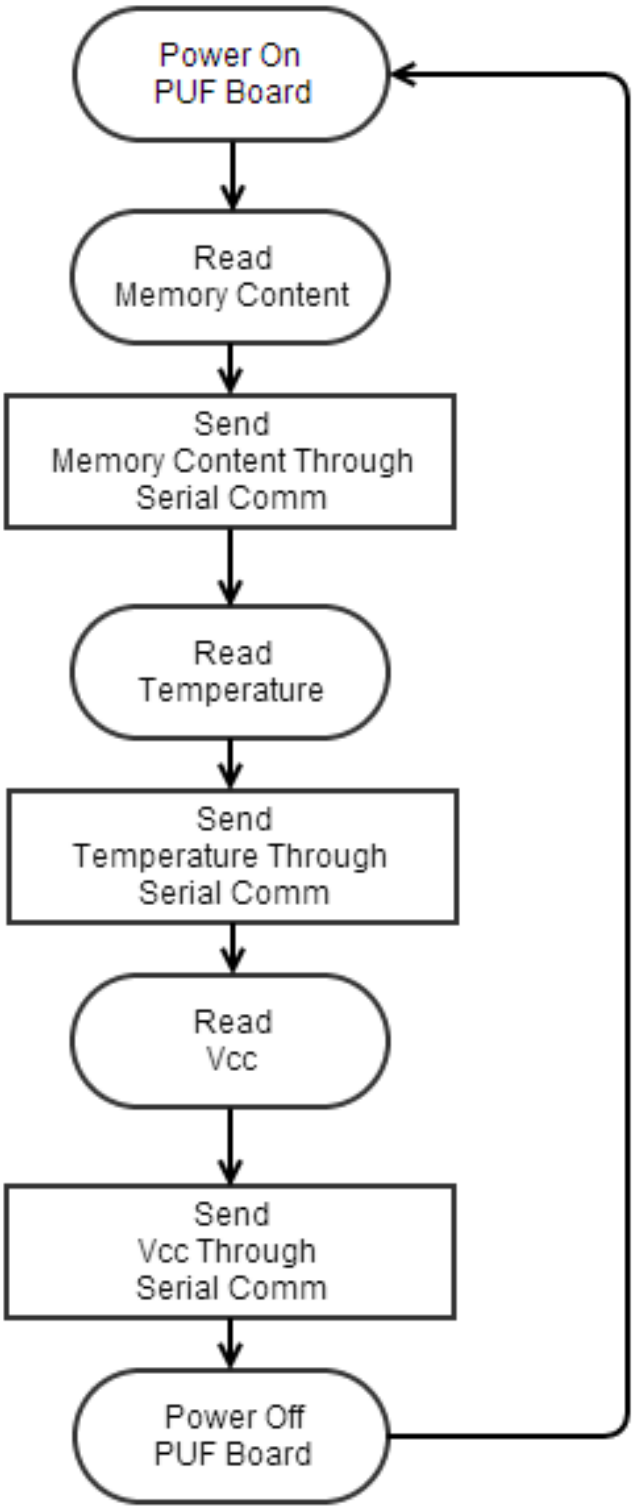


Fig. 12 – Diagrama funcional das medidas realizadas pelo *framework*

O pacote de informações que o sistema obtém a cada medição é composto por 128 Bytes que representam o conteúdo inicial da memória após energização do circuito, 2 Bytes para leituras de temperatura e 2 Bytes para leituras de Vcc, ambos representados em little endian, isto é, a informação que vem primeiro no pacote é o Byte menos significativo do valor lido tanto para temperatura quando para tensão de alimentação. Uma representação gráfica do frame recebido da placa pode ser verificado na Fig. 13.

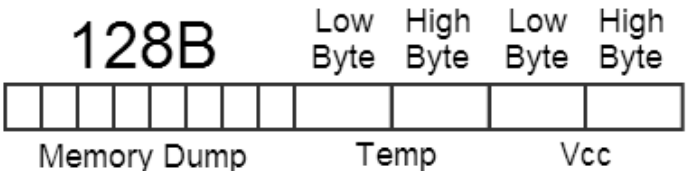


Fig. 13 – Formato do pacotes de dados recebido pelo *framework*

4.2 PROTOCOLO SERIAL ASSÍNCRONO

É bastante comum em engenharia que a comunicação entre dispositivos seja feita de maneira serial. O escopo deste trabalho não é prover informações detalhadas sobre a definição de comunicação serial, todavia é importante mencionar que este é um tipo de comunicação digital que consiste em enviar informação bit a bit, sequencialmente, em um canal ou barramento. Como discutido anteriormente, decidiu-se utilizar neste *framework* um protocolo de comunicação serial assíncrona implementado diretamente no microcontrolador. Isso permitirá algumas vantagens, como controle instantâneo dos bits sendo enviados no canal, e implementação direta na placa sem utilizar interrupções, nem consumir memória de dados do microcontrolador, pois as únicas variáveis que serão instaciadas no programa poderão ser armazenadas em registradores internos do processador, o que não prejudica a memória de dados, que estará sendo utilizada para o propósito de avaliação das características da PUF. O motivo pelo qual isso é considerado uma vantagem durante a implementação será discutido quando a arquitetura e o desenvolvimento do software embarcado no microcontrolador for apresentado.

A idéia que embasa a implementação deste protocolo é simples, todavia bastante eficaz. Suponha a seguinte situação: Um sistema embarcado deseja enviar um byte genérico para o computador. O protocolo proposto necessita de três linhas para efetuar a comunicação. O primeiro pino, representa a linha *Data* onde os bits sendo transmitidos são enviados efetivamente, o segundo, por sua vez, representa uma linha chamada *DataAV*, uma abreviação para *Data Available*, que nada mais é do que um pino que deve ser setado para indicar que há informação para ser lida na linha *Data*. O terceito e último pino, completa a arquitetura *Handshake* e se chama *DataRead*, pois representa uma linha a ser utilizada pelo destinatário para informar que a informação foi lida.

Sob o ponto de vista do microcontrolador, que nesta aplicação, representa a entidade que envia informação o protocolo requer os seguintes passos:

i) MCU seta o bit a ser enviado na linha de comunicação (*Data*).
ii) MCU seta o pino de controle que indica que o bit de dados está disponível para leitura (*DataAV* = 1).
iii) MCU aguarda por um sinal no outro pino de controle, *DataRead* = 1, que indica que o bit foi lido pelo destinatário.
iv) MCU reseta o pino de controle, *DataAV* = 0, para indicar que foi notificado e está ciente que o bit já foi lido.
v) MCU aguarda que o destinatário resete o pino *DataRead*, pois isso é a confirmação de que o destinatário sabe que o ciclo foi encerrado e que novo bit pode ser transmitido.

Sob o ponto de vista do destinatário a lógica é analogamente igual:

i) PC aguarda pelo sinal no pino de controle que indica que há dado disponível, *DataAV*=1.
ii) PC lê o dado da linha *Data*/
iii) PC seta o pino de controle que indica que o dado foi lido (*DataRead* = 1)
iv) PC aguarda que o pino DataAV seja resetado indicando que o microcontrolador está ciente que o bit já foi lido.
v) PC reseta o pino *DataRead* para confirmar que está ciente de que o ciclo de comunicação terminou.

Na Fig. 14 é mostrado um diagrama de tempo dos sinais envolvidos na comunicação.

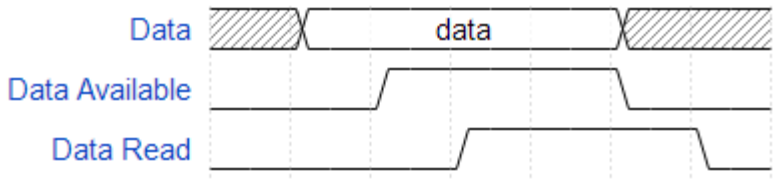


Fig. 14 – Diagrama de tempo dos sinais envolvidos na implementação do protocolo serial assíncrono.

Dessa maneira, o *framework* deve seguir o fluxo de estados para implementar a plataforma de testes capaz de receber informações sobre a memória do microcontrolador (Fig. 15).

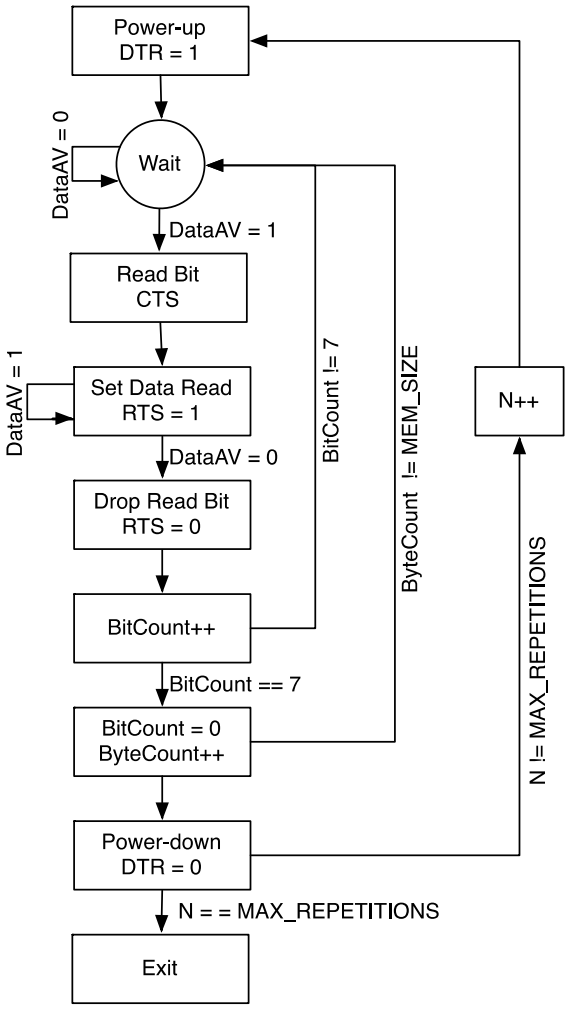


Fig. 15 – Fluxo de execução do framework de aquisição de dados da memória SRAM.

4.3 IMPLEMENTAÇÃO

Uma descrição mais específica de quais pinos do microcontrolador são conectados ao conversor USB-Serial FTDI é fornecida no diagrama da Fig. 16. Note que todos os pinos de comunicação utilizam um diodo diretamente polarizado no sentido do destino da informação, por exemplo, os pinos P1.0 e P1.1 representam respectivamente as linhas *Data* e *DataAV* que são por onde os bits de dados e o sinal *Data Available* fluem do microcontrolador para o conector serial pelo pino CTS e DSR e por isso o diodo está polarizado diretamente nesse sentido. Já no caso dos sinais *Data Read* e *Vcc* o sentido é inverso, pois o fluxo de informação é proveniente do computador.

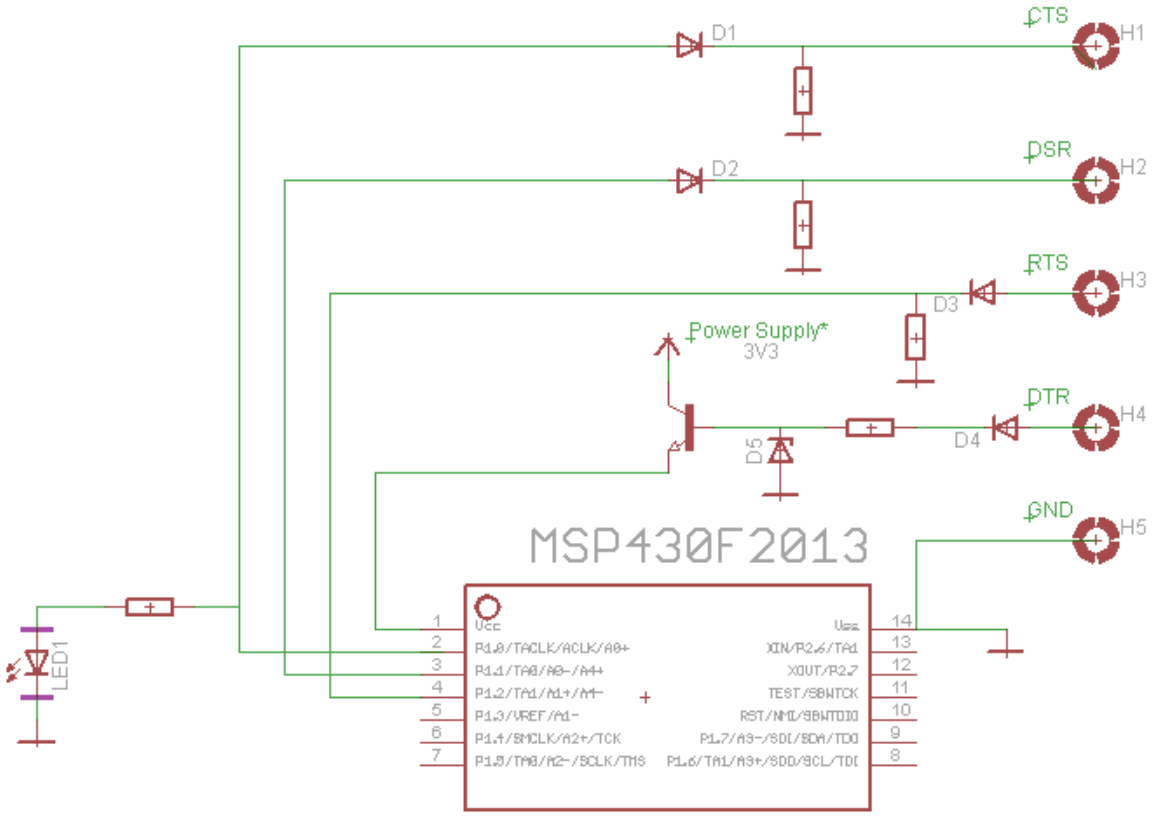


Fig. 16 – Esquema de conexões dos pinos do microcontrolador para o conector serial.

Considerou-se pertinente visualizar a arquitetura do sistema de duas formas distintas. A primeira delas relacionada à implementação do software embarcado no microcontrolador que deve ser responsável por garantir a leitura da informação pós-*startup* do circuito sem que essas informações sejam corrompidas, transmitir esses dados para a plataforma de aquisição de dados, efetuar medições de tensão de alimentação e temperatura de forma a proporcionar a capacidade de controle dessas variáveis. Do outro lado, visualiza-se a plataforma de aquisição de dados, que é representada por um software que roda em um PC por meio de uma interface USB-Serial. Esse componente, por sua vez, representa a recepção da informação relacionada às leituras de memória e dos valores medidos para as variáveis relacionadas ao ambiente, o controle do acionamento da unidade microcontroladora por meio do sinal DTR da porta serial que está conectado de forma a habilitar a alimentação da placa. Por meio do pino *Power Supply* indicado na **Fig. 16** é possível fornecer uma tensão controlada por uma fonte externa de forma que os experimentos a serem conduzidos possam validar a influência da variação da tensão de alimentação sobre a resposta da PUF.

4.4 IMPLEMENTAÇÃO DO SOFTWARE EMBARCADO NO MICROCONTROLADOR

De acordo com o exposto anteriormente o primeiro passo a ser implementado pelo software embarcado no microcontrolador é a rotina de comunicação serial que deve ser capaz de enviar os dados da memória imediatamente após o *startup* do circuito para o computador. O protocolo serial se baseia no *handshake* entre os pinos *DataAV* e *DataRead.* O
Algoritmo 1 descreve o conceito de implementação da comunicação por meio da interface serial assíncrona. Há alguns fatores importantes a serem ressaltados por meio deste trecho de código. O primeiro deles é o registro do ponteiro pDataByte para a posição de memória START_ADDRESS, que é uma *MACRO* para a posição de memória inicial de dados no microcontrolador que no caso deste microcontrolador possui o valor 0x200. Outro fator importante é verificar que a comunicação serial é feita de maneira sequencial, e não utiliza nenhuma chamada a nenhuma função ou interrupção, o que confere ao programa embarcado a possibilidade de otimização em tempo de compilação para utilizar o mínimo possível a ***stack***, o que é extremamente necessário para a aplicação desejada neste projeto.

```
	WDTCTL = WDTPW | WDTHOLD;// Stop watchdog timer

	register unsigned int bitCount = 0;
	register unsigned int bytesCounter = 0;

	register unsigned char * pDataByte = ( unsigned char *) START_ADDRESS;

	P1OUT &= 0x00;
	P1DIR = 0x03; // P1.0 ==> DataTX - P1.1 ==> DataAvailable - P1.2 <==
DataRead

	bitCount = 0;
	bytesCounter = 0;

	// We have 8 bits (1B) to send.
	while( bytesCounter < BYTES_QTY_TO_SEND )
	{

		//if( bytesCounter <= BYTES_QTY_TO_SEND )
		//{

			if( bitCount < 8 )
			{
				if ( *pDataByte & ( 0x01 << bitCount ) )
				{
					P1OUT |= 0x01; //Send 1
				}
				else
				{
					P1OUT &= 0xFE; // Send 0
				}

				// After we sent data set DataAvailable pin
				P1OUT |= 0x02;

				while( !(P1IN & (0x01 << 2) ) );// Wait for DataRead

				P1OUT &= 0xFD; // DataRead received. Drop DataAvailable

				while( (P1IN & (0x01 << 2)) );// Wait for DataRead DROP.

				bitCount++; //Increment bitCount
```

```
                }
                else if (bitCount >= 8)
                {
                        bitCount = 0;
                        pDataByte++;
                        bytesCounter++;
                }
        }
```

Algoritmo 1 – Envio de dados via interface serial para o computador

Quando um programa é carregado na memória ele é organizado em três áreas diferentes, chamadas segmentos: *Text, Stack e Heap*. O segmento *Text*, comumente chamado de segmento *Code* é onde o programa compilado é armazenado na memória, isto é, após todos os processos de compilação, linkagem, transformação em código assembly e por fim instruções de máquina, o programa nada mais é do que uma série de instruções armazenadas na memória que são carregadas e executadas a cada conjunto de ciclos do processador, e é nessa posição de memória que o programa começa a ser armazenado. O segmento *Heap* é utilizado quando o programa do usuário aloca memória em tempo de execução por meio de funções como malloc e calloc, mas o seu escopo não é de grande relevância para a implementação do código embarcado no microcontrolador, por isso não será discutido em detalhes. Já o segmento *Stack* representa é utilizado para armazenar variáveis **locais**, **passagem de argumentos para funções** por meio da instrução *return* e o endereço para o qual o programa deve retornar após execução de alguma chamada de função [34]. Esse conceito pode ser visualizado na
Fig. 17. Essa discussão em torno da estrutura de um programa após sua organização na memória em tempo de execução leva à seguinte conclusão a respeito de como o código do software a ser embarcado no microcontrolador deve ser estruturado de forma a não gerar utilização da memória RAM para o segmento *stack*: O software implementado no MCU não pode ter nenhuma declaração de variável que não explicitamente indicada para ser armazenada em um registrador "interno" ao processador, nem chamada a nenhuma função que não seja a principal (***main***), tampouco rotina de tratamento de interrupções, pois todos esses elementos obrigariam o compilador a alocar um espaço da memória de dados para o segmento *stack*, e isso no contexto do projeto é altamente indesejado, já que a caracterização da PUF em SRAM deve ser capaz de avaliar senão todos, o máximo possível de endereços, sem exceção, quanto ao seu comportamento durante o *startup* do sistema. É importante também, mencionar o termo registrador interno ao processador com certa ressalva, pois no caso deste microcontrolador (MSP430F2013) como na maioria dos modelos comerciais em produção atualmente não há registradores internos ao processador que o programador possa acessar, estes registradores são mapeados em memória, ou seja, não são registradores propriamente ditos implementados diretamente no processado, mas sim ponteiros para alguma posição de memória FLASH reservada a este proposito. Embora, a presença de registradores internos ao processador sejam uma alternativa extremamente rápida para acesso à informação, a utilização de registradores mapeados em memória se tornou uma prática comum e eficaz, pois facilita a construção do hardware do processador e implementa a mesma funcionalidade. Vale mencionar também que, como exposto na Fig. 9, a memória da microcontroladora possui os endereços de 0x00 a 0x0F para o que chamam-se de *Special Function Registers*¸ são estes registradores que são utilizados para armazenar as variáveis *bitCount* e *bytesCounter* de forma que seus valores não sejam armazenados na memória de dados do microcontrolador, pois estes endereços estão sendo utilizados na medição e caracterização da PUF em SRAM.

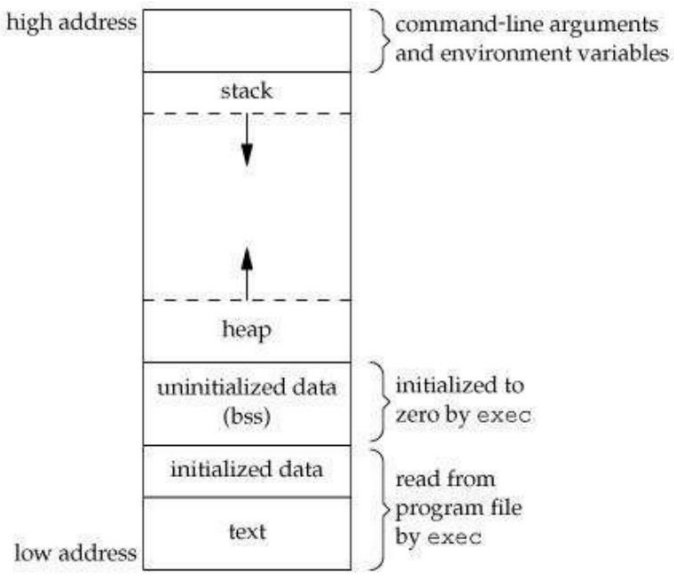


Fig. 17 – Organização típica de um programa compilado na memória de um processador

Mencionou-se anteriormente que o ambiente deve influenciar nas respostas de uma PUF. De forma, a isolar algumas dessas variáveis implementou-se aqui utilizando o conversor SD16 integrado ao microcontrolador MSP430F2013 de forma a medir a temperatura sob a qual o ensaio está sendo realizado e a tensão de alimentação (Vcc) aplicada ao sistema durante o ensaio. O que é considerado pertinente, portanto, são os detalhes de implementação e interfaceamento com este periférico do microcontrolador e como isso foi desenvolvido utilizando as ferramentas disponibilizadas pelo fabricante. Os exemplos a serem demonstrados aqui lançam mão de pequenos trechos de código para explicar o conceito de implementação da leitura de dados por meio do conversor SD16. A Fig. 11, pode ser utilizada como referência para os termos e registradores que aqui são mostrados.

i) Leitura de dados de temperatura:

O procedimento para efetuar leitura de temperatura utilizando o conversor A/D SD16 possui alguns passos a serem seguidos, os quais são descritos por meio da apresentação do algoritmo abaixo:

```
SD16CTL = SD16REFON + SD16SSEL_1;    // 1.2V ref, SMCLK
SD16INCTL0 = SD16INCH_6;             // A6+/-
SD16CCTL0 = SD16SNGL + SD16UNI;      // Single conv, Unipolar

SD16CCTL0 |= SD16SC;                 // Start SD16 conversion

while( !(SD16CCTL0 & 0x04) );       //Wait
```

Algoritmo 2 – Configuração de registradores que habilita o funcionamento do conversor SD16 para leitura de temperatura

ii) Leitura de dados de informação sobre tensão de alimentação Vcc:

O procedimento para efetuar a leitura de tensão de alimentação – Vcc – é análogo ao anterior, todavia o canal selecionado para a entrada do conversor SD16 nesse caso é o canal 5, pois o fabricante disponibilizou um divisor de tensão resistivo que pode servir para medições de tensão de alimentação utilizando o conversor SD16. O mostra a configuração necessária.

```
   SD16INCTL0 = SD16INCH_5;            // A5+/-
   SD16CCTL0 = SD16SNGL + SD16UNI;     // Single conv, Unipolar

   SD16CCTL0 |= SD16SC;                // Start SD16 conversion


   while( !(SD16CCTL0 & 0x04) );      //Wait until there is temperature
data in SD16MEM0 - When SD16IFG is set continue.
```

Algoritmo 3 – Configuração de registradores que habilita o funcionamento do conversor SD16 para leitura de tensão de alimentação por meio do divisor resistivo

Os registradores de interesse, de acordo com informações do fabricante, devem ser configurados de maneira a possibilitar a conversão A/D por meio do conversor SD16 da maneira apropriada. Nos algoritmos Algoritmo 2 e Algoritmo 3 são demonstradas as configurações específicas para que o conversor utilize tensão de referência de 1.2 V e efetue apenas uma conversão unipolar. A conversão unipolar de acordo com dados adquiridos por meio do datasheet fornecido pela *Texas Instruments* funciona de acordo com o esquema apresentado na Fig. 18.

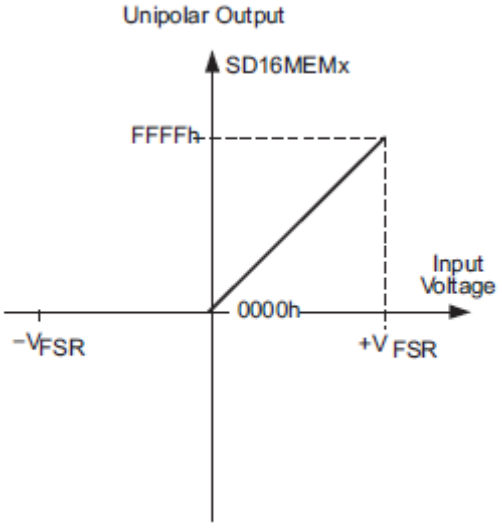


Fig. 18 – Curva característica da resposta de uma conversão unipolar

Sendo assim, os dados de temperatura e tensão convertidos em decimal serão armazenados no registrador SD16MEM0. O valor lido é então transmitido por meio do protocolo serial assíncrono mencionado na sessão 4.2. O algoritmo para implementação dessa funcionalidade é apresentado a seguir:

```
Algoritmo 2 – OMITIDO.

      pDataByte = (unsigned char*)START_ADDRESS;

      *pDataByte = LOWBYTE(SD16MEM0);
      bitCount = 0;

      /*ROTINA DE ENVIO DO ENDEREÇO pDataByte PELO PROTOCOLO SERIAL OMITIDA*/

       pDataByte++;
      *pDataByte = HIGHBYTE(SD16MEM0);
      bitCount = 0;

      /*ROTINA DE ENVIO DO ENDEREÇO pDataByte PELO PROTOCOLO SERIAL OMITIDA*/

     Algoritmo 3 – OMITIDO

      pDataByte = (unsigned char*)START_ADDRESS;

      *pDataByte = LOWBYTE(SD16MEM0);
      bitCount = 0;

      /*ROTINA DE ENVIO DO ENDEREÇO pDataByte PELO PROTOCOLO SERIAL OMITIDA*/

       pDataByte++;
      *pDataByte = HIGHBYTE(SD16MEM0);
      bitCount = 0;

      /*ROTINA DE ENVIO DO ENDEREÇO pDataByte PELO PROTOCOLO SERIAL OMITIDA*/
```

Algoritmo 4 – Algoritmo para envio dos dados de temperatura e Vcc

O Algoritmo 4 possui algumas características interessantes a serem discutidas. A primeira delas é o fato de armazenar a informação contida no registrador SD16MEM0 na posição de memória *START_ADDRESS* que é uma *MACRO* para a posição de memória inicial de dados no microcontrolador, que neste caso em específico possui o valor 0x200. A este ponto, isso poderia levantar o seguinte questionamento: Se a memória de dados está sendo utilizada para caracterização da SRAM-PUF não seria incorreto utilizar os endereços de memória de dados para armazenar informação? A resposta é sim, todavia é pertinente ressaltar que as medições de temperatura e Vcc são realizadas apenas depois de toda a informação de memória durante o estágio *power-up* ter sido lida e devidamente enviada para o computador. As macros *HIGHBYTE* e *LOWBYTE* foram definidas para simplificar a implementação do código e a descrição de seus comportamentos podem ser vistas no Algoritmo 5. A representação de números inteiros no microcontroladores da família MSP430X2XXX possuem dois bytes, ou seja, são armazenados em 16 bits. Como a comunicação serial implementada neste trabalho envia dados byte a byte, é necessário enviar cada byte do inteiro em um ciclo de comunicação com o computador. Isso justifica a criação dessas macros para acessar as informações contidas no registrador SD16MEM0, que neste caso, está sendo utilizado para armazenamento das informações de temperatura e tensão de alimentação do sistema.

```
#define LOWBYTE(v)   ((unsigned char) (v) )
#define HIGHBYTE(v)  ((unsigned char) (((unsigned int) (v)) >> 8))
```

Algoritmo 5 –Macros para acesso aos bytes de um inteiro na memória do microcontrolador.

4.5 IMPLEMENTAÇÃO DO FRAMEWORK DE AQUISIÇÃO E ANÁLISE DE DADOS

De modo a receber o pacote de dados de informação fornecido pela placa que contém a PUF em análise o framework implementa uma série de comandos que atuam sob os pinos CTS, DSR, RTS e DTR da porta de comunicação serial do computador de modo a gerar os estados necessários para a comunicação serial assíncrona. Esses pinos são comumente utilizados em sistemas onde o protocolo de comunicação é RS232 para realizar um controle de estados na comunicação conhecido por *Flow Control*, por exemplo, o pino CTS – Clear To Send – é utilizado para informar ao computador que o canal está livre para transmissão em uma comunicação padrão. O significado destes pinos neste projeto não possuem nenhuma correlação com a implementação padrão RS232, eles são utilizados aqui por serem pinos de fácil acesso por meio das APIs do sistema operacional e setar a linha ou resetar a linha é bastante simples quando se têm domínio das bibliotecas de controle de acesso a recursos seriais tanto em ambiente Windows como Linux. O framework de aquisição de dados realiza um processo análogo ao mostrado anteriormente na implementação do software embarcado no MCU.

```
 unsigned char memDumpSerialData[256];

int bitCount = 0;
int bytesCounter = 0;
int i = 0;

DEFINIÇAO DE VARIAVEIS E CONFIGURAÇAO DA PORTA OMITIDAS

rc = OpenSerialComm(&comPort, &config);

puf_msp430_setVccResetPin(comPort, 0);
Sleep(2000); // Sleep 0.5 sec

puf_msp430_setVccResetPin(comPort, 1);

rc = puf_msp430_resetPins(comPort);
if (rc < 0) return;

while (1)
{
      if (bitCount >= 8)
      {
            bitCount = 0;

            memDumpSerialData[bytesCounter] = rxByte;

            rxByte = 0x00;
            bytesCounter++;

      }

      if (bytesCounter >= MEM_SIZE)
```

```
            {
                  break;
            }

            rc = puf_msp430_getDataAvailablePin(comPort, &dsr);
            if (dsr)
            {

                  //There is data to be read.

                  puf_msp430_readDataBit(comPort, &rxByte, bitCount);
                  bitCount++;

                  //SET Rts Pin to tell MCU that I read.
                  puf_msp430_setDataReadenPin(comPort, 1);

                  //Wait for MCU to Drop Data Available Pin
                  rc = puf_msp430_getDataAvailablePin(comPort, &dsr);
                  if (dsr == 0)
                  {
                        puf_msp430_setDataReadenPin(comPort, 0);
                  }
            }
      }

      for (i = 0; i < MEM_SIZE - 4 /*Porque tem 4 bytes para Vcc e Temperatura*/;
i++)
      {
            sprintf(sMemPrintBuffer + strlen(sMemPrintBuffer), "%02X ",
                        memDumpSerialData[i]);
      }
      TraceFileLog(sLogFileName, MSGS, "Memory dump size[%d] Readen: %s",
                  MEM_SIZE, sMemPrintBuffer);

      iVcc = (memDumpSerialData[MEM_SIZE-1] << 8 ) | memDumpSerialData[MEM_SIZE-
2];
      iTemperature = (memDumpSerialData[MEM_SIZE-3] << 8 ) |
memDumpSerialData[MEM_SIZE-4];

      fVcc = calculaVcc(iVcc);
      fTemperature = calculaTemp(iTemperature);

      TraceFileLog(sLogFileName, MSGS, "Vcc = %f - Temp = %f \n", fVcc,
fTemperature);


      CloseSerialComm(&comPort, &config);
```

Algoritmo 6 – Rotina de comunicação serial implementada sob a ótica do framework de aquisição de dados

É importante notar que o código aqui é um pouco diferente do código embarcado no microcontrolador, isso é devido a uma série de fatores, entre eles destaca-se o fato de a disponibilidade de recursos de processamento e memória de um computador pessoal ser praticamente ilimitada para esse tipo de aplicação e a interação com o sistema operacional terminam por tornar o código bem mais simples e versátil. O ciclo de aquisição de dados inicia com o sistema solicitando ao usuário a quantidade de testes que ele deseja realizar e o nome do arquivo onde ele deseja guardar as informações de dumps consecutivos obtidos por meio da leitura da memória, temperatura e tensão de alimentação. Após a confirmação do usuário o sistema inicia o processo de reiniciar a placa por meio do sinal de controle DTR como mostrado anteriormente na Fig. 16, e inicia a aquisição do pacote de informações até que todas as posições de memória sejam lidas. À partir deste momento o sistema

efetua a leitura de outros 4 Bytes referentes às leituras de temperatura e tensão de alimentação (Vcc). Quando o ciclo é finalizado o sistema reinicia a placa novamente e repete este procedimento até que o número limite de ensaios seja atingido.
`

Após o término do ciclo de aquisição de dados o *framework* gera no diretório corrente uma série de arquivos com as informações obtidas a cada ensaio e um arquivo global, em que todos os dumps realizados são expostos de maneira alinhada para proporcionar um primeiro contato visual com a forma dos dados obtidos por meio dos ensaios. Esta etapa é importante, pois caso haja alguma falha durante o processo, como por exemplo algum cabo ser desconectado durante a as medidas, uma linha inteira será mostrada com valores **nulos** e isso deve saltar aos olhos de forma que o experimento problemático possa ser eliminado e não interferir na análise das medidas. A
Fig. 19 mostra um trecho de um arquivo global de dumps de memória adquiridos por meio do *framework* em modo de aquisição de dados.

```
[  1] - [D6 02 00 40 02 10 02 02 00 84 00 08 04 00 02 00 02 24 20 00 40 02
[  2] - [23 A5 13 62 02 10 02 02 00 84 00 08 04 00 02 00 02 24 20 00 40 02
[  3] - [27 A5 0E 62 02 10 02 02 00 84 00 08 04 00 02 00 02 24 20 00 40 02
[  4] - [2C A5 1B 62 02 10 02 02 00 84 00 08 04 00 02 00 02 24 20 00 40 02
[  5] - [31 A5 10 62 02 10 02 02 00 84 00 08 04 00 02 00 02 24 20 00 40 02
[  6] - [37 A5 16 62 02 10 02 02 00 84 00 08 04 00 02 00 02 24 20 00 40 02
[  7] - [34 A5 07 62 02 10 02 02 00 84 00 08 04 00 02 00 02 24 20 00 40 02
[  8] - [36 A5 12 62 02 10 02 02 00 84 00 08 04 00 02 00 02 24 20 00 40 02
[  9] - [36 A5 2C 62 02 10 02 02 00 84 00 08 04 00 02 00 02 24 20 00 40 02
[ 10] - [38 A5 0E 62 02 10 02 02 00 84 00 08 04 00 02 00 02 24 20 00 40 02
[ 11] - [D6 82 A0 44 22 80 42 82 C2 84 46 04 44 06 42 80 41 2A 61 0A 43 41
```

Fig. 19 – Exemplo de arquivo de saída do framework quando utilizado em modo de aquisição de dados

Note que o arquivo contém em cada linha a representação de um bloco de memória em forma hexadecimal. A primeira posição da linha 1, por exemplo, representa o conteúdo da memória no endereço 0x200h logo após a placa ter sido energizada, a segunda posição o endereço 0x201h e assim sucessivamente até o byte 0x27Fh que representa a última posição de memória disponível para leitura no microcontrolador utilizado. Esses valores não são fixos, e podem ser ajustados para que o sistema leia uma quantidade diferente de bytes, caso desejemos utilizar um dispositivo diferente para avaliar, bastaria seguir os passos apresentados anteriormente para implementação do software embarcado de forma que o sistema que roda no computador não seria afetado e receberia os dados da mesma maneira.

Após a execução do software em modo de aquisição de dados toda a memória *startup* do circuito foi lida por uma quantidade determinada de vezes. O próximo passo é desenvolver uma estratégia de análise para essas informações. A primeira idéia que surge quando o assunto é comparar *streams* de bits é utilizar o conceito de distância de *Hamming*. Este conceito é útil, pois é capaz de mensurar quão diferente uma palavra binária é da outra. Por exemplo, a distância de *Hamming* entre 0101 e 0100 é 1, pois apenas o bit menos significativo é diferente nas duas palavras. Desse modo o framework deve ser capaz de calcular a distância de Hamming entre os dumps de memória, apontar quais as posições de memória que mudaram pelo menos uma vez durante o *set* de experimentos e dentre essas posições que variaram quantas vezes cada uma delas variou de forma a atribuir pesos a estas medidas. A Fig. 20 representa este conceito de maneira clara para um exemplo de dado armazenado na posição 0x200h da memória do microcontrolador. Note que o diagrama exemplifica o conceito por meio de três dumps consecutivos e que a comparação é realizada de forma a comparar as combinações entre os dados coletados e gerar uma máscara que identifica o comportamento dessa variação.

Dump number	*Memory[0x200h]*
1	0xFC – 1111 1100
2	0xFE – 1111 1110
3	0xFD – 1111 1101
MASK (1 – 2)	0x02 – 0000 0010 ($HD_{intra} = 1$)
MASK (1 – 3)	0x01 – 0000 0001 ($HD_{intra} = 1$)
MASK (2 – 3)	0x03 – 0000 0011 ($HD_{intra} = 2$)
Global MASK	0x03 – 0000 0011 ($HD_{intra} = 2$)

Fig. 20 – Exemplo de comparação entre posições de memória consecutivas em dumps diferentes de memória.

Essa máscara global possui zeros nas posições de memória que não sofreram nenhuma modificação em nenhum dos dumps realizados durante o ensaio e uns nas posições que sofreram alteração pelo menos uma vez durante os experimentos. Após esse procedimento bastaria contar a quantidade de uns na máscara de saída para verificar qual o percentual da memória que sofreu alteração durante os 1000 experimentos. E dessa maneira, é interessante retomar o conceito de distâncias intra-classe mencionado na sessão 2.1, pois para verificar a distribuição estatística dessas medidas deve-se armazenar de alguma forma as distâncias de hamming intra-classe e organizá-las em um histograma. Este procedimento deve ser capaz de demonstrar a posição onde a média das medidas se concentra e então mensurar qual o percentual da célula de memória que na média poderia ser utilizada para a implementação de uma PUF em SRAM.

Dessas observações surge a necessidade de implementar um algoritmo capaz de mesclar as informações obtidas por meio da comparação dos dumps de memória aos pares de forma a caracterizar o comportamento do bloco de memória. Definiu-se então três maneiras diferentes para efetuar a comparação dos valores coletados. A primeira delas é denominada Modo-0 de comparação e consiste em comparar todas as combinações possíveis de dumps de memória coletados. Por meio da utilização da função lógica *XOR – Exclusive OR* – que é capaz de gerar o que é mostrado na Fig. 20, são calculadas máscaras individuais (1-2, 1-3, 2-3, ...), e por meio da expressão lógica *OR* entre todas as máscaras individuais apresenta-se o que foi definido como máscara global. Note que essa metologia de cálculo considera a variação de cada bit em uma determinada posição da memória de forma não-homogênea, por exemplo, no caso em que todas as distâncias de Hamming associadas a estas combinações sejam organizadas graficamente em um histograma, a variação de um mesmo bit mais de uma vez durante o experimento irá gerar a mesma máscara global de saída, todavia, gerará valores maiores de distância de Hamming associado a essa comparação, pois um efeito que já havia sido considerado em uma comparação anterior está sendo considerado novamente nesta medição. A esta altura é plausível questionar a validade de realizar este tipo de comparação, pois com base no que acaba de ser exposto parece que ela representa um excesso de esforço sem beneficiar a qualidade das medidas, entretanto percebeu-se que essa estratégia proporciona ao sistema uma característica interessante: A possibilidade de atribuição de pesos para as posições de memória que variaram durante o experimento. Para cada vez que um determinado bit varia seu valor um medidor é incrementado de forma a computar a quantidade de vezes que aquela posição de memória sofreu alteração e com isso, pode fornecer uma noção de quão aleatório é o comportamento de cada célula SRAM. Um exemplo gráfico deste método é demonstrado na Fig. 21.

Outra forma desenvolvida para comparar as medidas foi denominada Modo-1 de comparação e consiste em comparar todos os dumps coletados com uma referência que neste caso é considerada como o primeiro dump adquirido. É possível verificar por meio da análise da Fig. 20 que a máscara de saída será idêntica e representa corretamente as posições de memória que sofreram alteração pelo menos uma vez durante a realização do experimento. Quando as distâncias de Hamming são plotadas em um Histograma elas fornecem de maneira objetiva uma idéia sobre a distribuição da variação das posições de memória ao longo de todo o espaço amostral definido para o experimento, ou seja, uma análise estatística do experimento como um todo. Esse procedimento é exemplificado por meio do diagrama na Fig. 22. Adicionalmente foi desenvolvido um terceiro método com o intuito de fornecer

uma idéia de como as medidas variam em pequenos grupos de análises consecutivas, ou seja, um método capaz de comparar amostras consecutivas e organizar as distâncias de Hamming em forma de um histograma de forma a possibilitar a comparação entre os resultados obtidos por meio do método anterior e a análise de amostras consecutivas (Fig. 23). Este conceito é análogo à definição do conceito de velocidade média e velocidade instantânea da mecânica, que embora representem valores diferentes, quando avaliadas em conjunto ajudam a interpretar em detalhes as características de um corpo em movimento.

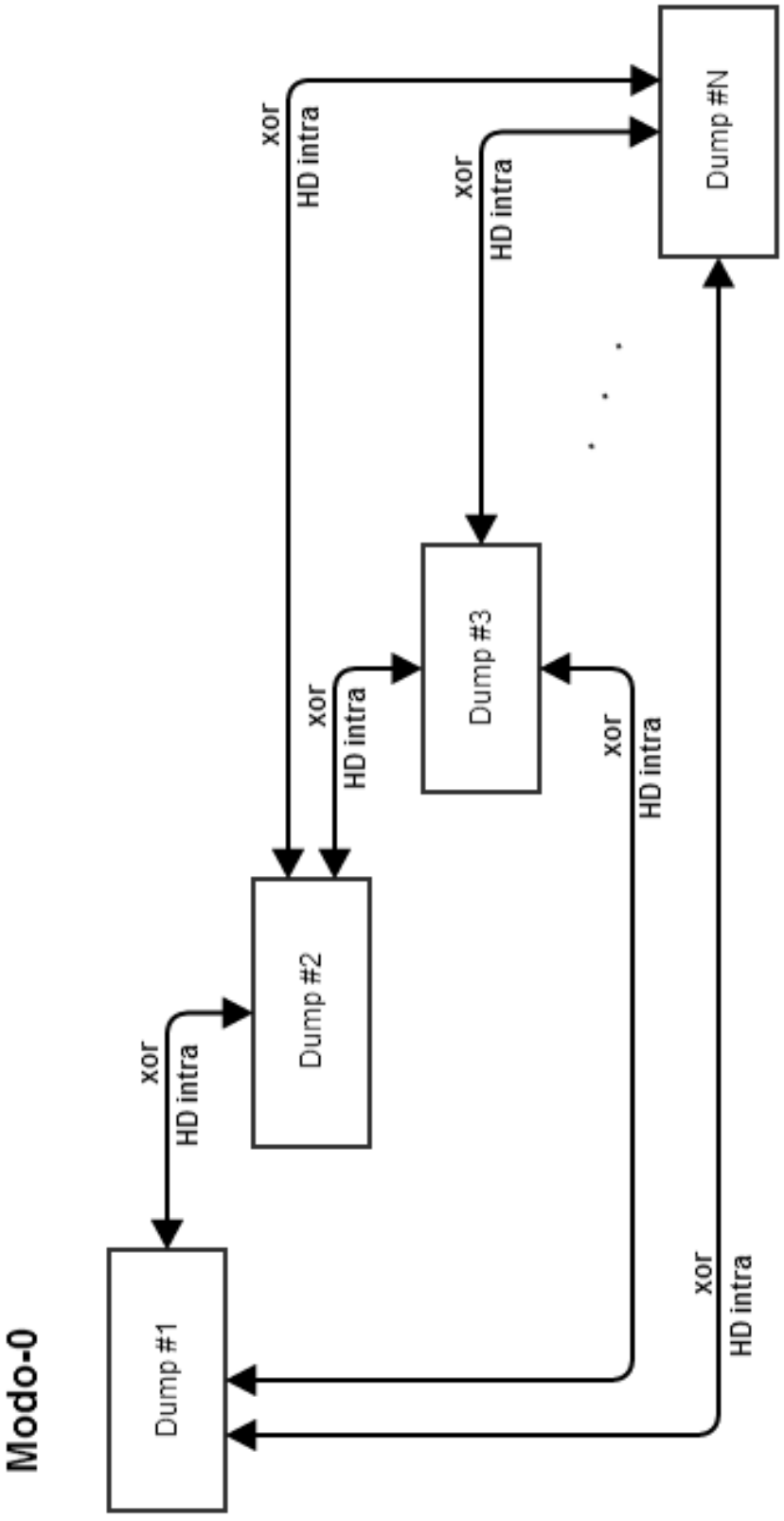


Fig. 21 – Exemplo gráfico do Modo-0 de comparação

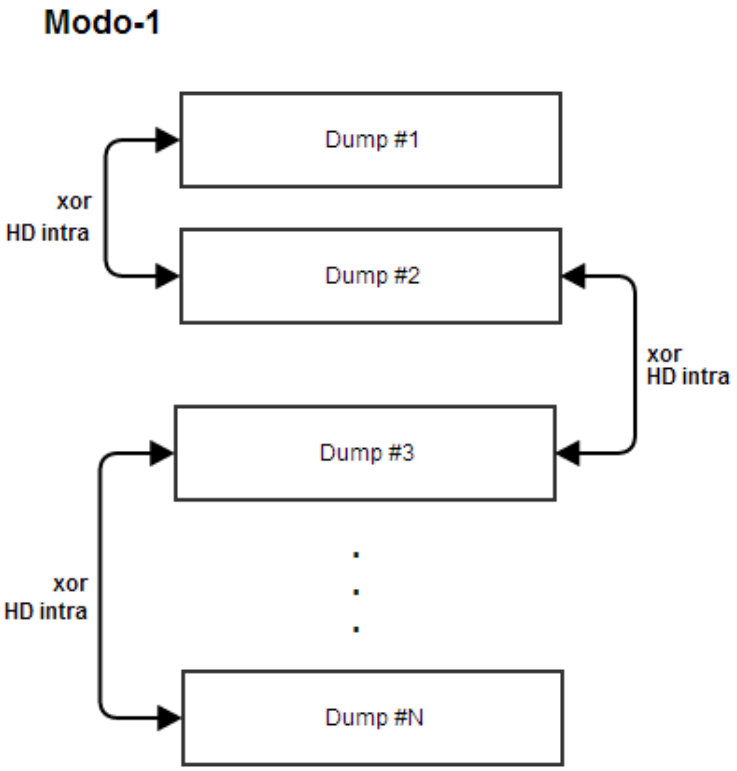


Fig. 22 – Exemplo gráfico do Modo-1 de comparação

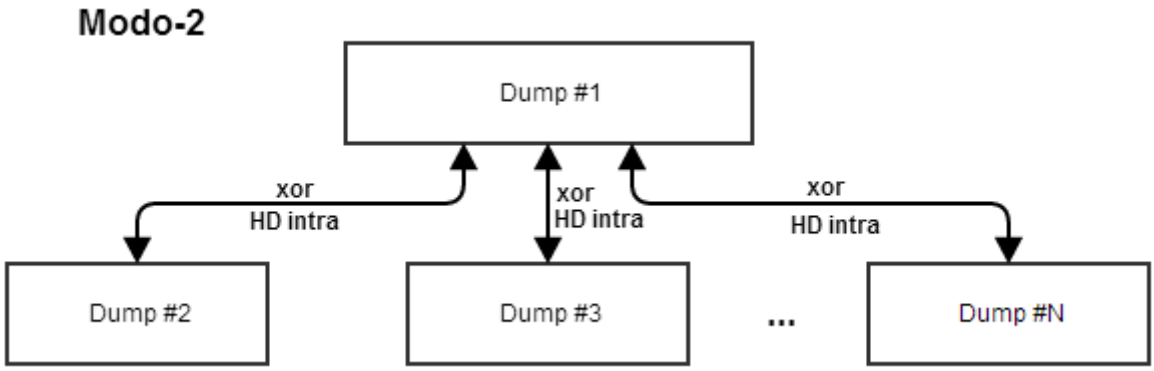


Fig. 23 – Exemplo gráfico do Modo-2 de comparação

De forma a sistematizar a comparação dos histogramas a serem apresentados serão utilizados valores fixos para as variáveis do ambiente, isto é, serão fixadas diferentes condições de Vcc e temperatura e então realizados os experimentos, de tal forma que além da avaliação da célula de memória como uma aplicação para a PUF será possível indicar uma tendência de variação nas medidas conforme a variaçãõ de determinada variável do ambiente. Sendo assim, de posse dos dados das variáveis relacionadas ao ambiente transmitidos em representação binária, é necessário interpretá-los de forma a obter as medidas de forma visualmente agradável. Para o cálculo da temperatura medida utiliza-se a curva característica de resposta do sensor de temperatura incluído no microcontrolador. Esses dados são fornecidos pelo fabricante e estão disponíveis na Fig. 24 na forma de uma curva linear que relaciona a tensão lida em Volts com a temperatura ambiente em Celsius. Para

utilização apropriada da curva característica de transferência do sensor de temperatura é necessário conhecer os coeficientes mencionados na equação da figura a seguir.

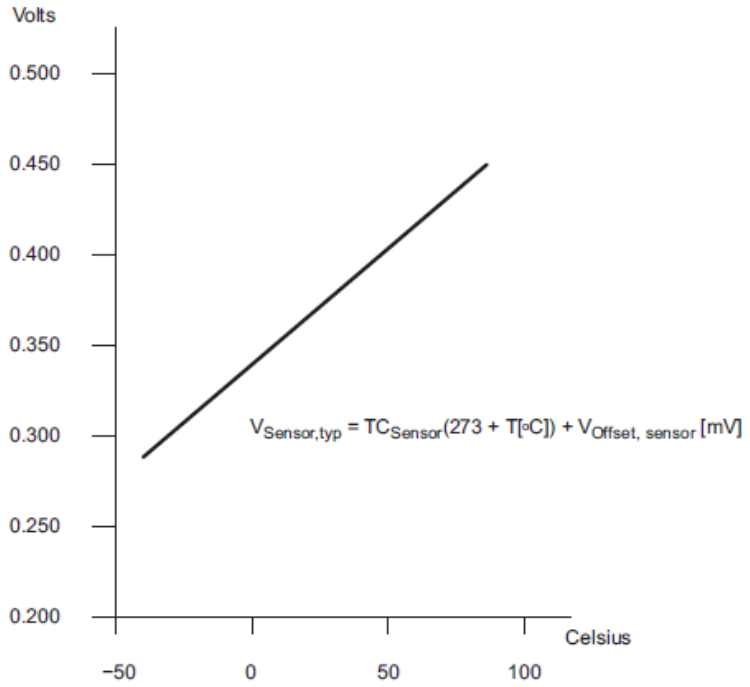


Fig. 24 – Função de transferência do sensor de temperatura

PARAMETER		TEST CONDITIONS	T_A	V_{CC}	MIN	TYP	MAX	UNIT
V_{REF}	Internal reference voltage	SD16REFON = 1, SD16VMIDON = 0		3 V	1.14	1.20	1.26	V
I_{REF}	Reference supply current	SD16REFON = 1, SD16VMIDON = 0	-40°C to 85°C	3 V		190	280	µA
			105°C	3 V			295	
TC	Temperature coefficient	SD16REFON = 1, SD16VMIDON = 0		3 V		18	50	ppm/°C
C_{REF}	V_{REF} load capacitance	SD16REFON = 1, SD16VMIDON = 0 (1)				100		nF
I_{LOAD}	$V_{REF(I)}$ maximum load current	SD16REFON = 1, SD16VMIDON = 0		3 V			±200	nA
t_{ON}	Turn-on time	SD16REFON = 0 → 1, SD16VMIDON = 0, C_{REF} = 100 nF		3 V		5		ms
DC PSR	DC power supply rejection $\Delta V_{REF}/\Delta V_{CC}$	SD16REFON = 1, SD16VMIDON = 0, V_{CC} = 2.5 V to 3.6 V		2.5 V to 3.6 V		100		µV/V

Fig. 25 – Parâmetros característicos dos sensores associados ao conversor SD16.

É possível definir a seguinte equação para o cálculo da temperatura medida em Celsius:

$$T_{celsius} = \frac{Valor\ Decimal \cdot \frac{1}{2^{16}} \cdot \frac{Vref}{2} \cdot 1000}{TCsensor} - 273\ ;\ Vref = 1.2V, TCsensor = 1.32mV/ºC$$

Obs: Valor decimal representa o valor de temperatura que o microcontrolador enviou para o computador.

Exemplo: Imagine que o sistema recebeu ***0xA7D6*** para a leitura de temperatura. Convertendo este valor para decimal obtem-se 42966 e aplicando a equação acima:

$$T_{celsius} = \frac{42966 \cdot \frac{1}{2^{16}} \cdot \frac{1.2}{2} \cdot 1000}{1.32} - 273 = 25.004\ ℃$$

Para derivarmos uma equação para cálculo da tensão de alimentação basta lembrar que a leitura de tensão vêm de um divisor resistivo sob o resistor R, onde o total da resistência série é 11R (Fig. 11). Da divisão de tensão, pode-se definir a seguinte expressão para a tensão sobre o resistor R, que é a entrada lida pelo conversor SD16:

$$Vr = \frac{R}{11R} \cdot Vcc, \qquad Vr\ representa\ a\ tensão\ sobre\ o\ resistor\ R$$

Como Vr é o valor que foi lido pelo conversor analógico-digital:

$$Vr = \ Valor\ Decimal \cdot \frac{1}{2^{16}} \cdot \frac{Vref}{2}$$

Entretanto, estamos interessados em calcular a tensão de alimentação do circuito.

Então:

$$Vcc = 11 \cdot Vr = 11 \cdot Valor\ Decimal \cdot \frac{1}{2^{16}} \cdot \frac{Vref}{2}$$

Exemplo: Imagine que o sistema recebeu ***0x814C*** para a leitura de tensão Vcc. Convertendo este valor para decimal obtem-se 33100 e aplicando a equação acima:

$$Vcc = 11 \cdot 33100 \cdot \frac{1}{2^{16}} \cdot \frac{1.2}{2} = \ 3.33\ V$$

De posse dos métodos de cálculo implementados, estruturas de aquisição, comparação e avaliação dos dados coletados considerou-se a execução de um *set* de experimentos que pudesse exemplificar o comportamento do bloco e proporcionar conclusões a respeito do sistema. Essas discussões serão detalhadas no capítulo a seguir.

5 SETUP EXPERIMENTAL E RESULTADOS

Este capítulo tem por objetivo apresentar os experimentos realizados e analisar os resultados provenientes dos gráficos e informações aqui demonstradas. O comportamento do bloco de memória SRAM de um microcontrolador MSP430F2013 é submetido a uma série de ensaios que envolvem a variação da tensão de alimentação e controle da temperatura sob a qual o sistema em teste está submetido.

As sessões anteriores deste trabalho se basearam na descrição teórica do problema de caracterizar o funcionamento de blocos de memória SRAM como PUFs, descreveram a necessidade da construção de um *setup* experimental que fôsse capaz de auxiliar a compreensão da distribuição estatística de dumps consecutivos de memória, bem como a apresentação de máscaras que demonstrem quais posições de memória podem ser consideradas confiáveis durante a construção de uma PUF baseada na arquitetura de memória presente em circuitos *off-the-shelf* como microcontroladores, e até mesmo em SoCs. Todos os experimentos se baseiam na análise dos dumps de memória coletados por meio dos modos discutidos anteriormente, e a organização dos resultados fornecidos por eles utilizando MATLAB em forma de histogramas e um gráfico que demonstra os pesos de variação de cada posição de memória.

5.1 DETERMINAÇÃO DA QUANTIDADE DE AMOSTRAS NECESSÁRIAS

Na capítulo 4 foram descritas as estratégias de cálculo e análise que seriam empregadas neste projeto. Foram mencionados os modos 0,1 e 2 de comparação, os quais diferem apenas entre a metodologia de combinação a ser considerada entre os valores coletados. É possível perceber, que o Modo-0, que considera todas as combinações possíveis dos valores medidos pode representar um esforço computacional elevado dependendo do número de amostras definido para cada experimento. Dessa observação surge a necessidade de escolher um valor que apresente relevância estatística e que ao mesmo tempo não torne o procedimento experimental inviável. Mencionou-se em outras etapas deste projeto também que a escolha se deu em torno do valor 1000 para o tamanho das amostras nos experimentos aqui realizados. Isso significa que o número de comparações a serem realizadas é, no pior caso – Modo-0, da ordem da combinação do número de experimentos tomado dois a dois. Sendo assim, a quantidade de comparações a serem realizadas no caso extremo pelo software de análise é da ordem de:

$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!} = \binom{1000}{2} = \frac{1000!}{998! \cdot 2!} = \frac{1000 \cdot 999}{2} = 499500$$

Para decidir um número de amostras que seja eficaz em termos de velocidade computacional e que possua relevância estatística é interessante verificar como a função binomial varia em função do número de amostras (N).

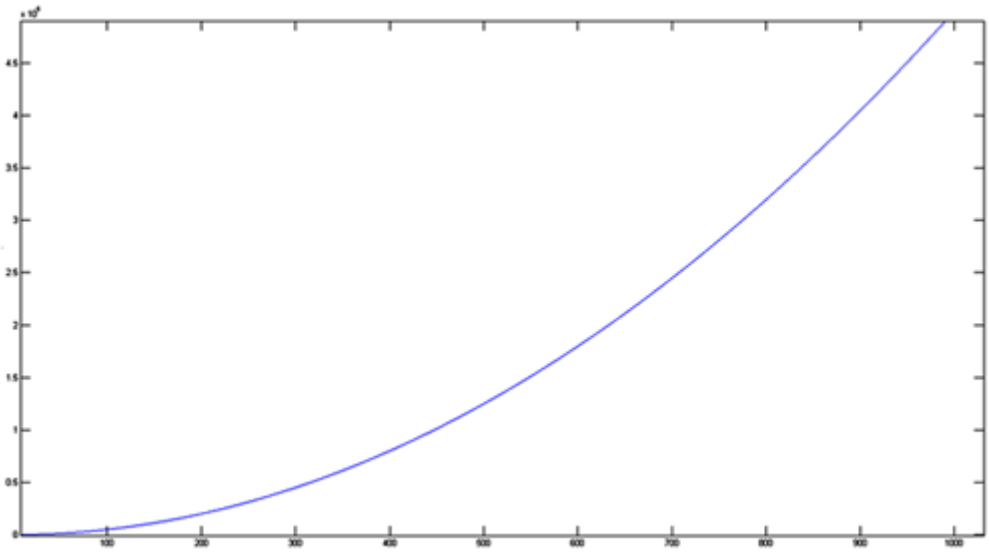


Fig. 26 – Comportamento da função binomial em função do número de amostras

$$\binom{n}{2} = \frac{n \cdot (n-1)}{2} = O(n^2)$$

A equação acima demonstra o conceito de que o algoritmo do Modo-0 em questão possui complexidade computacional da ordem de n^2. É intuitivo perceber que os algoritmos dos modos 1 e 2 possuem complexidade da ordem de n , o que garante que se for possível encontrar um valor que faça com que os experimentos sejam realizáveis para o Modo-0 este problema estará resolvido para os algoritmos dos modos 1 e 2. Considerando, portanto, que o conjunto de instruções de um processador comum que roda em um PC padrão possui instruções capaz de realizar esta comparação em apenas 1 ciclo de execução para cada bit de memória, somados aos ciclos de *fetch* e *decode* de cada instrução temos que um processador rodando uma frequência de aproximadamente 500 MHz seria capaz de realizar o processo em pouco mais de 1 segundo. Todavia, sabe-se que os processadores atuais realizam suas intruções em frequências da ordem de GHz, e o fato de realizar 1000 dumps por ensaio, embora represente um valor alto quanto à complexidade computacional e avaliar todas as combinações entre si de todos as amostras parece uma opção aceitável para que o software de análise execute as comparações de maneira eficiente. Resta portanto, avaliar a validade estatística dos valores encontrados comparados com o número de amostras por ensaio. Para caracterizar a validade das medidas quanto ao número de amostras fixou-se a tensão de 3.6 V e realizou-se o experimento com valores de amostras de 100, 250, 500 e 1000. A Fig. 27 demonstra a variação da resolução dos histogramas em função do número de amostras. É interessante mencionar que o caso em que o número de amostras é 100 o histograma transmite a idéia de que a média das medidas se encontra em torno de 10%, todavia quando coloca-se este histograma em comparação com outro que possui um número de amostras superior é possível perceber que o histograma fornece uma aproximação da média, todavia, não possui resolução suficiente para levantar adequadamente a distribuição dos valores ao longo do experimento.

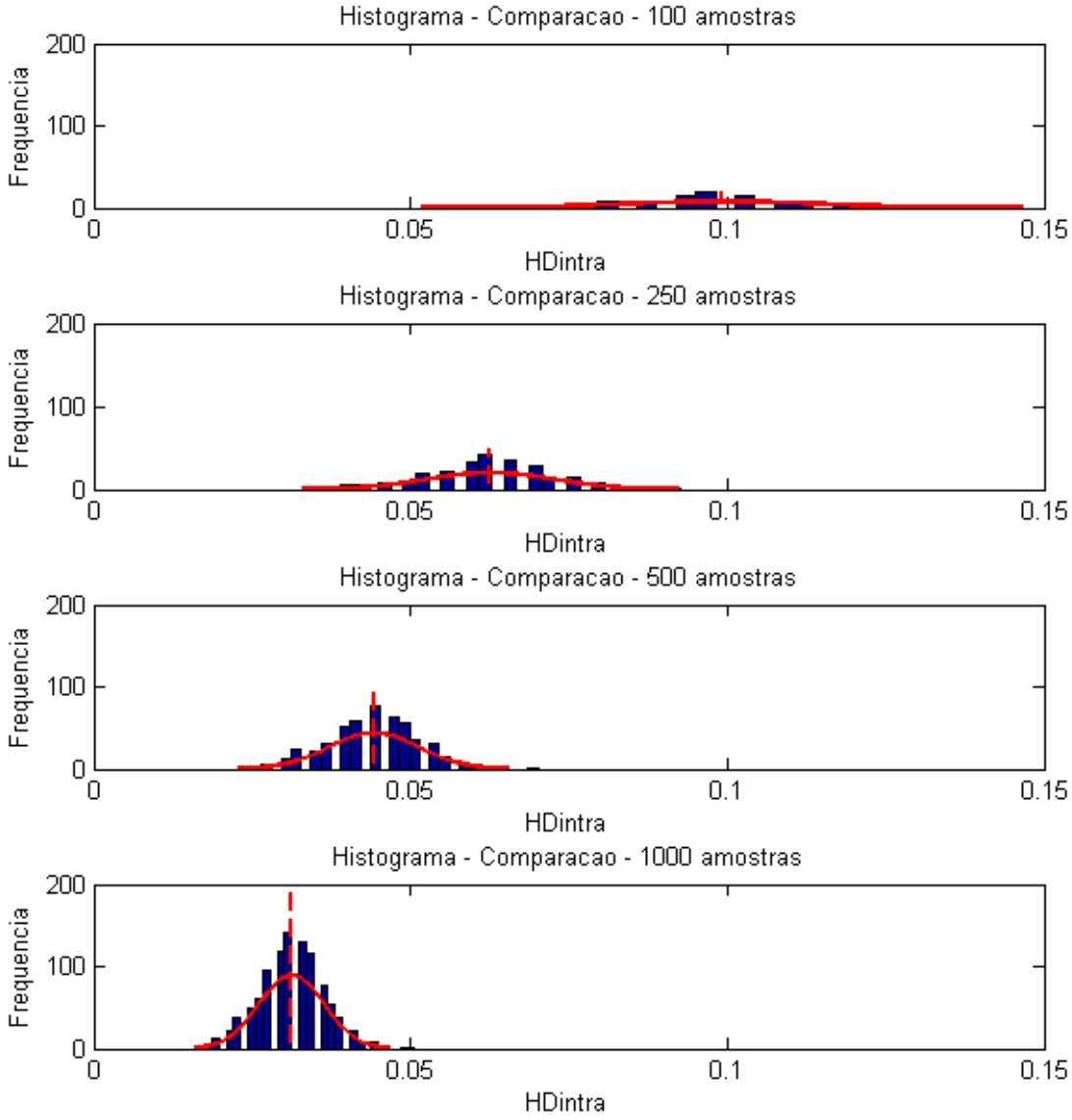


Fig. 27 – Comparação entre a resolução dos histogramas em função do número de amostras (3.6 V)

Note que conforme o número de amostras cresce a resolução dos histogramas é aumentada e comparando o histograma onde o número de amostras é 500 para o caso em que o histograma possui 1000 amostras a variação da média é alterada sutilmente. Os valores para a média da distribuição estão marcados pela linha vertical vermelha, e os valores considerados para essa análise para os gráficos são as seguintes:

100 amostras – Média 0.1
250 amostras – Média 0.06
500 amostras – Média 0.04
1000 amostras – Média 0.03

Dessa forma, definiu-se que o valor 1000 para as amostras dos experimentos é suficiente para fornecer informações cujo processamento computacional é viável. Esse conceito foi herdado da teoria de cálculo numérico, onde são realizadas iterações até que uma precisão considerada aceitável seja alcançada.

5.2 ANÁLISE DA VARIAÇÃO DE TENSÃO EM FUNÇÃO DA COMPARAÇÃO DOS DUMPS DE MEMÓRIA UTILIZANDO O MODO-1

Vcc	Temperatura	Quantidade
2.6 V	19 ℃	1000
3.0 V	18 ℃	1000
3.3 V	21 ℃	1000
3.6 V	23 ℃	1000

Fig. 28 – Condições para os experimentos realizados nesta sessão

Nesta sessão serão apresentados resultados referentes a experimentos utilizando as condições expostas na Fig. 28 e é importante notar que a tensão de alimentação varia de forma a cobrir o intervalo de funcionamento do microcontrolador especificado no datasheet anexado ao final do manuscrito. Nas especificações do fabricante a placa deve ser alimentado entre 1.8 V e 3.6 V, todavia se recordarmos a topologia empregada para desenvolvimento da placa de testes iremos verificar que existe um transistor BJT cuja queda de tensão típica é da ordem de 0.7 V a 0.8 V e por isso o valor mínimo utilizado foi 2.6 V. Outra observação importante envolve a variação da temperatura, pois embora o experimento esteja sendo conduzido de forma a avaliar apenas a variação da tensão de alimentação o framework deveria ser capaz de controlar a temperatura do circuito de forma a garantir que ela estivesse fixa no valor 20 ℃ que é a temperatura comum arbitrária escolhida para a realização dos experimentos. Todavia, não foi possível incluir no sistema uma ferramenta que fosse capaz de controlar a temperatura então procorou-se manter a variação de temperatura em torno do valor central desejado. Das informações fornecidas pelo fabricante do microcontrolador percebeu-se que o intervalo de temperaturas suportado é de -40 ℃ a 85 ℃ e e que para realizar um experimento que pudesse avaliar a influência da variação de temperatura sobre a resposta da PUF o framework deveria ser capaz de controlar a temperatura de maneira a produzir resultados eficientes para valores que representassem variação expressiva dentro deste intervalo. Considerando a variação apresentada pelo experimento, de 5 ℃, comparada à magnitude do intervalo de temperaturas do microcontrolador, 125 ℃, verifica-se que a variação de temperatura no experimento representa 4% de variação, o que foi considerado, um valor cuja relevância deveria ser desprezada. Como sugestão para trabalhos futuros sugere-se que seja acrescentado uma forma de controlar e variar a temperatura em torno dos valores fornecidos pelo fabricante.

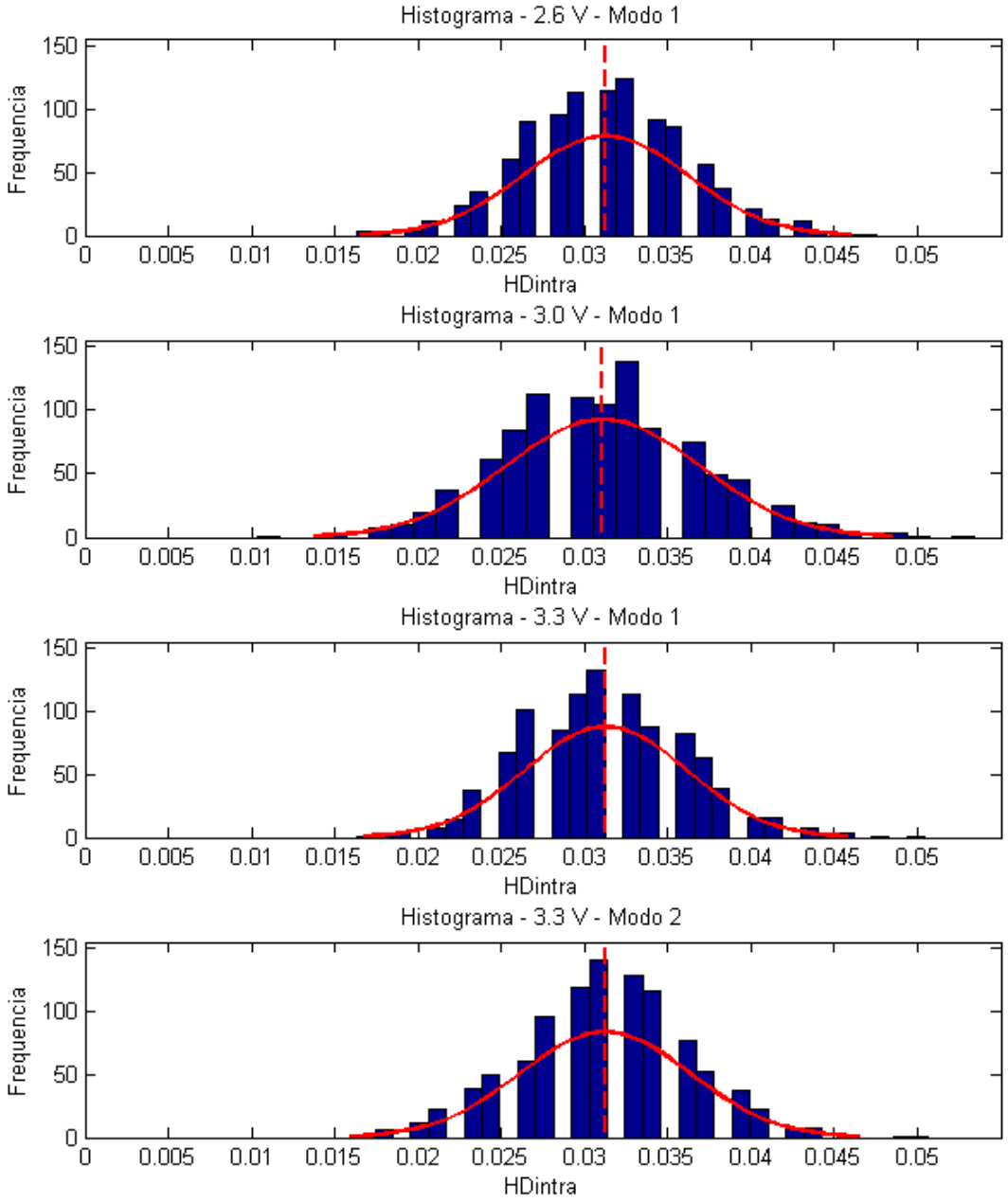


Fig. 29 – Histogramas referentes às medidas descritas nessa sessão – 1000 amostras considerando variação de tensão sob o Modo-1 de comparação.

Para o intervalo de tensões amostrado a média de variação das distâncias intra-classe permaneceu aproximadamente constante. Isso pode representar um indício de que a distribuição das distâncias de Hamming não deve ser afetada pela variação na tensão de alimentação (Vcc). É importante retomar o conceito de distância de Hamming intra-classe que representa a quantidade de bits que é diferente de um amostra para a outra. Nesta sessão, os valores são medidos comparando cada dump de memória coletado em relação ao primeiro dump que foi lido e essa estratégia leva a avaliar a variação intra-classe no experimento como um todo, ou seja, conforme o tempo passa, ela transmite uma idéia com relação ao primeiro valor amostrado que representaria neste caso o tempo t=0. Essa análise é importante, pois descreve o comportamento do sistema conforme a passagem do tempo e é extremamente desejável que a resposta da PUF seja de certa forma resistente à passagem do tempo. Como mencionado anteriormente há dois estágios para a autenticação via PUF, a primeira delas é representada pelo ciclo de *enrollment* que via de regra seria executado pelo fabricante para armazenar os pares *challenge-response* e o estágio de autenticação onde o sistema do usuário aplica um desafio, recebe a resposta e a compara com o CRP database adquirido no estágio anterior. Dessa forma, não é interessante que a resposta da PUF varie expressivamente com relação à passagem do tempo, pois deseja-se que o usuário seja capaz de reproduzir de maneira o mais próximo possível do especificado no CRP database.

A conclusão que pode-se inferir dos valores demonstrados na Fig. 29 é a de que há um indício de que a variação da tensão de alimentação dentro do intervalo apresentado aqui, de 2.6 V a 3.6 V, não deve influenciar majoritariamente na distribuição das distâncias de Hamming intra-classe. Uma dúvida que poderia surgir a este ponto é a seguinte: Se a variação da tensão de alimentação não influenciar na distribuição das medidas ao longo do espaço amostral como um todo então as máscaras, que representem quais posições de memória variaram durante os experimentos, deveriam ser iguais para todos os casos? Isso é um questionamento válido e com os resultados apresentados até aqui ainda não é possível inferir se a variação de Vcc influenciou na resposta da PUF em termos de alterar o comportamento de determinada célula de memória. Outro fator que ainda resta ser mensurado é o comportamento instantâneo das medidas, isto é, verificar como as distâncias de Hamming se comportariam caso estivéssemos avaliando dumps consecutivos de memória e não apenas comparando-os com uma referência que no caso do Modo-1 é o primeiro dump lido. As discussões a respeito destes dois assuntos motiva os expostos nas sessões a seguir.

5.3 ANÁLISE DA VARIAÇÃO DA TENSÃO EM FUNÇÃO DA COMPARAÇÃO DOS DUMPS DE MEMÓRIA UTILIZANDO MODO-2

Vcc	Temperatura	Quantidade
2.6 *V*	20 ℃	1000
3.0 *V*	21 ℃	1000
3.3 *V*	19 ℃	1000
3.6 *V*	23 ℃	1000

Fig. 30 – Condições para os experimentos realizados nesta sessão

Com base nos resultados apresentados na sessão anterior verificou-se uma análise da distribuição das distâncias de Hamming intra-classe quando as medidas são comparadas com relação a uma referência estabelecida por meio da primeira leitura de memória. Mostrou-se que quando se avalia a resposta da PUF em SRAM ao longo do espaço amostral como um todo a resposta não sofre influência majoritária da variação de tensão. Nesta sessão, o intuito é avaliar a influência da variação de tensão sobre a resposta da PUF quanto ao comportamento "instantâneo", isto é, leituras consecutivas de memória quando submetidas a tensões de alimentação diferentes apresentam comportamentos diferentes? Para responder a este questionamento realizaram-se as medidas utilizando 1000 leituras consecutivas de memória para cada valor de tensão desejado. Outra observação que vale a pena ser relembrada é a variação da temperatura, que será desprezada neste experimento, pois a

magnitude de variação de 4 ℃ quando comparada à escala de variação fornecida pelo fabricante do microcontrolador sendo analisado que é de 125 ℃ representa percentualmente um efeito que não será considerado aqui. A motivação para a comparação dos dumps consecutivos de memória é avaliar o comportamento do bloco de memória SRAM em termos de seu comportamento em espaços curtos de tempo, como mencionado anteriormente, esse conceito é análogo à análise da velocidade instantânea de um corpo em movimento. Essa análise deve ser capaz de fornecer uma idéia sobre a variação da distribuição das distâncias de Hamming intra-classe associadas ao bloco de memória quanto à variação da tensão de alimentação (Vcc).

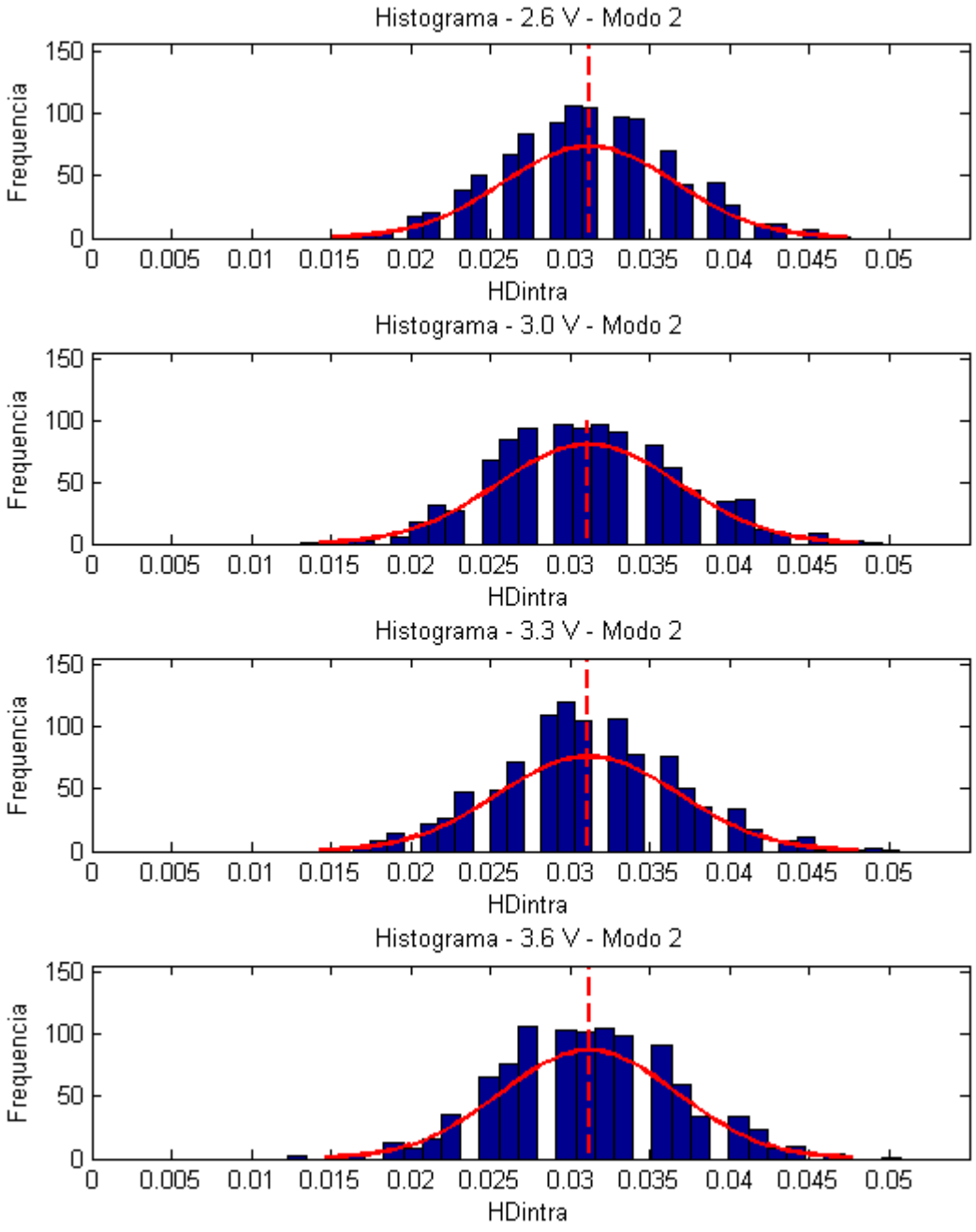


Fig. 31 – Histogramas referentes às medidas descritas nessa sessão – 1000 amostras considerando variação de tensão sob o Modo-2 de comparação

Para o intervalo de tensões amostrado durante os experimentos verificou-se que a média das distâncias de Hamming não variou em função da tensão de alimentação aplicada ao sistema. Dessa forma a análise de dumps consecutivos leva à conclusão de que embora a distribuição das distâncias de Hamming intra-classe não sejam idênticas, a média para este dispositivo em específico será aproximadamente constante e igual a 3%. Dessa observação pode-se dizer que o comportamento "instantâneo" das comparações entre os dumps de memória se comporta de maneira bastante similar ao comportamento da célula na média, o que foi apresentado na sessão anterior. Retomando o conceito de velocidade instantânea e média da mecânica pode-se inferir que o sistema se comporta como um corpo em movimento retilíneo uniforme, onde a velocidade instantânea é idêntica à velocidade média. Isso é importante, pois aqui é mostrado que uma memória SRAM apresenta células cujo comportamento é imprevisível, todavia, que se repetirá ao longo de uma série de experimentos ao qual a célula seja submetida. Esse resultado é considerado importante para a demonstração de que um bloco SRAM apresenta características que são adequadas à operação enquanto PUF. Todavia, ainda não foi realizada nenhuma análise a respeito do comportamento de cada célula individualmente. Estes tópicos serão discutidos na sessão a seguir.

5.4 ANÁLISE DO COMPORTAMENTO DAS CÉLULAS INDIVIDUAIS DE MEMÓRIA

Durante a pesquisa bibliográfica não foi encontrado um trabalho acerca de PUFs em SRAM que avaliasse o comportamento de cada célula de memória individualmente, isto é, as análises que foram encontradas até o ponto em que este trabalho foi escrito descreviam que o comportamento das células dependeriam de parâmetros de fabricação e que a célula após o *startup* poderia apresentar dois estados bem definidos, 0 ou 1, ou apresentar comportamento aleatório. De forma a perceber este efeito nas células individuais do bloco SRAM em questão será utilizado o Modo-0, que lança mão da comparação de todos os dumps de memória entre si e adicionalmente incrementa um contador individual para cada posição de memória. Assim, é possível plotar a resposta da PUF para um determinado valor de tensão e verificar visualmente o comportamento das células de memória quanto à variabilidade. Na Fig. 32 é mostrada a variabilidade de cada célula de memória em função da posição de memória que ela ocupa no bloco SRAM. Note que nas células que apresentam variação os valores são da ordem de 500 mil, o que leva a crer que as posições em que há alguma variação o estado assumido pela célula varia em praticamente 100% das amostras, pois todas as combinações representam um universo de 500 mil comparações como mencionado na sessão 5.1 e se o contador de variações apresenta valores da ordem desse número, isso leva a crer que as células que variam assumem o estado indefinido por detalhes de fabricação. Já no caso das células em que não há variação alguma percebe-se exatamente o contrário, o valor do contador permanece em zero e é possível notar que esse comportamento é o mais recorrente no bloco inteiro pela alta concentração de pontos em torno do eixo x do gráfico.

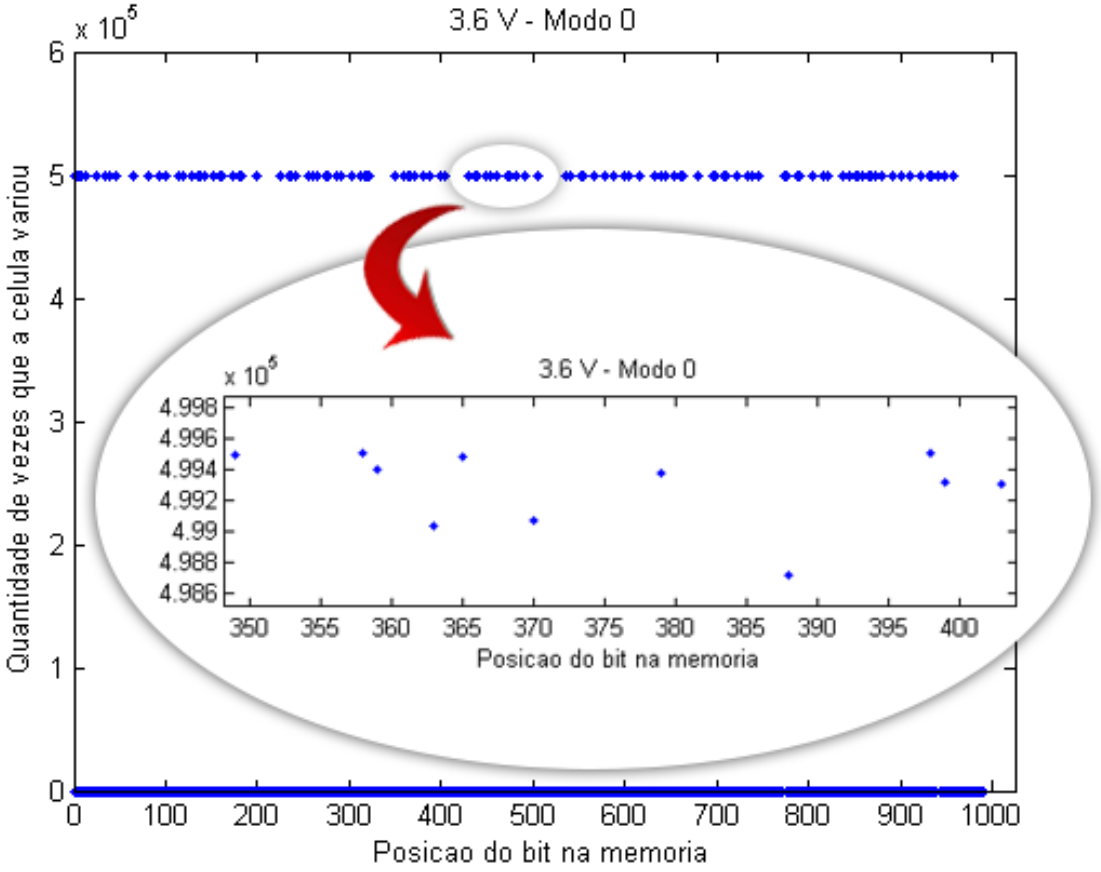


Fig. 32 – Comportamento da variabilidade em função da posição de memória da célula (3.6V)

Outra análise considerada importante neste ponto envolve verificar se a variação na tensão de alimentação (Vcc) faz com que alguma célula altere seu comportamento, isto é, se a aleatoriedade no comportamento de cada posição de memória é uma função da tensão Vcc. Para isso é interessante mencionar a idéia de que o valor do contador associado à quantidade de variações de uma determinada posição de memória será considerado um peso relacionado a esta posição. No gráfico apresentado, é possível definir um limite (*threshold*) bem definido para separar as posições de memória que variam das que não variam e aplicar um processamento que seja capaz de determinar se a posição de memória teve seu comportamento alterado em detrimento da variação da tensão de alimentação. Sendo assim, desenvolveu-se um algoritmo que compara os vetores que contém os pesos associados a cada posição de memória para as situações de tensão descritas nas sessões anteriores, que são 2.6 V, 3.0 V, 3.3 V e 3.6 V. Sempre que o módulo do valor do peso para uma determinada posição de memória em um ensaio subtraída do valor do peso para a mesma posição de memória relacionado a outro ensaio for maior do que o limite estabelecido um vetor auxiliar será marcado com o valor 1, e caso o valor dessa diferença seja menor do que o limite definido, o vetor será marcado com o valor 0. Dessa forma, pretende-se avaliar que para cada vez que o número 1 aparecer no gráfico a variação de Vcc apresentou influência no comportamento da célula, pois em um determinado ensaio, aquela posição de memória apresentou um determinado comportamento, seja ele, aleatório ou invariante, e após a variação da tensão de alimentação ela teria apresentado outro comportamento diferente.

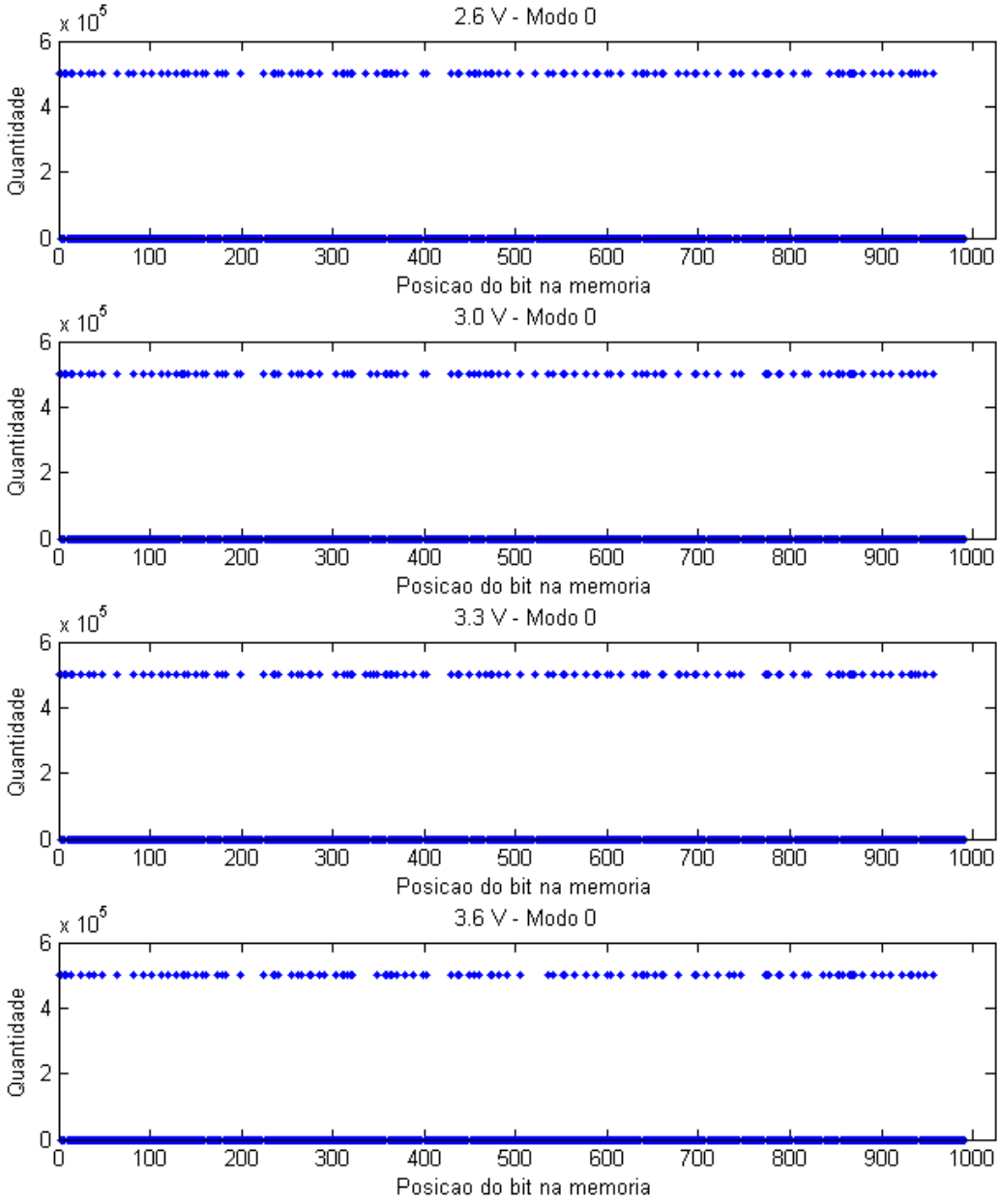


Fig. 33 – Pesos associados às posições de memória em diferentes cenários de tensão de alimentação

Na Fig. 33 é mostrado o comportamento dos pesos associados a cada posição de memória em diferentes cenários de tensão de alimentação. A semelhança entre as quatro curvas pode levar à impressão de que as respostas são iguais e que aparentemente o comportamento aleatório de cada célula de memória não foi alterado de um experimento para outro. Todavia, utilizando um algoritmo simples que compara essas quatro curvas de forma a marcar apenas as posições de memória em que o comportamento da célula variou em função da variação de Vcc, é possível notar que houve algumas posições de memória que variaram seu comportamento, como demonstrado pela Fig. 34.

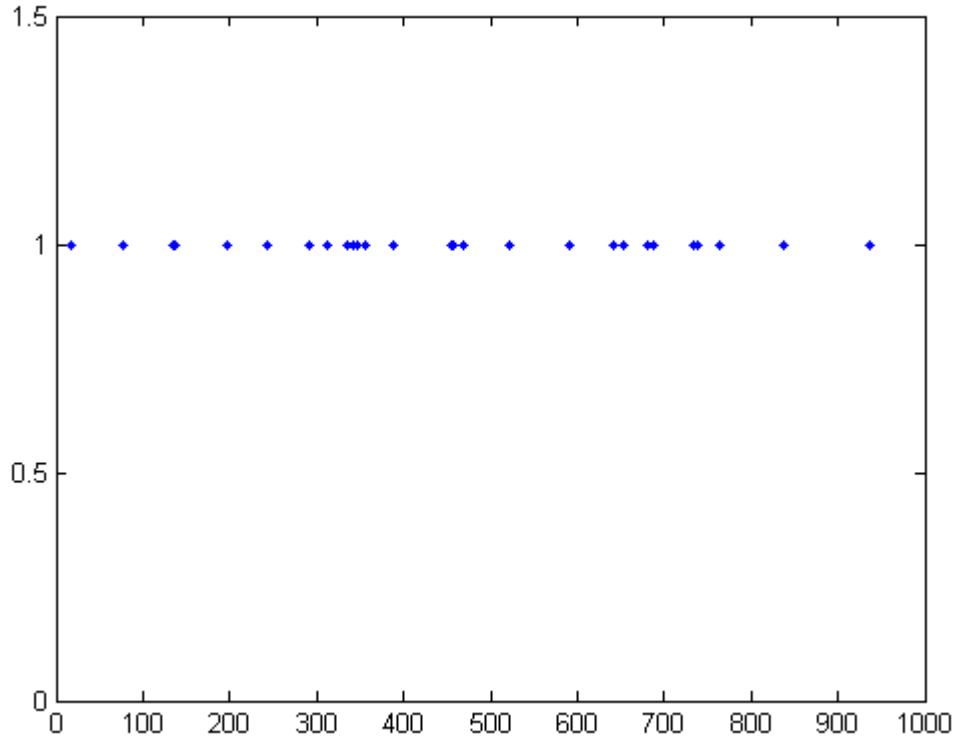


Fig. 34 – Análise do comportamento aleatório de cada célula em função da variação da tensão de alimentação

Uma conclusão importante deve ser extraída deste resultado que é o fato de que a aleatoriedade de algumas posições de memória é influenciada pela tensão de alimentação Vcc, isto é, uma célula cujo comportamento foi considerado estático em 0 ou em 1 em um experimento cuja tensão de alimentação foi fixada em 2.6 V, por exemplo, pode apresentar comportamento altamente aleatório em outro experimento conduzido a uma tensão de alimentação de 3.0 V. Todavia, estas posições que sofreram alteração, seja aumentando sua aleatoriedade ou caminhando para um estado estável em 0 ou em 1, em um experimento realizado por meio de um experimento cujo espaço amostral seja grande o suficiente não influencia na média da distribuição das distâncias de Hamming intra-classe. Para este experimento em específico 27 posições de memória sofreram alteração ao menos uma vez, isto é, para algum valor de tensão no experimento o valor do peso associado àquela posição de memória variou entre os limites superior e inferior do gráfico pelo menos uma vez. E isso quer dizer que $27\ em\ 1024$ células de memória sofreram alteração em seu comportamento em detrimento da variação da tensão de alimentação (Vcc), ou seja, apenas 2.6% sofreram alteração e mesmo com essa variação de comportamento, verificou-se nas sessões anteriores que este efeito não contribui para o deslocamento da média da distribuição das distâncias de Hamming intra-classe associadas às medidas. De forma a exemplificar este resultado é mostrado na Fig. 35 a variação de comportamento de uma célula , que no caso se encontra na posição de memória 937. Note que ora o peso associado a essa posição possui seu valor no limite superior do gráfico ($5x10^5$) e ora possui seu valor no limite inferior do gráfico. E é interessante verificar que as células vizinhas a ela não apresentaram o mesmo comportamento influenciado pela tensão de alimentação.

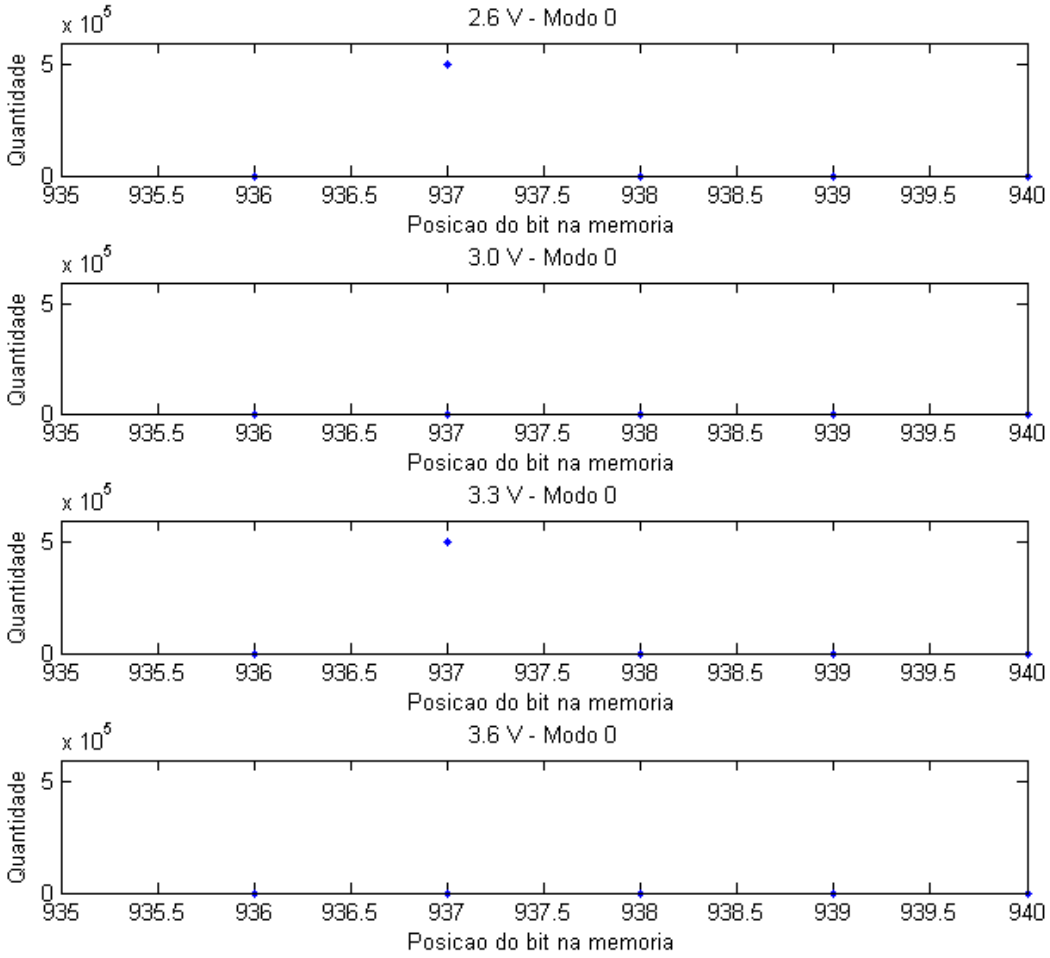


Fig. 35 – Exemplo de variação de comportamento da célula na posição 937 da memória SRAM em função da tensão de alimentação

Os resultados apresentados graficamente até aqui possibilitam a análise do comportamento do bloco de memória SRAM quanto às distâncias intra-classe, a distribuição estatística dessa métrica bem como a sua variação quando o bloco é submetido a diferentes tensões de alimentação. Outro fator interessante a ser discutido aqui é a distribuição espacial das células estáveis ou não estáveis quanto à sua localização na memória. Dessa forma, serão apresentados para cada valor de tensão de alimentação um *Heat Map*, que é uma representação visual do comportamento de cada posição individual. Uma consideração importante a ser realizada neste ponto é a de que para o sistema em questão a última posição de memória que apresentou comportamento variável foi a célula na posição 937 e pelo fato de um Heat Map só poder ser construído a partir de uma matriz quadrada serão considerados apenas 961 bits da memória. Isso não influencia na conclusão a ser tomada, pois a última posição de memória que apresentou variação em seu comportamento foi a célula mencionada anteriormente que estará incluída nos diagramas da Fig. 36.

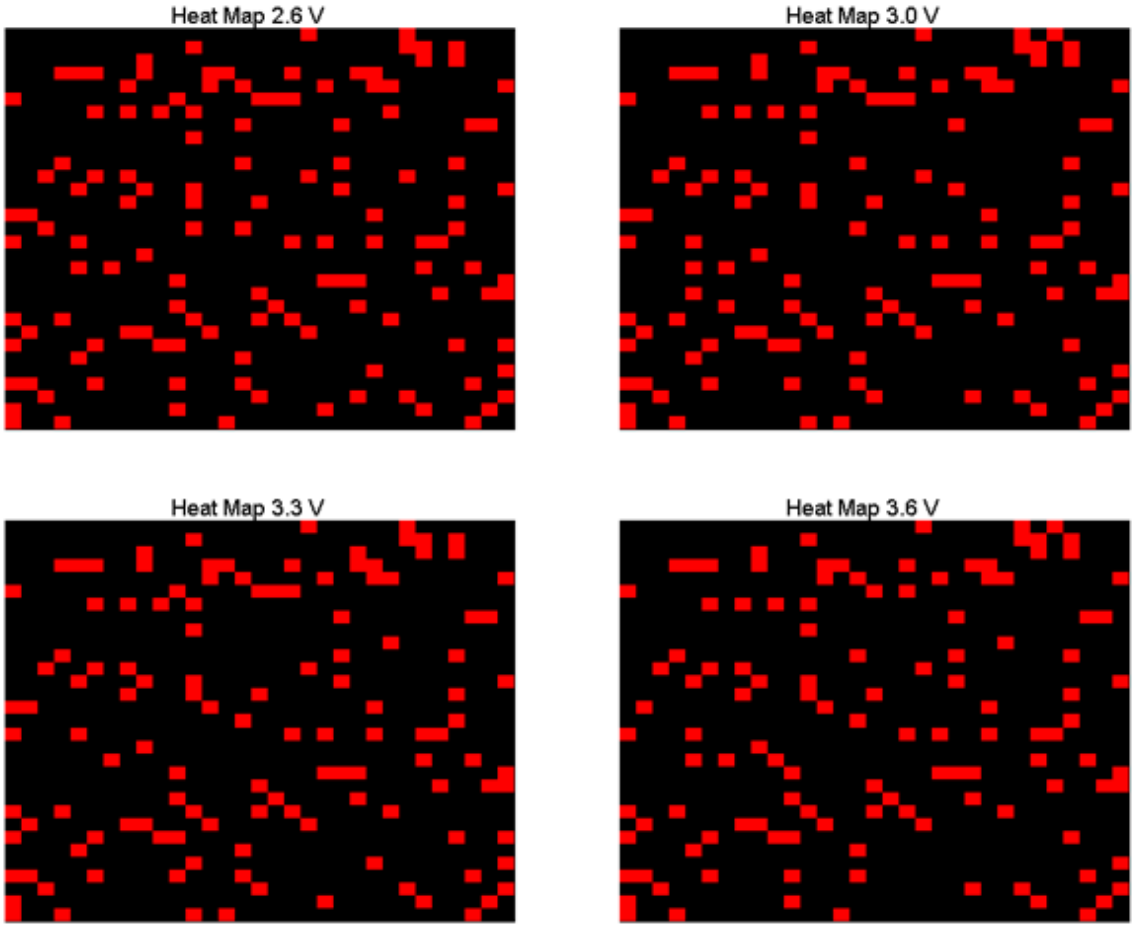


Fig. 36 – Heat Maps associados ao comportamento das posições de memória do bloco SRAM

Note que nos *Heat Maps* apresentados as células representadas pelos pixels em vermelho são as células cuja variabilidade foi comparada ao limite estabelecido por meio da análise dos gráficos apresentados anteriormente. Isto é, as células em preto representam células SRAM estáveis e as células em vermelho representam as células cujo comportamento foi considerado instável. Embora os *Heat Maps* sejam bastante parecidos eles representam o mesmo comportamento discutido anteriormente. Há algumas posições de memória que apresentam seu comportamento alterado pela variação da tensão de alimentação. Para demonstrar este conceito será apresentado um Heat Map em que as posições de memória que apresentaram seu comportamento alterado pela variação da tensão de alimentação são marcadas em vermelho, as células que apresentaram resistência a esta variação estão marcadas em verde, como pode ser verificado na Fig. 37.

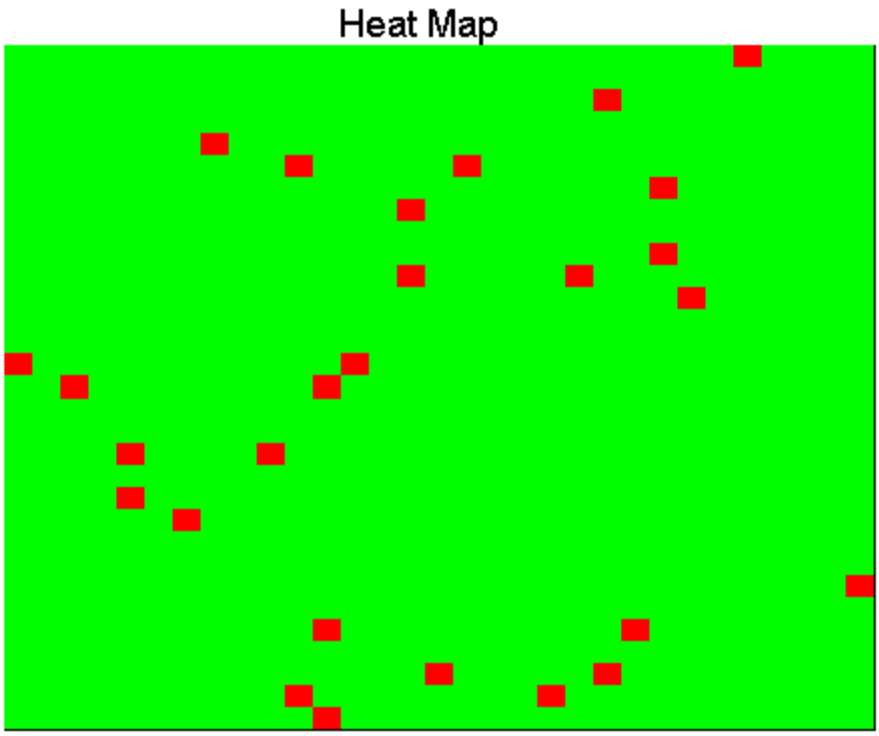


Fig. 37 – Heat Map associado às posições de memória que apresentaram seu comportamento alterado pela variação da tensão de alimentação (Vcc)

5.5 ANÁLISE PRELIMINAR DO COMPORTAMENTO INTER-CLASSE DE BLOCOS SRAM

O foco principal deste trabalho é apresentar e analisar o comportamento intra-classe de blocos de memória SRAM para implementação de Physically Unclonable Functions. Todavia, considerou-se válido apresentar a título de referência uma análise de três dispositivos da mesma série discutida ao longo deste manuscrito. Nas seções anteriores demonstrou-se que a variação da tensão de alimentação não influencia significativamente na média das distâncias de Hamming. Sendo assim, realizou-se o mesmo experimento utilizando 1000 amostras consecutivas para outros dois dispositivos microcontroladores MSP430F2013. Utilizando o modo-0 de comparação construiu-se o Heat Map associado à cada uma dessas placas e produziu-se um diagrama comparativo entre os diagramas de pesos apresentados anteriormente.

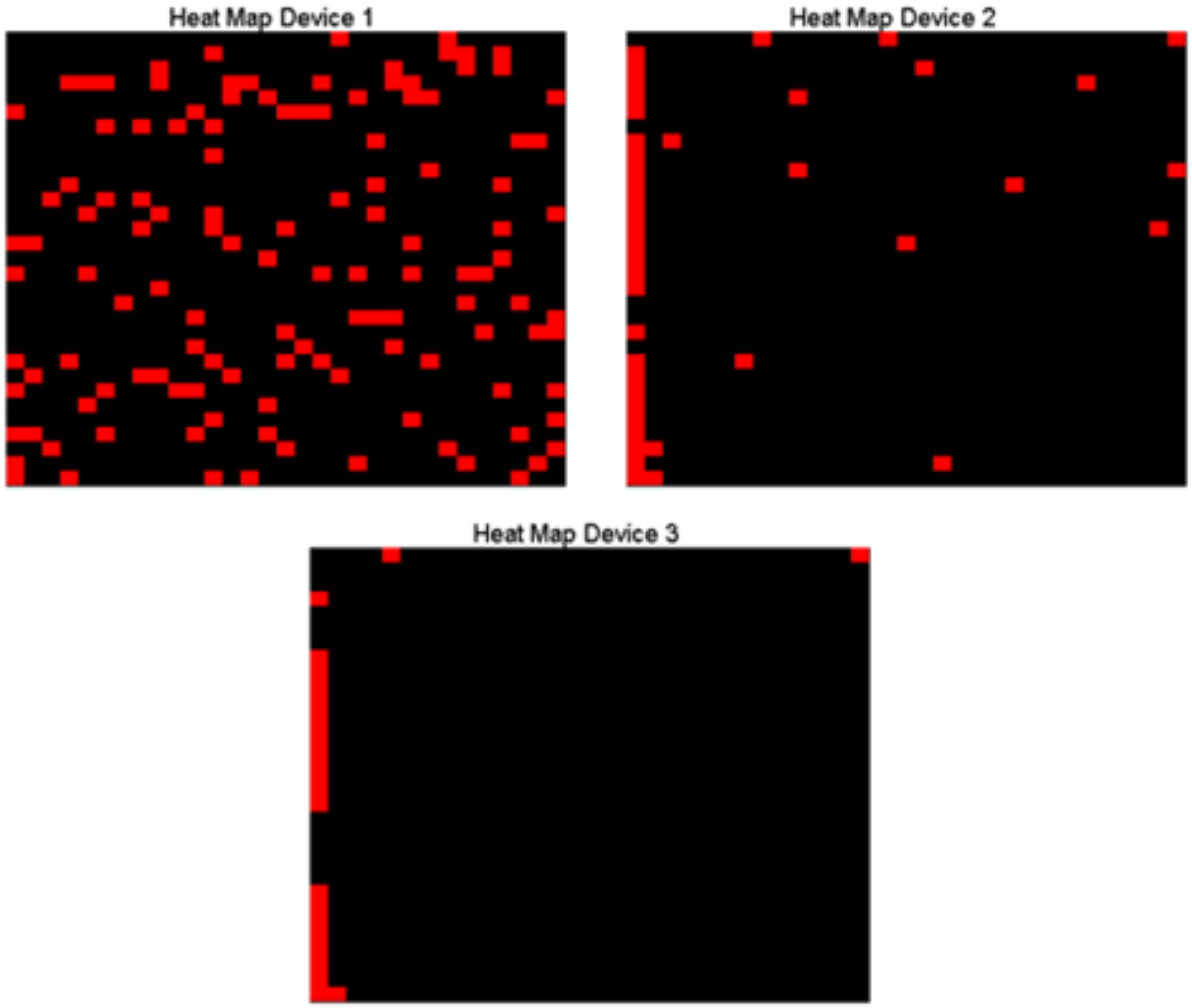


Fig. 38 – Heat Maps associados a três dispositivos do microcontrolador MSP430F2013

Na Fig. 38 é possível perceber que as placas apresentaram padrões de variação diferentes, o que pode representar um indício de que cada placa por diferenças durante o processo de fabricação possa apresentar um padrão único que pode ser utilizado para identificar o dispositivo.

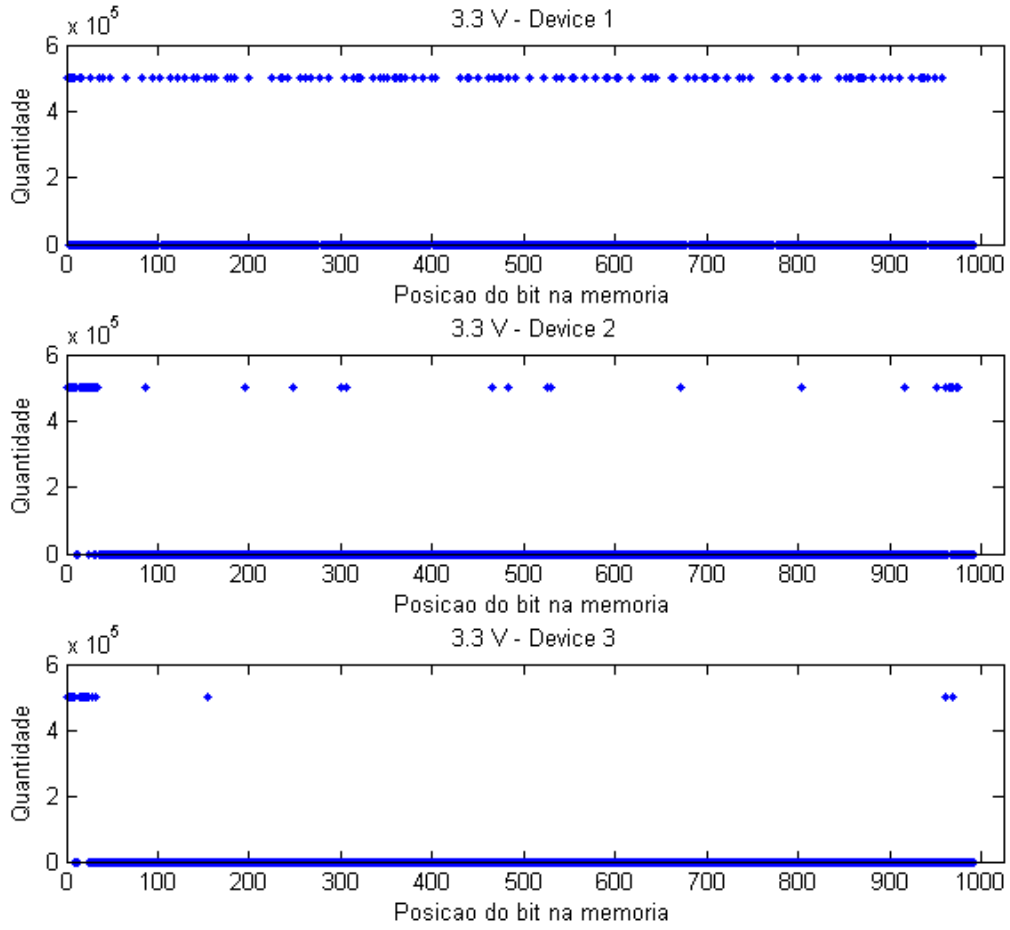


Fig. 39 – Diagrama dos pesos associados à variação de cada bloco de memória SRAM em três placas utilizando microcontrolador MSP430F2013

6 CONCLUSÕES

Neste trabalho foi apresentada a caracterização de um bloco de memória SRAM para implementação de PUFs. Os objetivos iniciais eram:

1 – Caracterizar o funcionamento de blocos de memória SRAM para implementação de PUFs.

2 – Desenvolver um framework que possa ser reutilizado para caracterização de outras figuras de mérito associadas ao funcionamento de PUFs em SRAM.

A caracterização do funcionamento dos blocos de memória (Objetivo 1) foi atingido por meio da análise da resposta de leituras consecutivas da SRAM sob diferentes cenários de tensão de alimentação (Vcc). Aplicaram-se diferentes métodos de comparação entre as amostras adquiridas de forma que uma análise quantitativa da média e da variância da distribuição das distâncias de Hamming intra-classe pudesse ser estimada. Tais métodos proporcionaram uma análise do comportamento do bloco de memória sob um escopo geral, ou seja, durante todo o espaço amostral considerado durante os experimentos, bem como sob um escopo restrito, que avaliou a relação entre experimentos consecutivos. Este estágio do projeto revelou que a média da distribuição das distâncias intra-classe para o dispositivo em análise foi de 3% e não apresentou variação significativa durante a variação de tensão de alimentação no intervalo de 2.6 V a 3.6 V. Aplicou-se também uma outra forma de comparação, que é capaz de demonstrar a quantidade de vezes que uma célula de memória variou durante o experimento. Esta última estratégia revelou que, embora a média da distribuição não seja alterada pela variação da tensão de alimentação, existem células de memória que apresentaram seu comportamento alterado por meio da variação desse parâmetro, isto é, passaram de um estado aleatório para um valor bem definido ou o oposto, quando submetidas a tensões distintas. Um resultado considerado interessante, mas que não foi possível propor uma causa específica foi o fato de que estas células que apresentaram seu comportamento alterado pela variação de tensão representam aproximadamente 3% das amostras, todavia não foi possível levantar razões para que este seja o valor que influencie a média das distribuições das distâncias intra-classe.

Desenvolveu-se um *framework* de aquisição e processamento de dados, definido como Objetivo 2, que se mostrou eficaz para aferir as métricas de interesse. O sistema pode ser reutilizado para mensurar outras características que não foram abordadas neste trabalho. E como sugestão para trabalhos futuros, recomenda-se que seja realizada uma análise de dispositivos diferentes considerando um espaço amostral suficiente para que as medidas sejam estatisticamente confiáveis de forma a demonstrar as propriedades inter-classe de blocos de memória SRAM para implementação de PUFs.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Pappu, Ravikanth, et al. "Physical one-way functions." *Science* 297.5589 (2002): 2026-2030.

[2] Maes, Roel, and Ingrid Verbauwhede. "Physically unclonable functions: A study on the state of the art and future research directions." *Towards Hardware-Intrinsic Security*. Springer Berlin Heidelberg, 2010. 3-37.

[3] Katzenbeisser, Stefan, et al. "PUFs: Myth, fact or busted? A security evaluation of physically unclonable functions (PUFs) cast in silicon."*Cryptographic Hardware and Embedded Systems–CHES 2012*. Springer Berlin Heidelberg, 2012. 283-301.

[4] Bohm, C., Maximilian Hofer, and Wolfgang Pribyl. "A microcontroller sram-puf."*Network and System Security (NSS), 2011 5th International Conference on*. IEEE, 2011.

[5] Guajardo, Jorge, et al. "FPGA intrinsic PUFs and their use for IP protection."*Cryptographic Hardware and Embedded Systems-CHES 2007*. Springer Berlin Heidelberg, 2007. 63-80.

[6] Hamming, Richard W. "Error detecting and error correcting codes." *Bell System technical journal* 29.2 (1950): 147-160.

[7] Gassend, Blaise LP. *Physical random functions*. Diss. Massachusetts Institute of Technology, 2003.

[8] Gassend, Blaise, et al. "Silicon physical random functions." *Proceedings of the 9th ACM conference on Computer and communications security*. ACM, 2002.

[9] Vivekraja, Vignesh, and Leyla Nazhandali. "Circuit-level techniques for reliable Physically Uncloneable Functions." *Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on*. IEEE, 2009.

[10] Suh, G. Edward, and Srinivas Devadas. "Physical unclonable functions for device authentication and secret key generation." *Proceedings of the 44th annual Design Automation Conference*. ACM, 2007.

[11] Tolk, K.: Reflective particle technology for identification of critical components. Tech.
Rep. SAND-92-1676C, Sandia National Labs, Albuquerque, NM (1992)

[12] Bauder, D. W. "An anti-counterfeiting concept for currency systems." *Sandia National Labs, Albuquerque, NM, Tech. Rep. PTK-11990* (1983).

[13] Commission on Engineering and Technical Systems (CETS): Counterfeit Deterrent Features for the Next-Generation Currency Design. The National Academic Press (1993). Appendix E

[14] Buchanan, J.D.R., Cowburn, R.P., Jausovec, A.V., Petit, D., Seem, P., Xiong, G., Atkinson, D., Fenton, K., Allwood, D.A., Bryan, M.T.: Forgery: `_ngerprinting' documents and packaging. Nature 436(7050), 475 (2005)

[15] Hammouri, Ghaith, Aykutlu Dana, and Berk Sunar. "CDs have fingerprints too."*Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer Berlin Heidelberg, 2009. 348-362.

[16] Indeck, Ronald S., and Marcel W. Muller. "Method and apparatus for fingerprinting magnetic media." U.S. Patent No. 5,365,586. 15 Nov. 1994.

[17] *MagneTek(R): MagnePrint(R). http://www.magneprint.com/*

[18] Vrijaldenhoven, Serge, et al. "Acoustical physical uncloneable functions." (2004).

[19] DeJean, Gerald, and Darko Kirovski. "RF-DNA: Radio-frequency certificates of authenticity." *Cryptographic Hardware and Embedded Systems-CHES 2007*. Springer Berlin Heidelberg, 2007. 346-363.

[20] Lofstrom, Keith, W. Robert Daasch, and Donald Taylor. "IC identification circuit using device mismatch." *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International*. IEEE, 2000.

[21] Helinski, Ryan, Dhruva Acharyya, and Jim Plusquellic. "A physical unclonable function defined using power distribution system equivalent resistance variations." *Proceedings of the 46th Annual Design Automation Conference*. ACM, 2009.

[22] Tuyls, Pim, et al. "Read-proof hardware from protective coatings." *Cryptographic Hardware and Embedded Systems-CHES 2006*. Springer Berlin Heidelberg, 2006. 369-383.

[23] Skoric, Boris, et al. "Information-theoretic analysis of capacitive physical unclonable functions." *Journal of Applied physics* 100.2 (2006): 024902-024902.

[24] Guajardo, Jorge, et al. "Anti-counterfeiting, key distribution, and key storage in an ambient world via physical unclonable functions." *Information Systems Frontiers* 11.1 (2009): 19-41.

[25] Lim, Daihyun, et al. "Extracting secret keys from integrated circuits." *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 13.10 (2005): 1200-1205.

[26] Lee, Jae W., et al. "A technique to build a secret key in integrated circuits for identification and authentication applications." *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on*. IEEE, 2004.

[27] Kumar, Sandeep S., et al. "The butterfly PUF protecting IP on every FPGA."*Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*. IEEE, 2008.

[28] Su, Ying, Jeremy Holleman, and Brian Otis. "A 1.6 pJ/bit 96% stable chip-ID generating circuit using process variations." *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*. IEEE, 2007.

[29] NXP Semiconductor, PUF – Physical Unclonable Functions - Protecting next-generation Smart Card ICs with SRAM-based PUFs, Technical Report, 2013

[30] Integrated Circuit Engineering Corporation, SRAM Technology.

[31] Holcomb, Daniel E., Wayne P. Burleson, and Kevin Fu. "Power-up SRAM state as an identifying fingerprint and source of true random numbers." *Computers, IEEE Transactions on* 58.9 (2009): 1198-1210.

[32] Texas Instruments, *MSP430F2013 Datasheet*

[33] Texas Instruments, *MSP430X2XXX Datasheet.*

[34] Ferres, Leo. "Memory management in C: The heap and the stack." *Department of Computer Science, Universidad de Concepcion* (2010).

[35] Sampaio, Arthur M., Medeiros, José E. G.. “A Framework for assessing the use of SoC SRAMs as Physically Unclonable Functions”. Chip in Curitiba, SForum 2013.

ANEXOS

Para informações detalhadas sobre a disposição dos arquivos apresentados a seguir nos diretórios do projeto favor consultar o CD em anexo.

Implementação do sistema de aquisição e processamento de dados.

puf.c

```
/*
 * identify.c
 *
 *  Created on: Apr 13, 2013
 *      Author: Arthur
 */

#include <puf.h>

#define DEBUG

int main(int argc, char* argv[])
{

	int i = 0;
	int iMaskPresent = 0;
	int iNumberDumpsPresent = 0;

	char line[1024];
	char sFileNamePatternIndividual[256];
	char sFileNamePatternGlobal[256];

	memset(gsDifference, 0x00, sizeof(gsDifference));
	memset(gsTempDiff, 0x00, sizeof(gsTempDiff));

	TraceFileLog(sLogFileName, MSGS,

	"================================================================================
=====\n");
	TraceFileLog(sLogFileName, MSGS, "PUF STARTING \n");

	if (iDebugDetail == 1)
	{
		bTraceLogsEnabled = 1;
	}

#ifdef __WINDOWS__
	system("cls");
#endif

#ifdef __LINUX__
	sytem("clear");
#endif

	if (argc <= 1)
	{
```

```
            printUsage();

            //printf("\nPress Enter to Start Serial Comm with MSP 430 Board or X
to quit. \n\n");

            int iDumpsQtyRequest;
            int count = 0;

            memset(sFileNamePatternSave, 0x00, sizeof(sFileNamePatternSave));
            printf("What is the file name mask that you wanna use? \n");
            scanf("%s", sFileNamePatternSave);
            printf("Ok, %s is the mask of the file name. \n\n",
                        sFileNamePatternSave);

            printf("How many memory dumps would you like to perform? \n");
            scanf("%d", &iDumpsQtyRequest);

            printf("Ok, %d dumps to be perfomed. Starting now. \n",
                        iDumpsQtyRequest);

            //int cKeyPressed;
            while (count < iDumpsQtyRequest)
            {
                  //printf("cKeyPressed = [%d]\n", cKeyPressed);

                  /*if (cKeyPressed == 120)
                   {
                   //printf("cKeyPressed = [%d]\n", cKeyPressed);
                   return 0;
                   }*/

                  printf("Dump number: [%d] - Starting Comm... \n", count);
                  puf_msp430_uart(); // O resultado desta funcao vem em
sMemPrintBuffer
                  //printf("\nPress Enter to Start Serial Comm with MSP 430 Board
or X to quit. \n\n");
                  //printf("ESTOU AQUI! sMemPrintBuffer[%d] na saida da
puf_msp430_uart: [%s]",strlen(sMemPrintBuffer),sMemPrintBuffer);

                  //unsigned short vcc = (sMemPrintBuffer[MEM_SIZE-1] << 8) |
sMemPrintBuffer[MEM_SIZE-2];
                  //unsigned char lowVcc = sMemPrintBuffer[MEM_SIZE-2];
                  //unsigned char highVcc = sMemPrintBuffer[MEM_SIZE-1];

                  //Start Saving dump into file mask given.

                  memset(sFileNamePatternGlobal, 0x00,
                              sizeof(sFileNamePatternGlobal));
                  memset(sFileNamePatternIndividual, 0x00,
                              sizeof(sFileNamePatternIndividual));

                  count++;

                  sprintf(sFileNamePatternIndividual, "%s%d.dat",
                              sFileNamePatternSave, count);

                  sprintf(sFileNamePatternGlobal, "%s_global.dat",
                              sFileNamePatternSave);

                  memset(line, 0x00, sizeof(line));
```

```
                    FILE * fpGlobal = fopen(sFileNamePatternGlobal, "a");
                    if (fpGlobal != NULL )
                    {
                          //Preparar o buffer calculando temperatura e vcc para
guardar no arquivo
                          sprintf(line, "[%3d] - [%s] \n", count,
sMemPrintBuffer);
                          fputs(line, fpGlobal);
                          fclose(fpGlobal);
                    }

                    memset(line, 0x00, sizeof(line));

                    char * pch;
                    char * buffer = str_replace(sMemPrintBuffer, " ", "\n");

                    pch = strchr(buffer, ' ');
                    while (pch != NULL )
                    {
                          buffer = str_replace(buffer, " ", "\n");
                          pch = strchr(buffer, ' ');
                    }

                    //Montar nome do arquivo atual
                    FILE *fp = fopen(sFileNamePatternIndividual, "w+");
                    if (fp != NULL )
                    {
                          sprintf(line, "MEM_DUMP_FILE_UNB_Number[%d]\n", count);
                          fputs(line, fp);
                          fputs(buffer, fp);
                          fputs("\n", fp);
                          fclose(fp);
                    }

              }

              printf("[%d] - Dumps complete. \n", count);
              _getch();
              return 0;

        }
        else
        {

              for (i = 0; i < argc - 1; i++)
              {
                    bConsoleLogsEnabled = 0;
                    TraceFileLog(sLogFileName, MSGS,
                                "Comparing Parameters: [%d] - [%s] \n", i, argv[i
+ 1]);
                    bConsoleLogsEnabled = 1;
                    if (strcmp("-mask", argv[i]) == 0 && iMaskPresent == 0)
                    {
                          iMaskPresent = 1;
                          //If user prompted mask of memory_dump_files use his...
otherwise use standard value
                          memset(sFileNamePattern, 0x00,
sizeof(sFileNamePattern));
                          strcpy(sFileNamePattern, argv[i + 1]);

                    }
```

```
                    if (strcmp("-n", argv[i]) == 0 && iNumberDumpsPresent == 0)
                    {
                          iNumberDumpsPresent = 1;
                          //If user prompted number of memory_dump_files use
his... otherwise use standard value
                          iMemDumpQty = atoi(argv[i + 1]);
                    }

                    if (strcmp("-debug", argv[i]) == 0)
                    {
                          iDebugDetail = 1;
                    }

             }

      }

      if (iMaskPresent == 0)
      {
             //If user prompted mask of memory_dump_files use his... otherwise use
standard value
             memset(sFileNamePattern, 0x00, sizeof(sFileNamePattern));
             strcpy(sFileNamePattern, "msp430_startup_memdump");
      }

      if (iNumberDumpsPresent == 0)
      {
             //If user prompted mask of memory_dump_files use his... otherwise use
standard value
             iMemDumpQty = 2;
      }

      sram_puf_gen(iMemDumpQty);

      return 0;

}

void printUsage()
{
      TraceFileLog(sLogFileName, MSGS,

      "================================================================================
=====\n");
      TraceFileLog(sLogFileName, MSGS,
                    "    SRAM PUF Framework - Based on MSP430 Memory DUMP \n\n");
      TraceFileLog(sLogFileName, MSGS,
                    "    University of Brasilia - Electrical Engineering Department
\n");
      TraceFileLog(sLogFileName, MSGS,
                    "    Arthur Morales Sampaio - Jose Edil Guimaraes de
Medeiros\n");
      TraceFileLog(sLogFileName, MSGS,
                    "    This program intends to gather and compare startup memory
dumps from MSP430F2013 \n");
      TraceFileLog(sLogFileName, MSGS,
                    "    and work with the Hamming distances of the Challenges (mem
dumps) \n");
      TraceFileLog(sLogFileName, MSGS, "    \n");
      TraceFileLog(sLogFileName, MSGS,

      "================================================================================
```

```
=====\n");
	TraceFileLog(sLogFileName, MSGS,
			" HOW TO USE: \n Save the startup memory dump of the MCU in the
TI format.\n\n");
	TraceFileLog(sLogFileName, MSGS,
			" Repeat the process N times depending on the precision you
want. \n");
	TraceFileLog(sLogFileName, MSGS,
			" Suppose you saved all the files with the names: \n\n");
	TraceFileLog(sLogFileName, MSGS, "    -startup_memory_dump1.dat \n");
	TraceFileLog(sLogFileName, MSGS, "    -startup_memory_dump1.dat \n");
	TraceFileLog(sLogFileName, MSGS, "    -startup_memory_dump2.dat \n");
	TraceFileLog(sLogFileName, MSGS, "    ...\n");
	TraceFileLog(sLogFileName, MSGS, "    -startup_memory_dumpN.dat \n\n");
	TraceFileLog(sLogFileName, MSGS,
			" Then you need to start this program using the following
parameters: \n\n");
	TraceFileLog(sLogFileName, MSGS,
			"      puf-msp430 -mask startup_memory_dump -n N \n\n");
	TraceFileLog(sLogFileName, MSGS,
			" NOTE: DO NOT INCLUDE THE NUMBER OF THE FILENAME in the
parameter mask neither the FILE FORMAT, only the common name. \n\n");
	TraceFileLog(sLogFileName, MSGS,
			" Replace N with the number of dumps you need to include to
generate the PUF. \n");
	TraceFileLog(sLogFileName, MSGS,

	"================================================================================
=====\n");

}

void sram_puf_gen(int iMemDumpQty)
{

	int i, j, iResultLen = 0;

	unsigned char usHex[512];

	char sFiles[iMemDumpQty][256];

	memset(usHex, 0x00, sizeof(usHex));

	printUsage();

	printf(
			"REMEMBER TO REMOVE THE HISTOGRAM DATA FILE BEFORE PERFORMING
THIS! \n\n");

	printf(
			"Which mode do you want to use for calculating Hamming
Distribution? \n 0 - BRUTE FORCE - ALL COMBINATIONS \n 1 - FIRST DUMP AS REFERENCE
\n 2 - COMPARE CONSECUTIVE DUMPS\n Option: ");
	scanf("%d", &iModeComputeHamming);

	printf("Ok, Mode %d selected. \n", iModeComputeHamming);
	printf("The process will start... Press ENTER\n");

	_getch();

	system("cls");
```

```
	/*From now on Only log stuff in logFile!!*/
	bConsoleLogsEnabled = 0;

	memset(sFiles, 0x00, sizeof(sFiles));

	// We need to replace the code ABOVE for a routine that checks a folder or
receives the parameter of the number of memory dumps
	TraceFileLog(sLogFileName, MSGS,
			"[%04d] %s Start processing memory dump files... \n", __LINE__,
			__FUNCTION__);

	TraceFileLog(sLogFileName, MSGS,
			"[%04d] %s Memory dump files to process: [%d] \n", __LINE__,
			__FUNCTION__, iMemDumpQty);

	for (i = 0; i < iMemDumpQty; i++)
	{
		sprintf(sFiles[i], "%s%d.dat", sFileNamePattern, (i + 1));
	}

	/* This function is supposed to gather memory dump information from a .dat
file and compare which bits are equal and what is their position in the file */
	/* Stores into gsTempDiff the xor of the two files. Which means sets to 1
the positions where the bits are different.
	 * ATTENTION! You can ONLY use the buffer gsTempDiff AFTER THE CALL OF THIS
FUNCTION OTHERWISE YOU WILL TRY TO USE
	 * AN UNALLOCATED POINTER!
	 */

	iCombinations = iMemDumpQty * (iMemDumpQty - 1) / 2;

	switch (iModeComputeHamming)
	{
	case 0: //FORCA BRUTA!

		memset(giWeightsMode2, 0x00, sizeof(giWeightsMode2));

		for (i = 1; i <= (iMemDumpQty - 1); i++)
		{
			for (j = 1; j <= iMemDumpQty; j++)
			{
				if (j <= i)
				{
					continue;
				}

				TraceFileLog(sLogFileName, MSGS,
						"[%04d] %s STARTING MEMORY FILES
COMBINATION: [%d][%d] \n",
						__LINE__, __FUNCTION__, i, j);

				if (iDebugDetail == 1)
				{
					bTraceLogsEnabled = 1;
				}
				iResultLen = getMemFiles(sFiles[i - 1], sFiles[j - 1]);
				if (iDebugDetail == 1)
				{
					bTraceLogsEnabled = 0;
				}
			}
```

```
            }

            break;

      case 1: //FIRST DUMP AS REFERENCE

            for (j = 1; j <= iMemDumpQty; j++)
            {
                  if (j == 1)
                  {
                        continue;
                  }

                  TraceFileLog(sLogFileName, MSGS,
                              "[%04d] %s STARTING MEMORY FILES COMBINATION:
[%d][%d] \n",
                              __LINE__, __FUNCTION__, i, j);

                  if (iDebugDetail == 1)
                  {
                        bTraceLogsEnabled = 1;
                  }
                  printf("GET MEM FILES [%s] [%s]", sFiles[0], sFiles[j - 1]);
                  iResultLen = getMemFiles(sFiles[0], sFiles[j - 1]);
                  if (iDebugDetail == 1)
                  {
                        bTraceLogsEnabled = 0;
                  }
            }

            break;

      case 2: //ADJACENTES
            for (j = 1; j < iMemDumpQty; j++)
            {
                  /*if (j == 1)
                   {
                   continue;
                   }*/

                  TraceFileLog(sLogFileName, MSGS,
                              "[%04d] %s STARTING MEMORY FILES COMBINATION:
[%d][%d] \n",
                              __LINE__, __FUNCTION__, i, j);

                  if (iDebugDetail == 1)
                  {
                        bTraceLogsEnabled = 1;
                  }
                  //printf("GET MEM FILES [%s] [%s]",sFiles[j],sFiles[j-1]);
                  iResultLen = getMemFiles(sFiles[j], sFiles[j - 1]);
                  if (iDebugDetail == 1)
                  {
                        bTraceLogsEnabled = 0;
                  }
            }
            break;
      }

      //usHex = (unsigned char*) calloc(1, iResultLen * sizeof(char));
      giHammingDistance = 0;
      for (i = 0; i < iResultLen; i++)
```

```
	{
		sprintf(usHex + strlen(usHex), "%02X ", gsDifference[i]);

		for (j = 0; j < 8; j++)
		{
			if ((gsDifference[i] & (0x01 << j)) != 0)
			{
				//If the XOR of the two bits being tested is ONE it
means it has changed from one dump between the other.
				// This will be useful to calculate the Hamming Distance
between two memory dumps and so on...
				// At the end we will be able to Calculate the Intra-
class measurements parameters referenced on the paper
				// "PUF Physically Unclonable Functions:     a Study on
the State of the Art and Future Research Directions."

				// This is very important! It calculates the Hamming
Distance!
				giHammingDistance++;

			}

		}
	}

	bConsoleLogsEnabled = 1;
	TraceFileLog(sLogFileName, MSGS,
			"[%04d] %s Succesfully Finished! \n\n The Maximum Changed BIT
MASK is [%s] (0 = Unchanged BIT, 1 = Bit changed at least Once) \n\n Maximum TOTAL
number of Bits Changed: %d \n RESULTS SAVED IN GLOBAL FILE!\n",
			__LINE__, __FUNCTION__, usHex, giHammingDistance);

	char bufferMask[2048];

	memset(bufferMask, 0x00, sizeof(bufferMask));
	memset(sFileNamePatternSave, 0x00, sizeof(sFileNamePatternSave));
	sprintf(sFileNamePatternSave, "%s_global.dat", sFileNamePattern);

	TraceFileLog(sLogFileName, MSGS, "FILE GLOBAL NAME[%s]",
			sFileNamePatternSave);

	FILE * fpGlobal = fopen(sFileNamePatternSave, "a");
	if (fpGlobal != NULL )
	{
		sprintf(bufferMask, "[MSK] - [%s]\n", usHex);
		fputs(bufferMask, fpGlobal);
		fclose(fpGlobal);
	}

	char strAux[128];
	char auxWeightsFileName[128];
	memset(strAux, 0x00, sizeof(strAux));

	if (iModeComputeHamming == 0)
	{

		memset(auxWeightsFileName, 0x00, sizeof(auxWeightsFileName));
			sprintf(auxWeightsFileName, "%s_weightsdata_mode%d.dat",
sFileNamePattern,
					iModeComputeHamming);
		FILE * fpGlobal = fopen(auxWeightsFileName, "a");
		if (fpGlobal != NULL )
```

```
		{
			for (i = 0; i < (MEM_SIZE - STACKBYTES) * 8; i++)
			{
				sprintf(strAux, "%d\n", giWeightsMode2[i]);
				fputs(strAux, fpGlobal);
			}

		}
		fclose(fpGlobal);
	}



	bConsoleLogsEnabled = 0;
}

int getMemFiles(char * file1, char * file2)
{

	unsigned char bufferBin1[2048];
	unsigned char bufferBin2[2048];

	memset(bufferBin1, 0x00, sizeof(bufferBin1));
	memset(bufferBin2, 0x00, sizeof(bufferBin2));
	int iLenBufferBin1 = 0, iLenBufferBin2 = 0;
	int i = 0, j = 0;

	char line[1024];
	char auxHistFileName[1024];

	TraceFileLog(sLogFileName, MSGS,
			"[%04d] %s Opening files in groups of two... \n", __LINE__,
			__FUNCTION__);

	TraceFileLog(sLogFileName, MSGS,
			"[%04d] %s START FETCHING DATA FROM File1 = %s \n", __LINE__,
			__FUNCTION__, file1);

	if (iDebugDetail == 1)
	{
		bTraceLogsEnabled = 1;
	}
	getMemoryDataFromFile(file1, bufferBin1, &iLenBufferBin1);

	TraceFileLog(sLogFileName, MSGS,
			"[%04d] %s START FETCHING DATA FROM File2 = %s \n", __LINE__,
			__FUNCTION__, file2);

	getMemoryDataFromFile(file2, bufferBin2, &iLenBufferBin2);
	if (iDebugDetail == 1)
	{
		bTraceLogsEnabled = 0;
	}

	TraceFileLog(sLogFileName, MSGS,
			"[%04d] %s ALLOCATING Pointer to STORE the difference between
the two last files processed. \n",
			__LINE__, __FUNCTION__, file2);

	//gsTempDiff = (char*) calloc(1, iLenBufferBin1 * sizeof(char) + 1); //
Assume all memory dumps have the same number of elements.
	//gsDifference = (char*) calloc(1, iLenBufferBin1 * sizeof(char) + 1); //
```

```
Assume all memory dumps have the same number of elements.

	TraceFileLog(sLogFileName, MSGS,
			"[%04d] %s gsTempDiff=[ALLOCATED] size=[%d] \n", __LINE__,
			__FUNCTION__, iLenBufferBin1);

	if (iDebugDetail == 1)
	{
		bTraceLogsEnabled = 1;
	}
	else
	{
		bTraceLogsEnabled = 0;
	}

	TraceFileLog(sLogFileName, MSGS,
			"[%04d] %s Bit Mask - Difference VECTOR: \n", __LINE__,
			__FUNCTION__);

	for (i = 0; i < iLenBufferBin1; i++)
	{
		TraceFileLog(sLogFileName, MSGS, "[%04d] %s Starting Comparison...
\n",
				__LINE__, __FUNCTION__);

		TraceFileLog(sLogFileName, MSGS,
				"[%04d] %s Starting Comparison... [%d] bufferBin1=[%02X]
len=[%d] bufferBin2=[%02X] len=[%d] \n",
				__LINE__, __FUNCTION__, i, bufferBin1[i],
iLenBufferBin1,
				bufferBin2[i], iLenBufferBin2);

		gsTempDiff[i] = (bufferBin1[i]) ^ (bufferBin2[i]);
		//Iterate xor'ing the bits from the two files and storing in
gsTempDiff

		TraceFileLog(sLogFileName, MSGS,
				"[%04d] %s Bit Mask - Difference VECTOR: [%d] - [%02X]
\n",
				__LINE__, __FUNCTION__, i, gsTempDiff[i]);

		for (j = 0; j < 8; j++)
		{
			if ((gsTempDiff[i] & (0x01 << j)) != 0)
			{
				//If the XOR of the two bits being tested is ONE it
means it has changed from one dump between the other.
				// This will be useful to calculate the Hamming Distance
between two memory dumps and so on...
				// At the end we will be able to Calculate the Intra-
class measurements parameters referenced on the paper
				// "PUF Physically Unclonable Functions:     a Study on
the State of the Art and Future Research Directions."

				TraceFileLog(sLogFileName, MSGS,
						"[%04d] %s Difference Bits Caught: [%d]
\n", __LINE__,
						__FUNCTION__, i);

				// This is very important! It calculates the Hamming
Distance!
				giHammingDistance++;
```

```
                }

            }

        }

        bTraceLogsEnabled = 1;
        bConsoleLogsEnabled = 1;
        TraceFileLog(sLogFileName, MSGS,
                "[%04d] %s Hamming Distance Intra-class measure: [%d] \n",
__LINE__,
                __FUNCTION__, giHammingDistance);

        memset(auxHistFileName, 0x00, sizeof(auxHistFileName));
        sprintf(auxHistFileName, "%s_histogramdata_mode%d.dat", sFileNamePattern,
                iModeComputeHamming);
        //Montar nome do arquivo atual
        FILE *fp = fopen(auxHistFileName, "a+");
        if (fp != NULL )
        {
            sprintf(line, "%d\n", (int) giHammingDistance);
            fputs(line, fp);
            fclose(fp);
        }

        bConsoleLogsEnabled = 0;
        giHammingDistance = 0;

        memset(line, 0x00, sizeof(line));

        char * pch;
        char * buffer = str_replace(sMemPrintBuffer, " ", "\n");

        pch = strchr(buffer, ' ');
        while (pch != NULL )
        {
            buffer = str_replace(buffer, " ", "\n");
            pch = strchr(buffer, ' ');
        }

        for (i = 0; i < iLenBufferBin1; i++)
        {

            gsDifference[i] |= gsTempDiff[i];

            if (iModeComputeHamming == 0)
            {
                //Alem de calcular os Hammings da comparacao FORCA BRUTA dos
dumps tem que calcular o vetor com os pesos!!!!
                for (j = 0; j < 8; j++)
                {
                    //printf("Testando Bits da mascara de diff. %02X i=%d
j=%d\n",
                    //    gsDifference[i], i, j);
                    if (gsDifference[i] & (0x01 << j))
                    {
                        giWeightsMode2[(i + 1) * 8 - (j + 1)]++;
                        //printf("Incrementando mascaraBitPos [%d]\n", i*8
+ j);
                    }
                }
```

```
                        //system("pause");
                        //printf("Position %d da mascara tem 1 \n", i);

                }

                if (gsDifference[i] > 1)
                {
                        /*TraceFileLog(sLogFileName, MSGS,
                         "[%04d] %s When there is Difference: gsDifference[%02X]
gsTempDiff[%02X]\n",
                         __LINE__, __FUNCTION__, gsDifference[i], gsTempDiff[i]);*/
                }

        }

        //Since the functions allocated the buffers we need to free them.
        //free(bufferBin1);
        //free(bufferBin2);
        //free(gsTempDiff);
        return iLenBufferBin1;
}

void getMemoryDataFromFile(char * file, unsigned char * usRetBufferBin,
                int * piLenRetBufferBin)
{

        unsigned char buffer[2048];
        char line[80];
        //long length = 0;
        //char * pch = 0;
        int iLenBufferBin = 0;
        unsigned char sBufferBinAux[2048];
        int res = 0, i = 0;

        memset(sBufferBinAux, 0x00, sizeof(sBufferBinAux));
        memset(buffer, 0x00, sizeof(buffer));
        memset(line, 0x00, sizeof(line));

        FILE * h = fopen(file, "r");

        fgets(line, 80, h);

        while (fgets(line, 80, h) != NULL )
        {

                if (line[0] == '\0' || line[0] == '\n' || line[1] == '\n'
                                || line[0] == '\r' || line[1] == '\r')
                        continue;

                buffer[i++] = (unsigned char) line[0];
                buffer[i++] = (unsigned char) line[1];
        }

        fclose(h);

        // Start to process buffer
        bConsoleLogsEnabled = 1;
        TraceFileLog(sLogFileName, MSGS,
                        "[%04d] %s Processing stream of bits... buffer_size[%d] \n",
                        __LINE__, __FUNCTION__, strlen(buffer));

        TraceFileLog(sLogFileName, MSGS, "[%04d] %s Buffer Hex: [%s] %s\n",
```

```
                    __LINE__, __FUNCTION__, buffer, file);

        bConsoleLogsEnabled = 0;

        if (strlen(buffer) == 0)
                system("pause");

        res = convAsciiHexBin(buffer, strlen(buffer), sBufferBinAux,
                        &iLenBufferBin);

        /*TraceFileLog(sLogFileName, MSGS, "[%04d] %s Buffer Binary: \n",
         __LINE__, __FUNCTION__);

         for (i = 0; i < iLenBufferBin; i++)
         {
         TraceFileLog(sLogFileName, MSGS, "Buffer Binary [%d]:  %02X \n", i,
         sBufferBinAux[i]);
         }*/

        memcpy(usRetBufferBin, sBufferBinAux, iLenBufferBin);
        *piLenRetBufferBin = iLenBufferBin;
}

char * removeFirstLine(char * buffer)
{
        char * pch;
        char * sDataAux = (char*) calloc(1, strlen(buffer));
        int pos;

        pch = strchr(buffer, '\n'); //Locate first occurrence of \n to get position
of second line start.

        pos = (int) (pch - buffer + 1);

        memcpy(sDataAux, buffer + pos, strlen(buffer) - pos);

        return sDataAux;

}

void puf_msp430_uart()
{
        int dsr = 0;
        HANDLE comPort;
        SERIAL_CFG config;

        unsigned char memDumpSerialData[256];

        int bitCount = 0;
        int bytesCounter = 0;
        int i = 0;

        unsigned char rxByte = 0x00;

        int rc = 0;

        memset(memDumpSerialData, 0x00, sizeof(memDumpSerialData));
        memset(sMemPrintBuffer, 0x00, sizeof(sMemPrintBuffer));

        memset(&config, 0x00, sizeof(SERIAL_CFG));

        sprintf(config.device, "COM2");
```

```
	//config.baudrate = 115200;
	config.baudrate = 115200;
	config.parity = 'E';
	config.databits = 8;
	config.stopbits = 1;
	config.flowctl = 'N';

	rc = OpenSerialComm(&comPort, &config);

	//_getch();
	puf_msp430_setVccResetPin(comPort, 0);
	//_getch();
	//printf("Sleeping...\n");
	Sleep(5000); // Sleep 0.5 sec
	//printf("Done Sleeping...\n");

#ifdef DEBUG
	TraceFileLog(sLogFileName, MSGS, "Turning On MCU Board. \n");
#endif
	puf_msp430_setVccResetPin(comPort, 1);

	//Initialize
	//SetRTSDTR(&comPort, 0, 0, &config);
	rc = puf_msp430_resetPins(comPort);
	if (rc < 0)
	{
#ifdef DEBUG
		TraceFileLog(sLogFileName, MSGS, "Error RESETING COMM Exiting...\n");
#endif

		return;
	}

	while (1)
	{

		if (bitCount >= 8)
		{
			//Ultimo bit do byte foi lido.

			bitCount = 0;


	//memcpy(memDumpSerialData+bytesCounter,&rxByte,sizeof(rxByte));
			memDumpSerialData[bytesCounter] = rxByte;

#ifdef DEBUG
			TraceFileLog(sLogFileName, MSGS, "Byte [%d] - %02X \n",
				bytesCounter, memDumpSerialData[bytesCounter]);
#endif

			rxByte = 0x00;
			bytesCounter++;

		}

		if (bytesCounter >= MEM_SIZE)
		{
			break;
		}

		rc = puf_msp430_getDataAvailablePin(comPort, &dsr);
```

```
            if (rc != 0)
            {
#ifdef DEBUG
                  TraceFileLog(sLogFileName, MSGS, "Error Reading DataAVL
pin.\n");
#endif
            }

            if (dsr)
            {
#ifdef DEBUG
                  TraceFileLog(sLogFileName, MSGS, "Data Available Pin On. [%d]
\n",
                              dsr);
#endif
                  //There is data to be read.

                  puf_msp430_readDataBit(comPort, &rxByte, bitCount);
                  bitCount++;

                  //SET Rts Pin to tell MCU that I read.
                  //printf("Set DataReaden Pin? (Enter)\n");
                  //_getch();

                  puf_msp430_setDataReadenPin(comPort, 1);

                  //Wait for MCU to Drop Data Available Pin

                  rc = puf_msp430_getDataAvailablePin(comPort, &dsr);

                  if (dsr == 0)
                  {
#ifdef DEBUG
                        TraceFileLog(sLogFileName, MSGS,
                                    "Data Available Pin Dropped. [%d] \n",
dsr);
#endif

                        //SET Rts Pin to tell MCU that I read.
                        //printf("Drop DataReaden Pin? (Enter)\n");
                        //_getch();

                        puf_msp430_setDataReadenPin(comPort, 0);

#ifdef DEBUG
                        TraceFileLog(sLogFileName, MSGS, "Dropped Data Readen
PIN! \n");
#endif


                  }

            }
            else
            {
#ifdef DEBUG
                  TraceFileLog(sLogFileName, MSGS, "Data Available Pin Off. [%d]
\n",
                              dsr);
#endif

            }
            //Sleep(500);
```

```
            /*
             if (bytesCounter >= MEM_SIZE - 1)
             {
             for (i = 0; i < MEM_SIZE; i++)
             {
             sprintf(sMemPrintBuffer + strlen(sMemPrintBuffer), "%02X ",
             memDumpSerialData[i]);
             }
             TraceFileLog(sLogFileName, MSGS,
             "Memory dump size[%d] i[%d] Readen: %s", sMemPrintBuffer,
             MEM_SIZE, i);
             break;
             }*/

      }

      for (i = 0; i < MEM_SIZE - 4 /*Porque tem 4 bytes para Vcc e Temperatura*/;
                  i++)
      {
            //printf("Pos: [%d]",strlen(sMemPrintBuffer));
            sprintf(sMemPrintBuffer + strlen(sMemPrintBuffer), "%02X ",
                        memDumpSerialData[i]);
            //printf(" Str: %s", sMemPrintBuffer);
            //printf("\n");
      }
      TraceFileLog(sLogFileName, MSGS, "Memory dump size[%d] Readen: %s",
                  MEM_SIZE, sMemPrintBuffer);

      iVcc = (memDumpSerialData[MEM_SIZE - 1] << 8)
                  | memDumpSerialData[MEM_SIZE - 2];
      iTemperature = (memDumpSerialData[MEM_SIZE - 3] << 8)
                  | memDumpSerialData[MEM_SIZE - 4];

      fVcc = calculaVcc(iVcc);
      fTemperature = calculaTemp(iTemperature);

      TraceFileLog(sLogFileName, MSGS, "Vcc = %f [%4X] - Temp = %f[%4X] \n", fVcc,
                  iVcc, fTemperature, iTemperature);

      CloseSerialComm(&comPort, &config);

}

float calculaTemp(int iTemp)
{
      // Implementar formula
      // [(2^(-16)*0.6*iTemp)*1000/1.32] - 273
      // Ou alternativamente
      // 0.00693581321022727*iTemp - 273
      return (0.00693581321022727 * iTemp - 273);
}

float calculaVcc(int iVcc)
{
      // Implementar formula
      // 2^(-16)*0.6*11*iVcc
      return (0.0001007080078 * iVcc * 1.184211);
}

int puf_msp430_getDataAvailablePin(HANDLE * hComm, int * dataAvailablePin)
{
      DWORD dwCommEvent;
```

```
	DWORD dwStoredFlags;
	DWORD dsrStatus;
	int rc = 0;

	dwStoredFlags = EV_DSR;

	if (!SetCommMask(hComm, EV_DSR))
	{
		// Error setting communications mask
		return -1;
	}

	if (!WaitCommEvent(hComm, &dwCommEvent, NULL ))
	{
		// An error occurred waiting for the event.
		return -1;
	}
	else
	{
		// Event has occurred.
		rc = GetCommModemStatus(hComm, &dsrStatus);
		if (rc != 0)
		{
			//Read Modem Status OK

			if (dsrStatus & MS_DSR_ON)
			{
				//CTS is ON
				/*TraceFileLog(sLogFileName, MSGS,
				 "Received EVENT CTS is ON!!!!! \n\n");*/
				*dataAvailablePin = 1;
			}
			else
			{
				//CTS is not ON
				//CTS is ON
				/*TraceFileLog(sLogFileName, MSGS,
				 "Received EVENT CTS is OFF!!!! \n\n");*/
				*dataAvailablePin = 0;
			}
			return 0;

		}
		else
		{
#ifdef DEBUG
			//Error reading modem status
			TraceFileLog(sLogFileName, MSGS, "Error Reading CTS Pin. \n");
#endif
		}

	}

	return 0;

}

int puf_msp430_readDataBit(HANDLE * hComm, unsigned char * rxByte, int bitCount)
{
	DWORD ctsStatus;
	int rc;
```

```
	rc = GetCommModemStatus(hComm, &ctsStatus);

	if (ctsStatus & MS_CTS_ON)
	{
#ifdef DEBUG
		//CTS is ON
		TraceFileLog(sLogFileName, MSGS, "CTS is ON!!!!! \n\n");
		//*dataAvailablePin = 1;
#endif

		*rxByte |= (0x01 << bitCount);

#ifdef DEBUG
		TraceFileLog(sLogFileName, MSGS, "%02X - [%d] \n", *rxByte,
bitCount);
#endif
	}
	else
	{
		//CTS is not ON
		//CTS is ON
#ifdef DEBUG
		TraceFileLog(sLogFileName, MSGS, "CTS is OFF!!!! \n\n");
#endif

		*rxByte &= ~(0x01 << bitCount);
#ifdef DEBUG
		TraceFileLog(sLogFileName, MSGS, "%02X - [%d] \n", *rxByte,
bitCount);
#endif
		//*dataAvailablePin = 0;
	}

	return 0;

}

int puf_msp430_resetPins(HANDLE * hComm)
{
	int rc = EscapeCommFunction(hComm, CLRRTS);

	if (rc != 0)
	{
#ifdef DEBUG
		TraceFileLog(sLogFileName, MSGS, "Reset Pins Ok. \n");
#endif

		return 0;
	}
	else
	{
#ifdef DEBUG
		LPTSTR pMsgBuf;
		FormatMessage(
				FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM,
				NULL, GetLastError(), MAKELANGID(LANG_NEUTRAL,
SUBLANG_DEFAULT),
				(LPTSTR) &pMsgBuf, 0, NULL );
		TraceFileLog(sLogFileName, MSGS, "%s", pMsgBuf);
		LocalFree(pMsgBuf);
#endif
```

```
			return -1;
		}
}

int puf_msp430_setDataReadenPin(HANDLE * hComm, int dataReadenPinValue)
{

		int rc = 0;

		switch (dataReadenPinValue)
		{
		case 0:
			rc = EscapeCommFunction(hComm, CLRRTS);

			if (rc != 0)
			{
#ifdef DEBUG
				TraceFileLog(sLogFileName, MSGS, "Data Readen Pin Dropped!
\n");
#endif
				return 0;
			}
			else
			{
#ifdef DEBUG
				LPTSTR pMsgBuf;
				FormatMessage(
					FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM,
					NULL, GetLastError(),
					MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
					(LPTSTR) &pMsgBuf, 0, NULL );
				TraceFileLog(sLogFileName, MSGS, "%s", pMsgBuf);
				LocalFree(pMsgBuf);
#endif
				return -1;
			}
			break;

		case 1:
			rc = EscapeCommFunction(hComm, SETRTS);

			if (rc != 0)
			{
#ifdef DEBUG
				TraceFileLog(sLogFileName, MSGS, "Data Readen Pin SET! \n");
#endif
				return 0;
			}
			else
			{
#ifdef DEBUG
				LPTSTR pMsgBuf;
				FormatMessage(
					FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM,
					NULL, GetLastError(),
					MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
					(LPTSTR) &pMsgBuf, 0, NULL );
				TraceFileLog(sLogFileName, MSGS, "%s", pMsgBuf);
				LocalFree(pMsgBuf);
```

```
#endif
                        return -1;
                }
                break;

        }

        return 0;
}

int puf_msp430_setVccResetPin(HANDLE * hComm, int vccResetPin)
{

        int rc = 0;

        switch (vccResetPin)
        {
        case 0:
                rc = EscapeCommFunction(hComm, CLRDTR);

                if (rc != 0)
                {
#ifdef DEBUG
                        TraceFileLog(sLogFileName, MSGS, "VCC Pin Dropped! \n");
#endif
                        return 0;
                }
                else
                {
#ifdef DEBUG
                        LPTSTR pMsgBuf;
                        FormatMessage(
                                        FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM,
                                        NULL, GetLastError(),
                                        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
                                        (LPTSTR) &pMsgBuf, 0, NULL );
                        TraceFileLog(sLogFileName, MSGS, "%s", pMsgBuf);
                        LocalFree(pMsgBuf);
#endif
                        return -1;
                }
                break;

        case 1:
                rc = EscapeCommFunction(hComm, SETDTR);

                if (rc != 0)
                {
#ifdef DEBUG
                        TraceFileLog(sLogFileName, MSGS, "VCC Pin SET! \n");
#endif
                        return 0;
                }
                else
                {
#ifdef DEBUG
                        LPTSTR pMsgBuf;
                        FormatMessage(
                                        FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM,
                                        NULL, GetLastError(),
```

```
                                MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
                                (LPTSTR) &pMsgBuf, 0, NULL );
                    TraceFileLog(sLogFileName, MSGS, "%s", pMsgBuf);
                    LocalFree(pMsgBuf);
#endif
                    return -1;
                }
                break;

        }

        return 0;
}
```

puf.h

```
/*

 * indentify.h
 *
 *  Created on: Apr 13, 2013
 *      Author: Arthur
 */

#ifndef IDENTIFY_H_
#define IDENTIFY_H_

#include <utils.h>
#include <commserial.h>

#define DUMP_QTY 2

#define MEM_SIZE 128
#define STACKBYTES 4 //Essa variavel representa o numero de bytes da STACK na
memoria. Esses bytes nao devem ser usados para a PUF!

#define sLogFileName "log/puf-msp430.log"
#define sCommLogFileName "log/puf-comm-msp430.log"


char sFileNamePattern[256]; //MAX filename size is 256
char sFileNamePatternSave[256];

unsigned char sMemPrintBuffer[1024]; // Memory dump atual.
unsigned int iTemperature = 0;
unsigned int iVcc = 0;

float fVcc = 0.0;
float fTemperature = 0.0;

int iMemDumpQty = 0;
int iDebugDetail = 0;
```

```
unsigned char gsTempDiff[512];
unsigned char gsDifference[512];

unsigned int giWeightsMode2[4096];

long giHammingDistance = 0;

int iCombinations = 0;

int iModeComputeHamming = 0;
      /* 0 - MODO FORCA BRUTA - COMPARAR TODOS OS DUMPS COMBINADOS 2 a 2
       * 1 - MODO PRIMEIRO DUMP COMO REFERENCIA
       * 2 - MODO COMPARAR APENAS ADJACENTES*/

void sram_puf_gen(int iMemDumpQty);
void printUsage();
int getMemFiles(char * file1, char * file2);
void getMemoryDataFromFile(char * file, unsigned char * usRetBufferBin, int *
piLenRetBufferBin);
char * removeFirstLine(char * buffer);
void gen_combinations_of_two(int N);
void puf_msp430_uart();
int puf_msp430_getDataAvailablePin(HANDLE * hComm, int * dataAvailablePin);
int puf_msp430_readDataBit(HANDLE * hComm, unsigned char * rxByte, int bitCount);
int puf_msp430_resetPins(HANDLE * hComm);
int puf_msp430_setDataReadenPin(HANDLE * hComm, int dataReadenPinValue);
int puf_msp430_setVccResetPin(HANDLE * hComm, int vccResetPin);

float calculaTemp(int iTemp);
float calculaVcc(int iVcc);

#endif
```

commserial.c

```
#include <commserial.h>
#include <time.h>

char * slogUtilscom = "./log/api/commserial_log.log";
//#ifdef __LINUX__
//struct termios ptSerialCfg->oldtio, ptSerialCfg->newtio;
//#else
//static DCB ptSerialCfg->oldtio, ptSerialCfg->newtio; //Deve estar no arquivo
config.c
//#endif
/*
 * Clear To Send end data set ready
 * Param ptSerialCfg:
 *         configuration values.
 * Param devHandle :

 *         return communication handler.
 * Return values :
 *         RETURN_OK - complete
```

```
 *          anything else is error
 */
int OpenSerialComm(HANDLE *devHandle, SERIAL_CFG *ptSerialCfg)
{
  int rc = COMM_NO_HANDLER;
  COMMTIMEOUTS timeouts =  { 0 };
#ifdef __LINUX__
  //Linux Open Comm
  int flags = 0;
  *devHandle = COMM_OPEN_ERROR;
  if (ptSerialCfg == NULL)
  {
    TraceFileLog(slogUtilscom,ERR, "[%04d] %s ER    Config=Null\n", __LINE__,
__FUNCTION__);
    return rc;
  }

  if (ptSerialCfg->device == NULL)
  {
    TraceFileLog(slogUtilscom,ERR, "[%04d] %s ER    config->device=Null\n",
__LINE__, __FUNCTION__);
    return COMM_INVALID_PARAMETER;
  }
  TraceFileLog(slogUtilscom,MSGS, "[%04d] %s WR    Dev[%s,%d,%d,%c,%d,%c]
Starting..\n", __LINE__, __FUNCTION__,ptSerialCfg->device,ptSerialCfg-
>baudrate,ptSerialCfg->databits,ptSerialCfg->parity,ptSerialCfg-
>stopbits,ptSerialCfg->flowctl);
  flags |= O_RDWR | O_NOCTTY;
  if (ptSerialCfg->blocking == 1)
  {
    flags &= ~O_NONBLOCK;
    TraceFileLog(slogUtilscom,MSGS, "[%04d] %s WR    Dev[%s]  Blk flags[%04X]
O_RDWR,O_NOCTTY Openning..\n", __LINE__, __FUNCTION__,ptSerialCfg->device,flags);
  }
  else if (ptSerialCfg->blocking == 0);
  {
    flags |= O_NONBLOCK | O_NDELAY;
    //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s WR    Dev[%s] NoBlk flags[%04X]
O_NONBLOCK,O_NDELAY Openning..\n", __LINE__, __FUNCTION__,ptSerialCfg-
>device,flags);
  }

  *devHandle = open(ptSerialCfg->device, flags);
  if (*devHandle < 0)
  {
    rc = COMM_OPEN_ERROR;
    TraceFileLog(slogUtilscom,ERR, "[%04d] %s ER    Dev[%s]       flags[%04X]
h[%04d] Fails to Open rc[%d]\n", __LINE__, __FUNCTION__,ptSerialCfg-
>device,flags,*devHandle,rc);
    return rc;
  }
  TraceFileLog(slogUtilscom,MSGS, "[%04d] %s OK    Dev[%s]       flags[%04X]
h[%04d] OpenOK Conf Port\n", __LINE__, __FUNCTION__,ptSerialCfg-
>device,flags,*devHandle);

  memset(&ptSerialCfg->oldtio, 0x00, sizeof(struct termios));

  memset(&ptSerialCfg->newtio, 0x00, sizeof(struct termios));

  // Reading attributes from Old Configuration
  rc = tcgetattr(*devHandle, &ptSerialCfg->oldtio);
  if (rc != 0)
```

```
  {
    TraceFileLog(slogUtilscom,WRN, "[%04d] %s WR    Dev[%s]       flags[%04X]
h[%04d] tcgetattr=rc[%d] Get Attributes Fail\n", __LINE__,
__FUNCTION__,ptSerialCfg->device,*devHandle,rc);
  }
  memcpy(&ptSerialCfg->newtio, &ptSerialCfg->oldtio, sizeof(ptSerialCfg->newtio));

  ptSerialCfg->newtio.c_cflag = CLOCAL | CREAD;
  char strDevice[128];
  memset(strDevice, 0, sizeof(strDevice));
  strcpy(strDevice, ptSerialCfg->device); //Corrected by KlocWorks
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s       Dev[%s]      c_flag[%04X]
CLOCAL CREAD\n", __LINE__, __FUNCTION__, ptSerialCfg->device,ptSerialCfg-
>newtio.c_cflag);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s  baudrate[%d]\n", __LINE__,
__FUNCTION__, ptSerialCfg->baudrate);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s  databits[%d]\n", __LINE__,
__FUNCTION__, ptSerialCfg->databits);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    parity[%c]\n", __LINE__,
__FUNCTION__, ptSerialCfg->parity);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s  stopbits[%d]\n", __LINE__,
__FUNCTION__, ptSerialCfg->stopbits);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s   flowctl[%c]\n", __LINE__,
__FUNCTION__, ptSerialCfg->flowctl);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s  blocking[%d]\n", __LINE__,
__FUNCTION__, ptSerialCfg->blocking);
  // Reading Old Configuration
  rc = GetCommState(*devHandle, &ptSerialCfg->oldtio);
  if (rc != 0)
  {
    CloseSerialComm((int*)devHandle,ptSerialCfg);
    *devHandle = 0;
    TraceFileLog(slogUtilscom,ERR, "[%04d] %s ER    Dev[%s] GetCommState=rc[%d]
Fail\n", __LINE__, __FUNCTION__, strDevice, rc);
    return rc;
  }
  //
  // Configurating Serial Ports
  //
  memcpy(&ptSerialCfg->newtio, &ptSerialCfg->oldtio, sizeof(ptSerialCfg->newtio));
  rc = SetBaudrate(ptSerialCfg->baudrate,ptSerialCfg);
  if (rc == TRUE)
  {
    //Set BaudRate OK
    rc = SetParity(ptSerialCfg->parity,ptSerialCfg);
    if (rc == TRUE)
    {
      //Set Parity OK
      rc = SetDataBits(ptSerialCfg->databits,ptSerialCfg);
      if (rc == TRUE)
      {
        //Set StopDataBits OK
        rc = SetStopBits(ptSerialCfg->stopbits,ptSerialCfg);
        if (rc == TRUE)
        {
          //Set StopBits OK
          rc = SetFlowControl(ptSerialCfg->flowctl,ptSerialCfg);
          if (rc == TRUE)
          {
            rc = SetCommState(*devHandle, &ptSerialCfg->newtio);
            if (rc == 0)
            {
```

```
              //Ports Was Successfull Configurated
              rc = GetCommTimeouts(*devHandle, &timeouts, ptSerialCfg);
                TraceFileLog(slogUtilscom,MSGS,"[%04d] %s rc[%d]\n", __LINE__,
__FUNCTION__,rc);
              if (rc != COMM_RETURN_OK)
              {
                TraceFileLog(slogUtilscom,MSGS,"[%04d] %s ER Dev[%s]
GetCommTimeouts  h[%04X] rc[%d] To Fail\n\n", __LINE__, __FUNCTION__, strDevice,
*devHandle, rc);
                rc = COMM_RETURN_OK;
              }
              timeouts.WriteTotalTimeoutConstant = 2000; //ADDED 27-01-2012 by
Oliveira
              timeouts.WriteTotalTimeoutMultiplier =1;  //ADDED 27-01-2012 by
Oliveira
              timeouts.ReadIntervalTimeout = 100;
              timeouts.ReadTotalTimeoutConstant = 100;
              timeouts.ReadTotalTimeoutMultiplier = 1;
              rc = SetCommTimeouts(*devHandle, &timeouts, ptSerialCfg);
              if (rc == COMM_RETURN_OK)
              {
                //Set Timeouts OK - PortConfiguration Successfull
                rc = COMM_RETURN_OK;
              }
              else
              {
                //Set Timeouts Error
                TraceFileLog(slogUtilscom,MSGS,"[%04d] %s ER    Dev[%s] h[%04X]
rc[%d] To Fail\n\n", __LINE__, __FUNCTION__, strDevice, *devHandle, rc);
              }
            }
            else
            {
              //Set CommState Fails
              TraceFileLog(slogUtilscom,MSGS,"[%04d] %s ER    Dev[%s] h[%04X]
rc[%d] SetCommSt Fail\n\n", __LINE__, __FUNCTION__, strDevice, *devHandle,rc);
            }
          }
          else
          {
            //Set FlowControl Error
            TraceFileLog(slogUtilscom,MSGS, "[%04d] %s ER    Dev[%s] SetFlowCtl
Fails h[%04X] RC[%d]\n\n", __LINE__, __FUNCTION__, strDevice, *devHandle, rc);
          }
        }
        else
        {
          //Set StopBits Error
          TraceFileLog(slogUtilscom,MSGS, "[%04d] %s ER    Dev[%s] SetStopBits
Fails h[%04X] RC[%d]\n\n", __LINE__, __FUNCTION__, strDevice, *devHandle, rc);
        }
      }
      else
      {
        //Set DataBits Error
        TraceFileLog(slogUtilscom,MSGS, "[%04d] %s ER    Dev[%s] SetDataBits Fails
h[%04X] RC[%d]\n\n", __LINE__, __FUNCTION__, strDevice, *devHandle, rc);
      }
    }
    else
    {
      //Set Parity Error
```

```
      TraceFileLog(slogUtilscom,MSGS, "[%04d] %s ER    Dev[%s] SetParity Fails
h[%04X] RC[%d]\n\n", __LINE__, __FUNCTION__, strDevice, *devHandle, rc);
    }
  }
  else
  {
    //Set BaudRate Error
    TraceFileLog(slogUtilscom,MSGS, "[%04d] %s ER    Dev[%s] SetBaudRate Fails
h[%04X] RC[%d]\n\n", __LINE__, __FUNCTION__, strDevice, *devHandle, rc);
  }
  if (rc != COMM_RETURN_OK)
  {
    //Any Set Error detected!
    CloseSerialComm((int*)devHandle,ptSerialCfg);
    return rc;
  }
  ptSerialCfg->newtio.c_lflag = 0;
  ptSerialCfg->newtio.c_oflag = 0;
  ptSerialCfg->newtio.c_cc[VTIME] = 0;
  ptSerialCfg->newtio.c_cc[VMIN] = 1;
  ptSerialCfg->newtio.c_iflag = IGNPAR;

  tcflush(*devHandle, TCIFLUSH);
  rc = tcsetattr(*devHandle, TCSANOW, &ptSerialCfg->newtio);
  if (rc != COMM_RETURN_OK)
  {
    TraceFileLog(slogUtilscom,ERR, "[%04d] %s ER    Dev[%s] h[%04X] rc[%d] Set
Attribute Fail!\n\n", __LINE__, __FUNCTION__, strDevice, *devHandle, rc);
    CloseSerialComm((int*)devHandle,ptSerialCfg);
    *devHandle = -1;
    return rc;
  }
  //TraceFileLog(slogUtilscom,WRN, "[%04d] %s OK    Dev[%s] h[%04X] rc[%d]
Successfull Openned!\n\n", __LINE__, __FUNCTION__, strDevice, *devHandle, rc);

  return rc;
#else
  //Windows Open Comm
  if (ptSerialCfg->device == NULL)
  {
    TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER No_Device to Open!!\n",
__LINE__, __FUNCTION__);
    return COMM_CFG_ERROR;
  }
  char strDevice[50];
  memset(strDevice, 0, sizeof(strDevice));
  strcpy(strDevice, "\\\\.\\"); //**ATTENTION** This is required for
WindowsCommPorts GreaTher then COM9
  strcat(strDevice, ptSerialCfg->device);
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s  Starting  Dev[%s]
*devHandle[%04X] ptSerialCfg[%04X]\n", __LINE__, __FUNCTION__, strDevice,
*devHandle, ptSerialCfg);
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s    device[%s]\n", __LINE__,
__FUNCTION__, ptSerialCfg->device);
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s  baudrate[%d]\n", __LINE__,
__FUNCTION__, ptSerialCfg->baudrate);
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s  databits[%d]\n", __LINE__,
__FUNCTION__, ptSerialCfg->databits);
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s    parity[%c]\n", __LINE__,
__FUNCTION__, ptSerialCfg->parity);
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s  stopbits[%d]\n", __LINE__,
__FUNCTION__, ptSerialCfg->stopbits);
```

```
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s   flowctl[%c]\n", __LINE__,
__FUNCTION__, ptSerialCfg->flowctl);
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s  blocking[%d]\n", __LINE__,
__FUNCTION__, ptSerialCfg->blocking);

  if (ptSerialCfg->blocking == TRUE)
  {
    *devHandle = CreateFile(strDevice, GENERIC_READ | GENERIC_WRITE, 0, 0,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
  }
  else
  {
    *devHandle = CreateFile(strDevice, GENERIC_READ | GENERIC_WRITE, 0, 0,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
  }
  if ((*devHandle == NULL) || (*devHandle == (HANDLE)(-1)))
  {
    rc = COMM_IO_ERROR;
    TraceFileLog(slogUtilscom, ERR, "[%04d] %s *devHandle[%04x]
Device[%s:%ld,%d,%c,%d] flw[%c] blk[%d] rc[%d] Fail\n", __LINE__, __FUNCTION__,
                                *devHandle,
                                strDevice,
                                ptSerialCfg->baudrate,
                                ptSerialCfg->databits,
                                ptSerialCfg->parity,
                                ptSerialCfg->stopbits,
                                ptSerialCfg->flowctl,
                                ptSerialCfg->blocking,
                                rc
                                );
    return rc;
  }
  memset(&ptSerialCfg->oldtio, 0x00, sizeof(DCB));
  memset(&ptSerialCfg->newtio, 0x00, sizeof(DCB));
  // Saving old config
  if (!GetCommState(*devHandle, &ptSerialCfg->oldtio))
  {
    rc = COMM_INVALID_PARAMETER;
    CloseSerialComm(*devHandle,ptSerialCfg);
    devHandle = NULL;
    TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER Dev[%s] Config error ClosingComm
RC[%d]\n", __LINE__, __FUNCTION__, strDevice, rc);
    return rc;
  }
  //
  // Configurating Serial Ports
  //
  memcpy(&ptSerialCfg->newtio, &ptSerialCfg->oldtio, sizeof(ptSerialCfg->newtio));
  rc = SetBaudrate(ptSerialCfg->baudrate,ptSerialCfg);
  if (rc >= COMM_RETURN_OK)
  {
    //Set BaudRate OK
    rc = SetParity(ptSerialCfg->parity,ptSerialCfg);
    if (rc >= COMM_RETURN_OK)
    {
      //Set Parity OK
      rc = SetDataBits(ptSerialCfg->databits,ptSerialCfg);
      if (rc >= COMM_RETURN_OK)
      {
        //Set StopDataBits OK
        rc = SetStopBits(ptSerialCfg->stopbits,ptSerialCfg);
        if (rc >= COMM_RETURN_OK)
```

```
        {
          //Set StopBits OK
          rc = SetFlowControl(ptSerialCfg->flowctl,ptSerialCfg);
          if (rc >= COMM_RETURN_OK)
          {
            SetCommState(*devHandle, &ptSerialCfg->newtio);
            timeouts.ReadIntervalTimeout = 100;
            timeouts.ReadTotalTimeoutConstant = 100;
            timeouts.ReadTotalTimeoutMultiplier = 1;
            rc = SetCommTimeouts(*devHandle, &timeouts);
            if (rc >= COMM_RETURN_OK)
            {
              //Set Timeouts OK - PortConfiguration Successfull
              rc = COMM_RETURN_OK;
              //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s OK Port Configuration
Successfull Dev[%s] *devHandle[%04X] RC[%d]\n", __LINE__, __FUNCTION__, strDevice,
*devHandle, rc);
              return rc;
            }
            else
            {
              //Set Timeouts Error
              TraceFileLog(slogUtilscom, MSGS, "[%04d] %s ER SetTimeout Fails
Dev[%s] *devHandle[%04X] RC[%d]\n", __LINE__, __FUNCTION__, strDevice, *devHandle,
rc);
            }
          }
          else
          {
            //Set FlowControl Error
            TraceFileLog(slogUtilscom, MSGS, "[%04d] %s ER SetFlowControl Fails
Dev[%s] *devHandle[%04X] RC[%d]\n", __LINE__, __FUNCTION__, strDevice, *devHandle,
rc);
          }
        }
        else
        {
          //Set StopBits Error
          TraceFileLog(slogUtilscom, MSGS, "[%04d] %s ER SetStopBits Fails Dev[%s]
*devHandle[%04X] RC[%d]\n", __LINE__, __FUNCTION__, strDevice, *devHandle, rc);
        }
      }
      else
      {
        //Set DataBits Error
        TraceFileLog(slogUtilscom, MSGS, "[%04d] %s ER SetDataBits Fails Dev[%s]
*devHandle[%04X] RC[%d]\n", __LINE__, __FUNCTION__, strDevice, *devHandle, rc);
      }
    }
    else
    {
      //Set Parity Error
      TraceFileLog(slogUtilscom, MSGS, "[%04d] %s ER SetParity Fails Dev[%s]
*devHandle[%04X] RC[%d]\n", __LINE__, __FUNCTION__, strDevice, *devHandle, rc);
    }
  }
  else
  {
    //Set BaudRate Error
    TraceFileLog(slogUtilscom, MSGS, "[%04d] %s ER SetBaudRate Fails Dev[%s]
*devHandle[%04X] RC[%d]\n", __LINE__, __FUNCTION__, strDevice, *devHandle, rc);
  }
```

```
  //Any Set Error detected!
  rc = COMM_INVALID_PARAMETER;
  TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER Dev[%s] SetCommTimeouts Fail
RC[%d]\n", __LINE__, __FUNCTION__, strDevice, rc);
  if (*devHandle)
  {
    CloseSerialComm(*devHandle,ptSerialCfg); //Void
  }
  *devHandle = NULL;
  TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER InvalidConfig Dev[%s] RC[%d]\n",
__LINE__, __FUNCTION__, strDevice, rc);

  return rc;
#endif
}
/*



 * Close serial communication



 * Param devHandle :



 *         communication handler



 * Return values :



 *         void

 */
void CloseSerialComm(HANDLE * devHandle, SERIAL_CFG *ptSerialCfg)
{
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s devHandle[%04X]
ptSerialCfg[%04X]\n", __LINE__, __FUNCTION__,devHandle,ptSerialCfg);
  #ifdef __WINDOWS__
  if ((!devHandle) || (*devHandle == (HANDLE)-1))
  #else
  if ((!devHandle) || (*devHandle < 0 ))
  #endif
  {
    //TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER devHandle[NULL]\n", __LINE__,
__FUNCTION__);
    return;
  }
#ifdef __LINUX__

  int rc = 0;

  HANDLE oldHandle;

  oldHandle = *devHandle;
//  rc = FlushSerialComm(*devHandle,0,ptSerialCfg);
//  if (rc != 0)
//  {
```

```
//    TraceFileLog(slogUtilscom,WRN, "[%04d] %s WR devHandle[%04X] ptSerialCfg-
>oldtio[%04X]   tcflush rc[%d]\n", __LINE__, __FUNCTION__, *devHandle,
&ptSerialCfg->oldtio,rc);
//  }
//  rc = tcsetattr(*devHandle, TCSANOW, &ptSerialCfg->oldtio);
//  if (rc != 0)
//  {
//    TraceFileLog(slogUtilscom,WRN, "[%04d] %s WR devHandle[%04X] ptSerialCfg-
>oldtio[%04X] tcsetattr rc[%d]\n", __LINE__, __FUNCTION__, *devHandle,
&ptSerialCfg->oldtio,rc);
//  }

  //Função close do sistema operacional dando problema com a IDC (demora 30 seg
para executar a função, se a IDC não estiver conectada)

  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s Chegou aqui\n", __LINE__,
__FUNCTION__);
  rc = close(*devHandle);
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s Chegou aqui\n", __LINE__,
__FUNCTION__);
  if (rc != 0)
  {
    TraceFileLog(slogUtilscom,ERR, "[%04d] %s ER H[%04X] ptSerialCfg->oldtio[%04X]
close rc[%d]\n", __LINE__, __FUNCTION__, *devHandle, &ptSerialCfg->oldtio,rc);
  }
  else
  {
    *devHandle = -1; //C.A.U.T.I.O.N: On Linux the Handle 0 is valid
    //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s OK devHandle[%04X] ptSerialCfg-
>oldtio[%04X]     close rc[%d]\n", __LINE__, __FUNCTION__, *devHandle,
&ptSerialCfg->oldtio,rc);
  }
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s     Old[%04X] H[%04X] ptSerialCfg-
>oldtio[%08X] rc[%d]\n\n", __LINE__, __FUNCTION__,oldHandle,*devHandle,
&ptSerialCfg->oldtio, rc);
#else
  CloseHandle(*devHandle);
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s devHandle[%04X] closed3\n",
__LINE__, __FUNCTION__,*devHandle);
  *devHandle = NULL;
#endif
}
//Concatenate [Header + CMD + Param + Folder + Checksum]  then Send >
DeviceHandler
int SendCMD(HANDLE devHandle, int iProtocol, char *pCMD, int iLenCMD, char*
pParam, int iLenParam, unsigned long long ulTxTimeout,SERIAL_CFG *ptSerialCfg)
{
  int rc = COMM_INVALID_COMMAND;
  int iLen = 0;
  char sLocalCMD[100];
  memset(sLocalCMD, 0, sizeof(sLocalCMD));
  memcpy(sLocalCMD, pCMD, iLenCMD);
  memcpy((sLocalCMD + iLenCMD), pParam, iLenParam);
  iLen = iLenCMD + iLenParam;
  rc = InsertCMDPayLoad(iProtocol, pCMD, &iLen,ptSerialCfg);
  if (rc == COMM_RETURN_OK)
  {
    rc = WriteData(devHandle, iLen, pCMD, 200,ptSerialCfg);
    if (rc == iLen)
    {
      return COMM_RETURN_OK;
    }
```

```
  }
  TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER RC=[%d]\n", __LINE__,
__FUNCTION__, rc);
  return rc;
}
//Receive [Header + RSP)Rsult + Folder + Checksum], compute Checksum,
//        Remove Headers & Folder and return only pRspBuff data and it Len
int RecvRSP(HANDLE devHandle, int iProtocol, char* pRspBuff, int *iLen, unsigned
long long ulRxTimeout, SERIAL_CFG *ptSerialCfg)
{
  int rc;
  char sRspBuffLocal[2048];
  memset(sRspBuffLocal, 0, sizeof(sRspBuffLocal));
  rc = ReadData(devHandle, *iLen, sRspBuffLocal,200,ptSerialCfg);
  if (rc != COMM_RETURN_OK)
  {
    return COMM_RSP_FORMAT_ERROR;
  }
  rc = RemoveRSPPayLoad(iProtocol, sRspBuffLocal, pRspBuff, iLen,ptSerialCfg);
  return rc;
}

int ReadGetKeyData(HANDLE devHandle, int iBytesToRead, char *data, unsigned long
ulTimeout,SERIAL_CFG *ptSerialCfg)
{
  int rc = 0;
  DWORD dwRead = 0;
  DWORD dwTimeout = ulTimeout;
  DWORD dwStartTime = 0, dwDifTime = 0;
  dwStartTime = GetTickCount(); //Start Time
  dwDifTime = 0;
  do
  {
#ifdef __LINUX__
    rc = ReadFile(devHandle, (void*) data, (DWORD) iBytesToRead, &dwRead,
&ulTimeout);
    if (rc > 0)
    {
      //Error or Data received
      break;
    }
#else
    rc = ReadFile(devHandle, (void*) data, (DWORD) iBytesToRead, &dwRead, NULL);
    if (dwRead > 0)
    {
      //Error or Data received
      break;
    }
#endif
    //TraceFileLog(slogUtilscom, ERR, "[%04d] %s RC=[%d] data[%02X] read[%d]\n\n",
__LINE__, __FUNCTION__, rc, data[0], dwRead);
    dwDifTime = GetTickCount() - dwStartTime;
    //my_usleep(dwTimeout/10);
    //usleep(dwTimeout/10);
  }while (dwTimeout > dwDifTime);

  if (rc < 0)
  {
    //TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER h[%04X] iBytesToRead[%d]
read[%d] RC[%d] t[%d]ms timeout set[%d]ms\n", __LINE__, __FUNCTION__, devHandle,
iBytesToRead, read, rc, dwDifTime,dwTimeout);
    return rc;
```

```
  }
  return (int) dwRead;
}

/*Adicionado por João Marcelo - wrapper para funcao basicas de leitura*/
int ReadData(HANDLE devHandle, int iBytesToRead, char *data, unsigned long
ulTimeout,SERIAL_CFG *ptSerialCfg)
{
  int rc = 0;
  DWORD dwRead = 0;
  DWORD dwTimeout = 200;
  DWORD dwStartTime = 0, dwDifTime = 0;
  dwStartTime = GetTickCount(); //Start Time
  dwDifTime = 0;
  while (dwTimeout > dwDifTime)
  {
#ifdef __LINUX__
    rc = ReadFile(devHandle, (void*) data, (DWORD) iBytesToRead, &dwRead,
&dwTimeout);
    if (rc > 0)
    {
      //Error or Data received
      break;
    }
#else
    rc = ReadFile(devHandle, (void*) data, (DWORD) iBytesToRead, &dwRead, NULL);
    if (dwRead > 0)
    {
      //Error or Data received
      break;
    }
#endif
    //TraceFileLog(slogUtilscom, ERR, "[%04d] %s RC=[%d] data[%02X] read[%d]\n\n",
__LINE__, __FUNCTION__, rc, data[0], dwRead);
    dwDifTime = GetTickCount() - dwStartTime;
    //my_usleep(dwTimeout/10);
//    usleep(dwTimeout/10);
  }
  if (rc < 0)
  {
    //TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER h[%04X] iBytesToRead[%d]
read[%d] RC[%d] t[%d]ms timeout set[%d]ms\n", __LINE__, __FUNCTION__, devHandle,
iBytesToRead, read, rc, dwDifTime,dwTimeout);
    return rc;
  }
  return (int) dwRead;
}
/*Adicionado por João Marcelo - wrapper para funcao basicas de escrita*/
// Timeout negativo para sem limite de tempo
int WriteData(HANDLE devHandle, int szTXBuffer, char *data, unsigned long
ulTimeout, SERIAL_CFG *ptSerialCfg)
{
  int rc;
  int written = 0;
  int offset = 0;
  SERIAL_CFG tCfgLocal;
  COMMTIMEOUTS ct;
  memset(&ct,0,sizeof(COMMTIMEOUTS));
  memset(&tCfgLocal,0,sizeof(SERIAL_CFG));
  DWORD dwCurrentTime = 0;
  DWORD dwTimeoutTime = 0;
  dwCurrentTime = GetTickCount(); //Current Time
```

```
  dwTimeoutTime = dwCurrentTime + ulTimeout; //Last Time = Current Time + Offset
#ifdef __LINUX__
  if (devHandle < 0 || data == NULL || szTXBuffer <= 0) //Corrected by Klocwork
#else
  if (devHandle == NULL || devHandle == (HANDLE)0xFFFFFFFF || data == NULL ||
szTXBuffer <= 0) //Corrected by Klocwork
#endif
  {
    if (szTXBuffer == 0)
    {
      return 0;
    }
    rc = COMM_INVALID_PARAMETER;
    if(data)
    {
      if ((szTXBuffer == 1) && (data[0] < ' '))
      {
        #ifdef __LINUX__
        if (devHandle < 0) //Corrected by Klocwork
        #else
        if ((devHandle == (HANDLE)0xFFFFFFFF) || devHandle == NULL )
        #endif
        {
          //No Log
        }
        else
        {
          TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER h[%04x] data[0]=[%02X]
L[%d], T[%d] RC[%d] ER Null Handle or ptr\n", __LINE__, __FUNCTION__,devHandle,
data[0], szTXBuffer,ulTimeout, rc);
        }
      }
      else
      {
        #ifdef __LINUX__
        if (devHandle < 0) //Corrected by Klocwork
        #else
        if ((devHandle == (HANDLE)0xFFFFFFFF) || devHandle == NULL )
        #endif
        {
          //No Log
        }
        else
        {
          TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER h[%04x] Data[%s] L[%d],
T[%d] RC[%d] ER Null ptr\n", __LINE__, __FUNCTION__,devHandle, data,
szTXBuffer,ulTimeout, rc);
        }
      }
    }
    else
    {
      rc = COMM_NO_HANDLER;
      TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER h[%04x] d[%04x] L[%d], T[%d]
RC[%d] ER Null Handle or ptr\n", __LINE__, __FUNCTION__,devHandle, data,
szTXBuffer,ulTimeout, rc);
    }
    return rc;
  }
  if (ulTimeout != 0)
  {
    #ifdef __LINUX__
```

```
    rc = GetCommTimeouts(devHandle, &ct,&tCfgLocal);
    #else
    rc = GetCommTimeouts(devHandle, &ct);
    #endif
    if (rc < 0)
    {
      TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER h[%04X] GetCommTimeouts
RC=[%d]\n", __LINE__, __FUNCTION__,devHandle,rc);
      return rc;
    }
    ct.ReadIntervalTimeout = 100;
    ct.ReadTotalTimeoutConstant = (DWORD) ulTimeout * 2;
    ct.ReadTotalTimeoutMultiplier = 1;
    ct.WriteTotalTimeoutMultiplier = 1;
    ct.WriteTotalTimeoutConstant = (DWORD) ulTimeout;
    #ifdef __LINUX__
    rc = SetCommTimeouts(devHandle, &ct,&tCfgLocal);
    #else
    rc = SetCommTimeouts(devHandle, &ct);
    #endif
    if (rc < 0)
    {
      TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER SetCommTimeouts RC=[%d]\n",
__LINE__, __FUNCTION__, rc);
      return rc;
    }
    //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s TxTimeout=%d\n", __LINE__,
__FUNCTION__,ct.WriteTotalTimeoutConstant);
  }
  //    TraceFileLog(slogUtilscom,MSGS, "[%d] %s h[%04d] L[%d], T[%d]\n",
__LINE__, __FUNCTION__,devHandle,szTXBuffer,timeout);
  while (szTXBuffer != 0)
  {
    //        TraceFileLog(slogUtilscom,ERR, "[%d] %s qtd[%d] cTime[%d] lTime[%d]
Diff[%d]\n", __LINE__, __FUNCTION__,szTXBuffer,cTime,lTime,(lTime-cTime));
    rc = WriteFile(devHandle, (void*) (data + offset), (DWORD) szTXBuffer,
(DWORD*) &written, NULL);
    if (written > 0)
    {
      offset += written;
      szTXBuffer -= written;
      written = 0;
    }
    dwCurrentTime = GetTickCount();
    if (dwCurrentTime > dwTimeoutTime) //Current Time > Last Time
    {
      rc = COMM_TXTIMEOUT;
      //TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER Timeout RC=[%d]\n",
__LINE__, __FUNCTION__, rc);
      return rc;
    }
  }
  dwCurrentTime = GetTickCount();
  //    TraceFileLog(slogUtilscom,MSGS, "[%d] %s OK Sent[%d]Bytes T[%d]\n",
__LINE__, __FUNCTION__,offset,(lTime-cTime));
  return offset;
}

int InsertCMDPayLoad(int iProtocol, char *pCMD_Parm, int *iLen,SERIAL_CFG
*ptSerialCfg)
{
  //1- Insert  Header on front     of <CMD_Message> = Header+<CMD_Message>
```

```
  //2- Insert  Folder on back      of <CMD_Message> = Header+<CMD_Message>+Folder
  //3- Compute/ADD CRC/CKS on Back of <CMD_Message> =
Header+<CMD_Message>+Folder+CRC/CKS
  int rc = COMM_RETURN_OK;
  TraceFileLog(slogUtilscom, ERR, "[%04d] %s NotImplemented RC=[%d]\n", __LINE__,
__FUNCTION__, rc);
  return rc;
}
int RemoveRSPPayLoad(int iProtocol, char *pRSPin, char *pRSPout, int
*iLen,SERIAL_CFG *ptSerialCfg)
{
  //1- Verify CRC/CKS
  //2- Remove Header from Front    of  Header+<Response_Message>+Folder+CRC/CKS
  //3- Remove Folder from back     of  Header+<Response_Message>+Folder+CRC/CKS
  int rc = COMM_RETURN_OK;
  TraceFileLog(slogUtilscom, ERR, "[%04d] %s NotImplemented RC=[%d]\n", __LINE__,
__FUNCTION__, rc);

  return rc;
}

//Configure BaudRate
BOOL SetBaudrate(unsigned long int baudrate,SERIAL_CFG *ptSerialCfg)
{
#ifdef __LINUX__
  //Linux Setting Baud Rate
  speed_t brate = 0;
  int rc;
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s     B50[%08X]\n", __LINE__,
__FUNCTION__,B50);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s     B75[%08X]\n", __LINE__,
__FUNCTION__,B75);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    B110[%08X]\n", __LINE__,
__FUNCTION__,B110);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    B134[%08X]\n", __LINE__,
__FUNCTION__,B134);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    B150[%08X]\n", __LINE__,
__FUNCTION__,B150);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    B200[%08X]\n", __LINE__,
__FUNCTION__,B200);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    B300[%08X]\n", __LINE__,
__FUNCTION__,B300);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    B600[%08X]\n", __LINE__,
__FUNCTION__,B600);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s   B1200[%08X]\n", __LINE__,
__FUNCTION__,B1200);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s   B2400[%08X]\n", __LINE__,
__FUNCTION__,B2400);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s   B4800[%08X]\n", __LINE__,
__FUNCTION__,B4800);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s   B9600[%08X]\n", __LINE__,
__FUNCTION__,B9600);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s  B19200[%08X]\n", __LINE__,
__FUNCTION__,B19200);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s  B38400[%08X]\n", __LINE__,
__FUNCTION__,B38400);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s  B57600[%08X]\n", __LINE__,
__FUNCTION__,B57600);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s B115200[%08X]\n", __LINE__,
__FUNCTION__,B115200);
  switch (baudrate)
  {
```

```
    case 50:
    {
      brate = B50;
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B50[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      break;
    }
    case 75:
    {
      brate = B75;
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B75[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      break;
    }
    case 110:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B110[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      brate = B110;
      break;
    }
    case 134:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B134[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      brate = B134;
      break;
    }
    case 150:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B150[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      brate = B150;
      break;
    }
    case 200:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B200[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      brate = B200;
      break;
    }
    case 300:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B300[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      brate = B300;
      break;
    }
    case 600:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B600[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      brate = B600;
```

```
      break;
    }
    case 1200:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B1200[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      brate = B1200;
      break;
    }
    case 1800:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B1800[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      brate = B1800;
      break;
    }
    case 2400:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B2400[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      brate = B2400;
      break;
    }
    case 4800:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B4800[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      brate = B4800;
      break;
    }
    case 9600:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B9600[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      brate = B9600;
      break;
    }
    case 19200:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B19200[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      brate = B19200;
      break;
    }
    case 38400:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B38400[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      brate = B38400;
      break;
    }
    case 57600:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B57600[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
```

```
>newtio.c_cflag,brate);
      brate = B57600;
      break;
    }
    case 115200:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    Baudrate[%lu]
c_cflag[%08X] B115200[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      brate = B115200;
      break;
    }
    default:
    {
      //TraceFileLog(slogUtilscom,ERR, "[%04d] %s ER Baudrate[%lu]
c_cflag[%08X] UNKOWN[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
      return (FALSE);
    }
  }
  rc = cfsetispeed(&ptSerialCfg->newtio, brate);
  if (rc != 0)
  {
    TraceFileLog(slogUtilscom,ERR, "[%04d] %s    ER Baudrate[%lu]=[%04X]
cfsetispeed rc[%d]!\n", __LINE__, __FUNCTION__, baudrate, brate,rc);
    return(FALSE);
  }
  rc = cfsetospeed(&ptSerialCfg->newtio, brate);
  if (rc != 0)
  {
    TraceFileLog(slogUtilscom,ERR, "[%04d] %s    ER Baudrate[%lu]=[%04X]
cfsetospeed rc[%d]!\n", __LINE__, __FUNCTION__, baudrate, brate,rc);
    return(FALSE);
  }
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    OK Baudrate[%lu]
c_cflag[%08X] brate[%08X]\n", __LINE__, __FUNCTION__, baudrate,ptSerialCfg-
>newtio.c_cflag,brate);
#else
  //Windows Setting Baud Rate
  int iBaudRate = 0;
  switch(baudrate)
  {
    case 50:
    {
      iBaudRate = CBR_110;
      break;
    }
    case 75:
    {
      iBaudRate = CBR_110;
      break;
    }
    case 110:
    {
      iBaudRate = CBR_110;
      break;
    }
    case 134:
    {
      iBaudRate = CBR_110;
      break;
    }
```

```
    case 150:
    {
      iBaudRate = CBR_110;
      break;
    }
    case 200:
    {
      iBaudRate = CBR_110;
      break;
    }
    case 300:
    {
      iBaudRate = CBR_300;
      break;
    }
    case 600:
    {
      iBaudRate = CBR_600;
      break;
    }
    case 1200:
    {
      iBaudRate = CBR_1200;
      break;
    }
    case 1800:
    {
      iBaudRate = CBR_1200;
      break;
    }
    case 2400:
    {
      iBaudRate = CBR_2400;
      break;
    }
    case 4800:
    {
      iBaudRate = CBR_4800;
      break;
    }
    case 9600:
    {
      iBaudRate = CBR_9600;
      break;
    }
    case 19200:
    {
      iBaudRate = CBR_19200;
      break;
    }
    case 38400:
    {
      iBaudRate = CBR_38400;
      break;
    }
    case 57600:
    {
      iBaudRate = CBR_57600;
      break;
    }
    case 115200:
    {
```

```
      iBaudRate = CBR_115200;
      break;
    }
    default:
    {
      TraceFileLog(slogUtilscom, MSGS, "[%04d] %s ER Baudrate[%lu]=[%04X] Invalid
Parameter!\n", __LINE__, __FUNCTION__, baudrate, ptSerialCfg->newtio.BaudRate);
      return (FALSE);
    }
  }
  ptSerialCfg->newtio.BaudRate = iBaudRate;
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s Baudrate[%lu] [%04X]\n", __LINE__,
__FUNCTION__, baudrate, ptSerialCfg->newtio.BaudRate);
#endif
  return (TRUE);
}
//Set Parity at DCB ptSerialCfg->newtio
BOOL SetParity(char parity,SERIAL_CFG * ptSerialCfg)
{
#ifdef __LINUX__
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s  parity[%c]\n", __LINE__,
__FUNCTION__,parity);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s  PARENB[%08X]\n", __LINE__,
__FUNCTION__,PARENB);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s  PARODD[%08X]\n", __LINE__,
__FUNCTION__,PARODD);
  switch (parity)
  {
    case 'E':
    case 'e':
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s      OK    Parity[%c] Even
c_cflag[%04X]\n", __LINE__, __FUNCTION__, parity, ptSerialCfg->newtio.c_cflag);
      ptSerialCfg->newtio.c_cflag |= PARENB;  //ON
      ptSerialCfg->newtio.c_cflag &= ~PARODD; //OFF
      break;

    case 'O':
    case 'o':
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s      OK    Parity[%c] Odd
c_cflag[%04X]\n", __LINE__, __FUNCTION__, parity, ptSerialCfg->newtio.c_cflag);
      ptSerialCfg->newtio.c_cflag |= PARENB; //ON
      ptSerialCfg->newtio.c_cflag |= PARODD; //ON
      break;

    case 'N': //No Parity
    case 'n':
      ptSerialCfg->newtio.c_cflag &= ~PARENB; //OFF
      ptSerialCfg->newtio.c_cflag &= ~PARODD; //OFF
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s      OK    Parity[%c] No
c_cflag[%04X]\n", __LINE__, __FUNCTION__, parity, ptSerialCfg->newtio.c_cflag);
      break;

    case 'M': //Mark Parity
    case 'm':
      ptSerialCfg->newtio.c_cflag |=  PARENB; //ON
      ptSerialCfg->newtio.c_cflag &= ~PARODD; //OFF
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s      OK    Parity[%c] Mark
c_cflag[%04X]\n", __LINE__, __FUNCTION__, parity, ptSerialCfg->newtio.c_cflag);
      break;

    case 'S': //Space Parity
    case 's':
```

```
      ptSerialCfg->newtio.c_cflag |= PARENB;
      ptSerialCfg->newtio.c_cflag &= ~PARODD;
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s      OK   Parity[%c] Space
c_cflag[%04X]\n", __LINE__, __FUNCTION__, parity, ptSerialCfg->newtio.c_cflag);
      break;

    default:
      TraceFileLog(slogUtilscom,ERR, "[%04d] %s      ER   Parity[%c]
c_cflag[%04X] Invalid Parameter!\n", __LINE__, __FUNCTION__, parity, ptSerialCfg-
>newtio.c_cflag);
      return (FALSE);
  }
#else
  switch(parity)
  {
    case 'E':
    case 'e':
    {
      ptSerialCfg->newtio.Parity = EVENPARITY;
      break;
    }

    case 'O':
    case 'o':
    {
      ptSerialCfg->newtio.Parity = ODDPARITY;
      break;
    }

    case 'N':
    case 'n':
    {
      ptSerialCfg->newtio.Parity = NOPARITY;
      break;
    }

    case 'M':
    case 'm':
    {
      ptSerialCfg->newtio.Parity = MARKPARITY;
      break;
    }

    case 'S':
    case 's':
    {
      ptSerialCfg->newtio.Parity = SPACEPARITY;
      break;
    }

    default:
    {
      TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER Parity[%c]=[%04X] Invalid
Parameter!\n", __LINE__, __FUNCTION__, parity, ptSerialCfg->newtio.BaudRate);
      return (FALSE);
    }
  }
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s Parity[%c]=[%02X]\n", __LINE__,
__FUNCTION__, parity, ptSerialCfg->newtio.Parity);
#endif
  return (TRUE);
}
```

```
BOOL SetDataBits(int databits,SERIAL_CFG * ptSerialCfg)
{
#ifdef __LINUX__
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s   CS5[%04X]\n", __LINE__,
__FUNCTION__,CS5);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s   CS6[%04X]\n", __LINE__,
__FUNCTION__,CS6);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s   CS7[%04X]\n", __LINE__,
__FUNCTION__,CS7);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s   CS8[%04X]\n", __LINE__,
__FUNCTION__,CS8);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    BG0 DataBits[%d] ptSerialCfg-
>newtio.c_cflag[%04X]\n", __LINE__, __FUNCTION__, databits, ptSerialCfg-
>newtio.c_cflag);
  ptSerialCfg->newtio.c_cflag = ptSerialCfg->newtio.c_cflag & 0xFFFFFFCF;
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    BG1 DataBits[%d] ptSerialCfg-
>newtio.c_cflag[%04X]\n", __LINE__, __FUNCTION__, databits, ptSerialCfg-
>newtio.c_cflag);
  switch (databits)
  {
    case 5:
    {
      ptSerialCfg->newtio.c_cflag |= CS5;
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    OK DataBits[%d] ptSerialCfg-
>newtio.c_cflag[%04X] CS5[%04X]\n", __LINE__, __FUNCTION__, databits, ptSerialCfg-
>newtio.c_cflag,CS5);
      break;
    }

    case 6:
    {
      ptSerialCfg->newtio.c_cflag |= CS6;
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    OK DataBits[%d] ptSerialCfg-
>newtio.c_cflag[%04X] CS6[%04X]\n", __LINE__, __FUNCTION__, databits, ptSerialCfg-
>newtio.c_cflag,CS6);
      break;
    }

    case 7:
    {
      ptSerialCfg->newtio.c_cflag |= CS7;
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    OK DataBits[%d] ptSerialCfg-
>newtio.c_cflag[%04X] CS7[%04X]\n", __LINE__, __FUNCTION__, databits, ptSerialCfg-
>newtio.c_cflag,CS7);
      break;
    }

    case 8:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    OK DataBits[%d] ptSerialCfg-
>newtio.c_cflag[%04X] CS8[%04X]\n", __LINE__, __FUNCTION__, databits, ptSerialCfg-
>newtio.c_cflag,CS8);
      ptSerialCfg->newtio.c_cflag |= CS8;
      break;
    }

    default:
    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    ER DataBits[%d] ptSerialCfg-
>newtio.c_cflag[%04X] Invalid Param\n", __LINE__, __FUNCTION__, databits,
ptSerialCfg->newtio.c_cflag);
```

```
      return (FALSE);
    }
  }
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    OK DataBits[%d] ptSerialCfg-
>newtio.c_cflag[%04X]\n", __LINE__, __FUNCTION__, databits, ptSerialCfg-
>newtio.c_cflag);
#else
  switch(databits)
  {
    case 7:
    {
      ptSerialCfg->newtio.ByteSize = databits;
      break;
    }

    case 8:
    {
      ptSerialCfg->newtio.ByteSize = databits;
      break;
    }

    default:
    {
      TraceFileLog(slogUtilscom, MSGS, "[%04d] %s ER DataBits[%d]=[%02X] Invalid
Parameter!\n", __LINE__, __FUNCTION__, databits, ptSerialCfg->newtio.ByteSize);
      return (FALSE);
    }
  }
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s DataBits[%d]=[%02X]\n", __LINE__,
__FUNCTION__, databits, ptSerialCfg->newtio.ByteSize);
#endif
  return (TRUE);
}

BOOL SetStopBits(int stopbits,SERIAL_CFG * ptSerialCfg)
{
#ifdef __LINUX__
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s  CSTOPB[%08X]\n", __LINE__,
__FUNCTION__,CSTOPB);
  switch (stopbits)
  {
    case 0:
    case 1:
    {
      break;
    }

    case 2:
    {
      ptSerialCfg->newtio.c_cflag |= CSTOPB;
      break;
    }

    default:
    {
      TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    ER StopBits[%d] ptSerialCfg-
>newtio.c_cflag[%04X] Invalid Param!\n", __LINE__, __FUNCTION__, stopbits,
ptSerialCfg->newtio.c_cflag);
      return (FALSE);
    }
  }
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    OK StopBits[%d] ptSerialCfg-
```

```
>newtio.c_cflag[%04X]\n", __LINE__, __FUNCTION__, stopbits, ptSerialCfg-
>newtio.c_cflag);
#else
  switch(stopbits)
  {
    case 1:
    {
      ptSerialCfg->newtio.StopBits = ONESTOPBIT;
      break;
    }

    case 2:
    {
      ptSerialCfg->newtio.StopBits = TWOSTOPBITS;
      break;
    }

    default:
    {
      TraceFileLog(slogUtilscom, MSGS, "[%04d] %s ER StopBits[%d]=[%02X]\n",
__LINE__, __FUNCTION__, stopbits, ptSerialCfg->newtio.StopBits);
      return (FALSE);
    }
  }
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s StopBits[%d]=[%02X]\n", __LINE__,
__FUNCTION__, stopbits, ptSerialCfg->newtio.StopBits);
#endif
  return (TRUE);
}

BOOL SetFlowControl(char flowctl,SERIAL_CFG * ptSerialCfg)
{
#ifdef __LINUX__
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s  CRTSCTS[%08X]\n", __LINE__,
__FUNCTION__,CRTSCTS);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s     IXON[%08X]\n", __LINE__,
__FUNCTION__,IXON);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    IXOFF[%08X]\n", __LINE__,
__FUNCTION__,IXOFF);
  //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s    IXANY[%08X]\n", __LINE__,
__FUNCTION__,IXANY);
  switch (flowctl)
  {
    case 'N':
    case 'n':
    {
      if(ptSerialCfg->newtio.c_cflag & CRTSCTS)
      {
        ptSerialCfg->newtio.c_cflag = ptSerialCfg->newtio.c_cflag ^ CRTSCTS;
      }
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s OK  FlowCtl[%c] No
ptSerialCfg->newtio.c_cflag[%04X]\n", __LINE__, __FUNCTION__, flowctl,
ptSerialCfg->newtio.c_cflag);
      break;
    }

    case 'S':
    case 's':
    {
      if((ptSerialCfg->newtio.c_cflag & CRTSCTS)== 0)
      {
        ptSerialCfg->newtio.c_cflag |= CRTSCTS;
```

```
      }
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s OK  FlowCtl[%c] Yes
ptSerialCfg->newtio.c_cflag[%04X] ON CRTSCTS[%04X]\n", __LINE__, __FUNCTION__,
flowctl, ptSerialCfg->newtio.c_cflag,CRTSCTS);
      break;
    }
    case 'H':
    case 'h':
    {
      ptSerialCfg->newtio.c_cflag |= (IXON | IXOFF | IXANY);
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s OK  FlowCtl[%c] Hsk
ptSerialCfg->newtio.c_cflag[%04X] ON IXON[%04X] IXOFF[%04X] IXANY[%04X]\n",
__LINE__, __FUNCTION__, flowctl, ptSerialCfg->newtio.c_cflag,IXON,IXOFF,IXANY);
      break;
    }
    case 'T':
    case 't': //Tougle

    {
      //TraceFileLog(slogUtilscom,MSGS, "[%04d] %s OK  FlowCtl[%c] Tgl
ptSerialCfg->newtio.c_cflag[%04X]\n", __LINE__, __FUNCTION__, flowctl,
ptSerialCfg->newtio.c_cflag);
      break;
    }
    default:
    {
      TraceFileLog(slogUtilscom,MSGS, "[%04d] %s ER  FlowCtl[%c] ptSerialCfg-
>newtio.c_cflag[%04X] Invalid Parameter!\n", __LINE__, __FUNCTION__, flowctl,
ptSerialCfg->newtio.c_cflag);
      return (FALSE);
    }
  }
#else
  switch(flowctl)
  {
    case 'N':
    case 'n':
    {
      ptSerialCfg->newtio.fRtsControl = RTS_CONTROL_DISABLE;
      break;
    }

    case 'S':
    case 's':
    {
      ptSerialCfg->newtio.fRtsControl = RTS_CONTROL_ENABLE;
      break;
    }

    case 'H':
    case 'h':
    {
      ptSerialCfg->newtio.fRtsControl = RTS_CONTROL_HANDSHAKE;
      break;
    }

    case 'T':
    case 't':
    {
      ptSerialCfg->newtio.fRtsControl = RTS_CONTROL_TOGGLE;
      break;
    }
```

```
    default:
    {
      TraceFileLog(slogUtilscom, MSGS, "[%04d] %s ER FlowControl[%c]=[%02X]
Invalid Parameter!\n", __LINE__, __FUNCTION__, flowctl, ptSerialCfg-
>newtio.fRtsControl);
      return (FALSE);
    }
  }
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s FlowControl[%c]=[%02X]\n",
__LINE__, __FUNCTION__, flowctl, ptSerialCfg->newtio.fRtsControl);
#endif
  return (TRUE);
}

/*
 * Clear To Send end data set ready
 * Param cts values :
 *         return 0 - not clear;
 *         return 1 - clear;
 * Param dsr values :
 *         return 0 - not ready;
 *         return 1 - ready;
 * Return values :
 *         RETURN_OK - complete
 *         anything else is error
 */
int GetCTSDSR(HANDLE devHandle, int *piCTS, int *piDSR ,SERIAL_CFG * ptSerialCfg)
{
  /*int rc = 0, status = 0;
  *cts = 0;
  *dsr = 0;
#ifdef __LINUX__
  if (devHandle < 0)
  {
    return -1;
  }

  rc = ioctl(devHandle, TIOCMGET, &status);
  if (rc < 0)
  {
    return -2;
  }
  else
  {
    if ((status & TIOCM_CTS) == TIOCM_CTS)
    {
      *cts = 1;
    }

    if ((status & TIOCM_DSR) == TIOCM_DSR)
    {
      *dsr = 1;
    }
  }
#else
  TraceFileLog(slogUtilscom, MSGS, "[%04d] %s starting h[%04X] cts[%d] dsr[%d]
\n", __LINE__, __FUNCTION__, devHandle, *cts, *dsr);
  if (devHandle < 0)
  {
    rc = COMM_NO_HANDLER;
    TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER NotOpenned! h[%04X] RC[%d]\n",
```

```
__LINE__, __FUNCTION__, devHandle, rc);
    return rc;
  }

  if (!GetCommState(devHandle, &ptSerialCfg->newtio))
  {
    rc = COMM_INVALID_PARAMETER;
    TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER CommState h[%04X] RC[%d]\n",
__LINE__, __FUNCTION__, devHandle, rc);
    return rc;
  }
  else
  {
#ifdef __LINUX__
#else
    if (ptSerialCfg->newtio.fOutxCtsFlow == TRUE)
    {
      *cts = 1;
    }
    if (ptSerialCfg->newtio.fOutxDsrFlow == TRUE)
    {
      *dsr = 1;
    }
#endif
  }
#endif
  rc = COMM_RETURN_OK;
  TraceFileLog(slogUtilscom, MSGS, "[%04d] %s OK h[%04X] cts[%d] dsr[%d] \n",
__LINE__, __FUNCTION__, devHandle, *cts, *dsr);
  return rc;*/

      int rc;

      DWORD dwModemStatus = 0x00;
        if(GetCommModemStatus(devHandle,&dwModemStatus))
        {
          // Microsoft Windows dwModemStatus Signals
          // MS_CTS_ON   0x0010 The CTS (clear-to-send) signal is on.
          // MS_DSR_ON   0x0020 The DSR (data-set-ready) signal is on.
          // MS_RING_ON  0x0040 The ring indicator signal is on.
          // MS_RLSD_ON  0x0080 The RLSD (receive-line-signal-detect) signal is
on.
          if(dwModemStatus & MS_DSR_ON)
          {
            *piDSR = 1;  //DSR=ON
            rc += 1;
          }
          if(dwModemStatus & MS_CTS_ON)
          {
            *piCTS = 1;  //CTS=ON
            rc += 2;
          }
          if(dwModemStatus & MS_RING_ON)
          {
            rc += 4;   //RI=ON
          }
          if(dwModemStatus & MS_RLSD_ON)
          {
            rc += 8;   //RSDL=ON
          }
        }
        return 0;
```

```
}



/*
 * Set request to send end data terminal ready
 * Param rts values :
 *         0 - data terminal not ready;
 *         1 - reverse operation;
 * Param dtr values :
 *         0 - data terminal not ready;
 *         1 - data terminal ready;
 * Return values :
 *         RETURN_OK - complete
 *         anything else is error
 */
int SetRTSDTR(HANDLE devHandle, int rts, int dtr,SERIAL_CFG * ptSerialCfg)
{
  int rc = COMM_OPEN_ERROR;
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s starting h[%04X] rts[%d] dtr[%d]
\n", __LINE__, __FUNCTION__, devHandle, rts, dtr);
  if (devHandle < 0)
  {
    return COMM_OPEN_ERROR;
  }
#ifdef __LINUX__
  int status = 0;
  int old_status = 0;

  if (devHandle < 0)
  {
    return -1;
  }

  rc = ioctl(devHandle, TIOCMGET, &status);
  if (rc < 0)
  {
    TraceFileLog(slogUtilscom,MSGS, "[%04d] %s ER h[%04X] rts[%d] dtr[%d] Fail to
GetIOStatus RC[%d]\n", __LINE__, __FUNCTION__, devHandle, rts, dtr, rc);
    return -2;
  }

  old_status = status;

  switch (dtr)
  {
    case 0:
    {
      status &= ~TIOCM_DTR;
      break;
    }

    case 1:
    {
      status |= TIOCM_DTR;
      break;
    }

    default:
    {
      break;
    }
```

```
  }
  switch (rts)
  {
    case 0:
    {
      status &= ~TIOCM_RTS;
      break;
    }

    case 1:
    {
      status |= TIOCM_RTS;
      break;
    }

    default:
    {
      break;
    }
  }
  if (old_status != status)
  {
    rc = ioctl(devHandle, TIOCMSET, &status);
    if (rc < 0)
    {
      TraceFileLog(slogUtilscom,MSGS, "[%04d] %s ER h[%04X] rts[%d] dtr[%d]  Fail
to SetIOStatus RC[%d]\n", __LINE__, __FUNCTION__, devHandle, rts, dtr, rc);
      return -5;
    }
  }
#else
  switch(rts)
  {
    case 0:
    {
      rc = EscapeCommFunction(devHandle, CLRRTS); //Fix: Oliveira/NiloPain
14/12/2010
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s CLRRTS h[%04X] rts[%d] dtr[%d]
RC[%d]\n", __LINE__, __FUNCTION__, devHandle, rts, dtr, rc);
      break;
    }
    case 1:
    {
      rc = EscapeCommFunction(devHandle, SETRTS); //Fix: Oliveira/NiloPain
14/12/2010
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s SETRTS h[%04X] rts[%d] dtr[%d]
RC[%d]\n", __LINE__, __FUNCTION__, devHandle, rts, dtr, rc);
      break;
    }
  }
  if (rc < 0)
  {
    TraceFileLog(slogUtilscom, MSGS, "[%04d] %s ER RTS h[%04X] rts[%d] dtr[%d]
RC[%d]\n", __LINE__, __FUNCTION__, devHandle, rts, dtr, rc);
  }
  switch(dtr)
  {
    case 0:
    {
      rc = EscapeCommFunction(devHandle, CLRDTR); //Fix: Oliveira/NiloPain
14/12/2010
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s CLRDTR h[%04X] rts[%d] dtr[%d]
```

```
RC[%d]\n", __LINE__, __FUNCTION__, devHandle, rts, dtr, rc);
      break;
    }
    case 1:
    {
      rc = EscapeCommFunction(devHandle, SETDTR); //Fix: Oliveira/NiloPain
14/12/2010
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s SETDTR h[%04X] rts[%d] dtr[%d]
RC[%d]\n", __LINE__, __FUNCTION__, devHandle, rts, dtr, rc);
      break;
    }
  }
#endif
  if (rc < 0)
  {
    TraceFileLog(slogUtilscom, MSGS, "[%04d] %s ER DTR h[%04X] rts[%d] dtr[%d]
RC[%d]\n", __LINE__, __FUNCTION__, devHandle, rts, dtr, rc);
  }
  return rc;
}

/*
 * data terminal ready (DTR)
 * Param dtr values :
 *         0 - data terminal not ready;
 *         1 - data terminal ready;
 * Return values :
 *         RETURN_OK - complete
 *         anything else is error
 */
int SetDTR(HANDLE devHandle, int dtr,SERIAL_CFG * ptSerialCfg)
{
  int rc = COMM_OPEN_ERROR;
  if (devHandle < 0)
  {
    TraceFileLog(slogUtilscom, ERR, "[%04d] %s h[%04X] dtr[%d] Invalid Handler\n",
__LINE__, __FUNCTION__, devHandle, dtr);
    return COMM_OPEN_ERROR;
  }
#ifdef __LINUX__
  int status = 0;
  int old_status = 0;
  rc = ioctl(devHandle, TIOCMGET, &status);
  if (rc < 0)
  {
    TraceFileLog(slogUtilscom,MSGS, "[%04d] %s ER     h[%04X] dtr[%d] Fail to
GetIOStatus RC[%d]\n", __LINE__, __FUNCTION__, devHandle, dtr, rc);
    return rc;
  }

  old_status = status;

  switch (dtr)
  {
    case 0:
    {
      status &= ~TIOCM_DTR;
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s CLRDTR h[%04X] dtr[%d]
status[%04X]\n", __LINE__, __FUNCTION__, devHandle, dtr, status);
      break;
    }
```

```
    case 1:
    {
      status |= TIOCM_DTR;
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s SETDTR h[%04X] dtr[%d]
status[%04X]\n", __LINE__, __FUNCTION__, devHandle, dtr, status);
      break;
    }

    default:
    {
      break;
    }
  }
  if (old_status != status)
  {
    rc = ioctl(devHandle, TIOCMSET, &status);
    if (rc < 0)
    {
      TraceFileLog(slogUtilscom,MSGS, "[%04d] %s ER h[%04X] dtr[%d]  Fail to
SetIOStatus RC[%d]\n", __LINE__, __FUNCTION__, devHandle, dtr, rc);
      return -5;
    }
  }
#else
  switch(dtr)
  {
    case 0:
    {
      rc = EscapeCommFunction(devHandle, CLRDTR); //Fix: Oliveira/NiloPain
14/12/2010
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s CLRDTR h[%04X] dtr[%d]
RC[%d]\n", __LINE__, __FUNCTION__, devHandle, dtr, rc);
      break;
    }
    case 1:
    {
      rc = EscapeCommFunction(devHandle, SETDTR); //Fix: Oliveira/NiloPain
14/12/2010
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s SETDTR h[%04X] dtr[%d]
RC[%d]\n", __LINE__, __FUNCTION__, devHandle, dtr, rc);
      break;
    }
  }
#endif
  if (rc < 0)
  {
    TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER DTR h[%04X] dtr[%d] rc[%d]\n",
__LINE__, __FUNCTION__, devHandle, dtr, rc);
  }
  else
  {
    //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s OK DTR h[%04X] dtr[%d] rc[%d]
clr[%d] set[%d]\n", __LINE__, __FUNCTION__, devHandle, dtr, rc, CLRDTR, SETDTR);
  }
  return rc;
}

/*
 *Func que muda corretamente o DTR
 */
int SetDTR2(HANDLE devHandle, int dtr,SERIAL_CFG * ptSerialCfg)
{
```

```
  int rc = COMM_OPEN_ERROR;
  if (devHandle < 0)
  {
    TraceFileLog(slogUtilscom, ERR, "[%04d] %s h[%04X] dtr[%d] Invalid Handler\n",
__LINE__, __FUNCTION__, devHandle, dtr);
    return COMM_OPEN_ERROR;
  }
#ifdef __LINUX__
  int status = 0;
  int old_status = 0;
  rc = ioctl(devHandle, TIOCMGET, &status);
  if (rc < 0)
  {
    TraceFileLog(slogUtilscom,MSGS, "[%04d] %s ER     h[%04X] dtr[%d] Fail to
GetIOStatus RC[%d]\n", __LINE__, __FUNCTION__, devHandle, dtr, rc);
    return rc;
  }

  old_status = status;

  switch (dtr)
  {
    case 0:
    {
      status &= ~TIOCM_DTR;
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s CLRDTR h[%04X] dtr[%d]
status[%04X]\n", __LINE__, __FUNCTION__, devHandle, dtr, status);
      break;
    }

    case 1:
    {
      status |= TIOCM_DTR;
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s SETDTR h[%04X] dtr[%d]
status[%04X]\n", __LINE__, __FUNCTION__, devHandle, dtr, status);
      break;
    }

    default:
    {
      break;
    }
  }
  if (old_status != status)
  {
    rc = ioctl(devHandle, TIOCMSET, &status);
    if (rc < 0)
    {
      TraceFileLog(slogUtilscom,MSGS, "[%04d] %s ER h[%04X] dtr[%d]  Fail to
SetIOStatus RC[%d]\n", __LINE__, __FUNCTION__, devHandle, dtr, rc);
      return -5;
    }
  }
#else
  switch(dtr)
  {
    case 0:
    {
      //rc = EscapeCommFunction(devHandle, CLRDTR_EscCommFunc); //Fix:
Oliveira/NiloPain 14/12/2010
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s CLRDTR h[%04X] dtr[%d]
RC[%d]\n", __LINE__, __FUNCTION__, devHandle, dtr, rc);
```

```
      break;
    }
    case 1:
    {
      //rc = EscapeCommFunction(devHandle, SETDTR_EscCommFunc); //Fix:
Oliveira/NiloPain 14/12/2010
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s SETDTR h[%04X] dtr[%d]
RC[%d]\n", __LINE__, __FUNCTION__, devHandle, dtr, rc);
      break;
    }
  }
#endif
  if (rc < 0)
  {
    TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER DTR h[%04X] dtr[%d] rc[%d]\n",
__LINE__, __FUNCTION__, devHandle, dtr, rc);
  }
  else
  {
    //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s OK DTR h[%04X] dtr[%d]
rc[%d]\n", __LINE__, __FUNCTION__, devHandle, dtr, rc);
  }
  return rc;
}

/*
 * Set request to send (RTS)
 * Param rts values :
 *         0 - data terminal not ready;
 *         1 - reverse operation;
 * Return values :
 *         RETURN_OK - complete
 *         anything else is error
 */
int SetRTS(HANDLE devHandle, int rts,SERIAL_CFG * ptSerialCfg)
{
  int rc = COMM_OPEN_ERROR;
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s starting h[%04X] rts[%d] \n",
__LINE__, __FUNCTION__, devHandle, rts);
  if (devHandle < 0)
  {
    TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER h[%04X] rts[%d] Invalid
Handler\n", __LINE__, __FUNCTION__, devHandle, rts);
    return COMM_OPEN_ERROR;
  }
#ifdef __LINUX__
  int status = 0;
  int old_status = 0;

  if (devHandle < 0)
  {
    return -1;
  }

  rc = ioctl(devHandle, TIOCMGET, &status);
  if (rc < 0)
  {
    TraceFileLog(slogUtilscom,MSGS, "[%04d] %s ER h[%04X] rts[%d] Fail to
GetIOStatus RC[%d]\n", __LINE__, __FUNCTION__, devHandle, rts, rc);
    return rc;
  }
```

```
  old_status = status;
  switch (rts)
  {
    case 0:
    {
      status &= ~TIOCM_RTS;
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s CLRRTS h[%04X] rts[%d]
status[%04X]\n", __LINE__, __FUNCTION__, devHandle, rts,status);
      break;
    }

    case 1:
    {
      status |= TIOCM_RTS;
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s SETRTS h[%04X] rts[%d]
status[%04X]\n", __LINE__, __FUNCTION__, devHandle, rts,status);
      break;
    }

    default:
    {
      TraceFileLog(slogUtilscom, MSGS, "[%04d] %s ???RTS h[%04X] rts[%d]
status[%04X]\n", __LINE__, __FUNCTION__, devHandle, rts,status);
      break;
    }
  }
  if (old_status != status)
  {
    rc = ioctl(devHandle, TIOCMSET, &status);
    if (rc < 0)
    {
      TraceFileLog(slogUtilscom,MSGS, "[%04d] %s ER h[%04X] rts[%d] Fail to
SetIOStatus RC[%d]\n", __LINE__, __FUNCTION__, devHandle, rts, rc);
      return rc;
    }
  }
#else
  switch(rts)
  {
    case 0:
    {
      rc = EscapeCommFunction(devHandle, CLRRTS); //Fix: Oliveira/NiloPain
14/12/2010
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s CLRRTS h[%04X] rts[%d]
RC[%d]\n", __LINE__, __FUNCTION__, devHandle, rts, rc);
      break;
    }
    case 1:
    {
      rc = EscapeCommFunction(devHandle, SETRTS); //Fix: Oliveira/NiloPain
14/12/2010
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s SETRTS h[%04X] rts[%d]
RC[%d]\n", __LINE__, __FUNCTION__, devHandle, rts, rc);
      break;
    }
  }
#endif
  if (rc < 0)
  {
    TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER RTS h[%04X] rts[%d] RC[%d]\n",
__LINE__, __FUNCTION__, devHandle, rts, rc);
  }
```

```
  else
  {
    //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s OK RTS h[%04X] rts[%d]
RC[%d]\n", __LINE__, __FUNCTION__, devHandle, rts, rc);
  }
  return rc;
}

/*
 * Flush serial communication channel
 * Param channel values :
 *         0 - flush both;
 *         1 - flush data received but not read;
 *         2 - flush data written but not transmitted
 * Return values :
 *         RETURN_OK - complete
 *         anything else is error
 */
int FlushSerialComm(HANDLE devHandle, int channel,SERIAL_CFG * ptSerialCfg)
{
  BOOL rc;
  switch(channel)
  {
    case 0:
      rc = PurgeComm(devHandle, (PURGE_RXABORT | PURGE_RXCLEAR | PURGE_TXABORT |
PURGE_TXCLEAR));
      //TraceFileLog(slogUtilscom, ERR, "[%04d] %s BOTH h[%04X] ch[%04d]
rc[%d]\n", __LINE__, __FUNCTION__, devHandle, channel, rc);
      break;
    case 1:
      rc = PurgeComm(devHandle, (PURGE_RXABORT | PURGE_RXCLEAR));
      //TraceFileLog(slogUtilscom, ERR, "[%04d] %s RX   h[%04X] ch[%04d]
rc[%d]\n", __LINE__, __FUNCTION__, devHandle, channel, rc);
      break;
    case 2:
      rc = PurgeComm(devHandle, (PURGE_TXABORT | PURGE_TXCLEAR));
      //TraceFileLog(slogUtilscom, ERR, "[%04d] %s TX   h[%04X] ch[%04d]
rc[%d]\n", __LINE__, __FUNCTION__, devHandle, channel, rc);
      break;
    default:
      rc = -1;
      //TraceFileLog(slogUtilscom, ERR, "[%04d] %s ER?  h[%04X] ch[%04d]
rc[%d]\n", __LINE__, __FUNCTION__, devHandle, channel, rc);
      break;
  }
  return rc;
}

/*
 * Determine the amount of data that can be read atomically from socket
 * Param dataSize :
 *         return size of data that can be read.
 * Return values :
 *         RETURN_OK - complete
 *         anything else is error
 */
int GetSerialCount(HANDLE devHandle, int *dataSize,SERIAL_CFG * ptSerialCfg)
{
  int rc = COMM_UNKNOWN_ERROR;
  *dataSize = 0;
  TraceFileLog(slogUtilscom, MSGS, "[%04d] %s starting h[%04X] starting\n",
__LINE__, __FUNCTION__, devHandle);
```

```
  if (devHandle > 0)
  {
    // Não existe implementação semelhante para Windows
    #ifdef __WINDOWS__
    TraceFileLog(slogUtilscom, MSGS, "[%04d] %s OK WindowsDoNotHave DataSize
h[%04X] starting\n", __LINE__, __FUNCTION__, devHandle);
    rc = COMM_RETURN_OK;
    #else
    #endif
  }
  else
  {
    rc = COMM_NO_HANDLER;
  }
  TraceFileLog(slogUtilscom, MSGS, "[%04d] %s Er h[%04X] RC[%d]\n", __LINE__,
__FUNCTION__, devHandle, rc);
  return rc;
}

/*Adicionado por Arthur Morales Sampaio - CTS - 18-05-2011*/
int SetCTS(HANDLE devHandle,SERIAL_CFG * ptSerialCfg)
{
  int rc;
  TraceFileLog(slogUtilscom, MSGS, "[%04d] %s starting h[%04X]\n", __LINE__,
__FUNCTION__, devHandle);
#ifdef __LINUX__
#else
  ptSerialCfg->newtio.fOutxCtsFlow = 1;
#endif
  rc = SetCommState(devHandle, &ptSerialCfg->newtio);
  TraceFileLog(slogUtilscom, MSGS, "[%04d] %s End h[%04X] RC[%d]\n", __LINE__,
__FUNCTION__, devHandle, rc);
  return rc;
}

#ifdef __LINUX__
int SetCommState(HANDLE devHandle, struct termios * newtio)
{
  return tcsetattr(devHandle,TCSANOW, newtio);
}
int GetCommState(HANDLE devHandle, struct termios * newtio)
{
  return tcgetattr(devHandle, newtio);
}
int SetCommTimeouts( HANDLE devHandle, COMMTIMEOUTS * timeouts, SERIAL_CFG
*ptSerialCfg)
{
  //struct termios ptSerialCfg->oldtio;
  int rc;
  rc = tcgetattr(devHandle,&ptSerialCfg->oldtio);
  if (rc == 0)
  {
    //TraceFileLog(slogUtilscom, WRN, "[%04d] %s h[%04x] OK tcgetattr timeouts-
>ReadIntervalTimeout[%d]ms(Linux)\n", __LINE__, __FUNCTION__, devHandle, timeouts-
>ReadIntervalTimeout);
    ptSerialCfg->oldtio.c_cc[VTIME] = timeouts->ReadIntervalTimeout;
    ptSerialCfg->oldtio.c_cc[VMIN] = 0;
    rc = tcsetattr(devHandle,TCSANOW,&ptSerialCfg->oldtio);
    if (rc != 0)
    {
      TraceFileLog(slogUtilscom, MSGS, "[%04d] %s h[%04x] ER tcsetattr
rc[%d](Linux)\n", __LINE__, __FUNCTION__, devHandle,rc);
```

```
    }
    else
    {
      //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s h[%04x] OK tcgetattr
rc[%d](Linux)\n", __LINE__, __FUNCTION__, devHandle,rc);
    }
  }
  else
  {
    TraceFileLog(slogUtilscom, MSGS, "[%04d] %s h[%04x] ER tcgetattr
rc[%d](Linux)\n", __LINE__, __FUNCTION__, devHandle,rc);
  }
  return rc;
}
int GetCommTimeouts( HANDLE devHandle, COMMTIMEOUTS * timeouts, SERIAL_CFG
*ptSerialCfg)
{
  //struct termios ptSerialCfg->oldtio;
  int rc;
  if (devHandle <0)
  {
    TraceFileLog(slogUtilscom, MSGS, "[%04d] %s h[%04x]<0 (Linux)\n", __LINE__,
__FUNCTION__, devHandle);
    return -111;
  }
  if (!timeouts)
  {
    TraceFileLog(slogUtilscom, MSGS, "[%04d] %s h[%04x] timeouts=NULL(Linux)\n",
__LINE__, __FUNCTION__, devHandle);
    return -112;
  }
  if (!ptSerialCfg)
  {
    TraceFileLog(slogUtilscom, MSGS, "[%04d] %s h[%04x]<0
ptSerialCfg=NULL(Linux)\n", __LINE__, __FUNCTION__, devHandle);
    return -113;
  }
  if (&ptSerialCfg->oldtio == NULL)
  {
    TraceFileLog(slogUtilscom, MSGS, "[%04d] %s h[%04x]<0 ptSerialCfg-
>oldtio=NULL(Linux)\n", __LINE__, __FUNCTION__, devHandle);
    return -114;
  }
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s h[%04x] &ptSerialCfg->oldtio[%ld]
*ptSerialCfg->oldtio[%d] ptSerialCfg->oldtio[%d]\n", __LINE__, __FUNCTION__,
devHandle, &ptSerialCfg->oldtio, ptSerialCfg->oldtio);
  rc = tcgetattr(devHandle,&ptSerialCfg->oldtio);
  //TraceFileLog(slogUtilscom, MSGS, "[%04d] %s h[%04x] ptSerialCfg-
>oldtio.c_cc[VTIME][%d] rc[%d](Linux)\n", __LINE__, __FUNCTION__,
devHandle,ptSerialCfg->oldtio.c_cc[VTIME],rc);
  if (rc == 0)
  {
    timeouts->ReadIntervalTimeout = ptSerialCfg->oldtio.c_cc[VTIME];
    timeouts->ReadTotalTimeoutMultiplier = 0;
    timeouts->ReadTotalTimeoutConstant = ptSerialCfg->oldtio.c_cc[VTIME];
    timeouts->WriteTotalTimeoutMultiplier = 0;
    timeouts->WriteTotalTimeoutConstant = ptSerialCfg->oldtio.c_cc[VTIME];
  }
  else
  {
    TraceFileLog(slogUtilscom, MSGS, "[%04d] %s h[%04x] ptSerialCfg-
>oldtio.c_cc[VTIME][%d] rc[%d](Linux)\n", __LINE__, __FUNCTION__,
```

```
devHandle,ptSerialCfg->oldtio.c_cc[VTIME],rc);
  }
  return rc;
}
//
// Windows > Linux Purge Serial Comm TX and or RX Buffers
//
int PurgeComm(HANDLE devHandle, DWORD dwFlags)
{
#ifdef __LINUX__
  if(dwFlags & PURGE_IN)
  {
    //Flushing RxBuffer
    tcflush(devHandle, TCIFLUSH);
  }
  if(dwFlags & PURGE_OUT)
  {
    //Flushing TxBuffer
    tcflush(devHandle, TCOFLUSH);
  }
#else
  DWORD purgeFlag = 0;
  if(flags & PURGE_IN)
  {
    //Flushing RxBuffer
    purgeFlag |= PURGE_RXCLEAR;
  }
  if(flags & PURGE_OUT)
  {
    //Flushing TxBuffer
    purgeFlag |= PURGE_TXCLEAR;
  }
  if(PurgeComm(devHandle, purgeFlag) == 0)
  {
    //Windows fails to Purge serial comm
    return -1;
  }
#endif
  return 0;
}

int EscapeCommFunction(HANDLE devHandle, int set)
{
  int status = 0;

  int old_status = 0;
  int rc = 0;

  if (devHandle < 0)
  {
    return -1;
  }

  rc = ioctl(devHandle, TIOCMGET, &status);
  if (rc < 0)
  {
    return -2;
  }

  old_status = status;

  switch (set)
```

```
  {
    case CLRDTR:
    {
      status &= ~TIOCM_DTR;
      break;
    }
    case SETDTR:
    {
      status |= TIOCM_DTR;
      break;
    }
    case CLRRTS:
    {
      status &= ~TIOCM_RTS;
      break;
    }
    case SETRTS:
    {
      status |= TIOCM_RTS;
      break;
    }
    default:
    {
      return -3;
    }
  }
  if (old_status != status)
  {
    rc = ioctl(devHandle, TIOCMSET, &status);
    if (rc < 0)
    {
      return -5;
    }
  }
  return 0;
}
// Windows to Linux Function Transformation
//This is a CommPort Close routine
//it transforms the Windows CloseHandle() function to Linux Close() function
int CloseHandle(HANDLE devHandle)
{
  close(devHandle);
  devHandle = 0;
  return 0;
}
#endif
//
// Serial Conversion Functions (Windows <> Linux)
//

//Convert Windows BaudRate to Linux   BaudRate or
//Convert Linux   BaudRate to Windows BaudRate
int devbr_to_OSbr(int devbr)
{
#ifdef __LINUX__
  switch(devbr)
  {
    case DEVBR_110: return B110;
    case DEVBR_300: return B300;
    case DEVBR_600: return B600;
    case DEVBR_1200: return B1200;
    case DEVBR_2400: return B2400;
```

```
    case DEVBR_4800: return B4800;
    case DEVBR_9600: return B9600;
    /*case DEVBR_14400:      return B14400;*/
    case DEVBR_19200: return B19200;
    case DEVBR_38400: return B38400;
    /*case DEVBR_56000:      return B56000;*/
    case DEVBR_57600: return B57600;
    case DEVBR_115200: return B115200;
    /*case DEVBR_128000:      return B128000;*/
    /*case DEVBR_256000:      return B256000;*/

    default:
    return -1;
  }
#else
  switch(devbr)
  {
    case DEVBR_110:
      return CBR_110;
    case DEVBR_300:
      return CBR_300;
    case DEVBR_600:
      return CBR_600;
    case DEVBR_1200:
      return CBR_1200;
    case DEVBR_2400:
      return CBR_2400;
    case DEVBR_4800:
      return CBR_4800;
    case DEVBR_9600:
      return CBR_9600;
    case DEVBR_14400:
      return CBR_14400;
    case DEVBR_19200:
      return CBR_19200;
    case DEVBR_38400:
      return CBR_38400;
    case DEVBR_56000:
      return CBR_56000;
    case DEVBR_57600:
      return CBR_57600;
    case DEVBR_115200:
      return CBR_115200;
    case DEVBR_128000:
      return CBR_128000;
    case DEVBR_256000:
      return CBR_256000;
    default:
      return -1;
  }
#endif
}
//Convert Windows Parity to Linux   Parity or
//Convert Linux   Parity to Windows Parity
int devpar_to_OSpar(char devpar)
{
#ifdef __LINUX__
  switch(devpar)
  {
    case 'E': return PARENB | ~PARODD;
    case 'N': return ~PARENB;
    case 'O': return PARENB | PARODD;
```

```
    default:
    return -1;
  }
#else
  switch(devpar)
  {
    case 'E':
      return EVENPARITY;
    case 'M':
      return MARKPARITY;
    case 'N':
      return NOPARITY;
    case 'O':
      return ODDPARITY;

    default:
      return -1;
  }
#endif
}
//Convert Windows StopBits to Linux   StopBits or
//Convert Linux   StopBits to Windows StopBits
int devsb_to_OSsb(int devsb)
{
#ifdef __LINUX__
  switch(devsb)
  {
    case 1: return ~CSTOPB;
    case 2: return CSTOPB;

    default:
    return -1;
  }
#else
  switch(devsb)
  {
    case 1:
      return ONESTOPBIT;
    case 15:
      return ONE5STOPBITS;
    case 2:
      return TWOSTOPBITS;

    default:
      return -1;
  }
#endif
}
#ifdef __LINUX__
//Convert Windows Device BautDare to Linux Baud rate
int devdb_to_OSdb(int devdb)
{
  switch(devdb)
  {
    case 5: return CS5;
    case 6: return CS6;
    case 7: return CS7;
    case 8: return CS8;

    default:
    return -1;
```

```
  }

}
#endif
char * GetCommErrorDetail(int iError)
{
  int i;
  memset(glstrErrorDetail,0,sizeof(glstrErrorDetail));
  for (i=0; i < 30  ;i++) //Corrected by Klocwork
  {
      if (iError == tCommError[i].iError)
      {
        sprintf(glstrErrorDetail,"%s
[%s]",tCommError[i].pstrErDesc,tCommError[i].pstrErName);
        break;
      }
      if(tCommError[i].pstrErName == NULL)
      {
        sprintf(glstrErrorDetail,"Erro Nao Encontrado Id[%d]",iError);
        break;
      }
  }
  return glstrErrorDetail;
}


#ifdef __WINDOWS__
void my_usleep(int waitTime) {
    __int64 time1 = 0, time2 = 0, freq = 0;

    QueryPerformanceCounter((LARGE_INTEGER *) &time1);
    QueryPerformanceFrequency((LARGE_INTEGER *)&freq);

    do {
        QueryPerformanceCounter((LARGE_INTEGER *) &time2);
    } while((time2-time1) < waitTime);
}
#endif

/*int my_usleep(unsigned long uSec) {
       nsleep(uSec*1000);
       return 0;
}

void nsleep(long us)
{
    struct timespec wait;

    //printf("Will sleep for is %ld\n", diff); //This will take extra ~70
microseconds
    wait.tv_sec = us / (1000 * 1000);
    wait.tv_nsec = us * 1000;
    nanosleep(&wait, NULL);
}*/
```

commserial.h

```
#include <commerr.h>
#include <utils.h>

#ifdef __WIN32__
#include <windows.h>
#else
#include <termios.h>
#endif
#ifdef __LINUX__
#define FILE_ATTRIBUTE_NORMAL 0
#define PURGE_TXABORT         0
#define PURGE_RXABORT         0
#define PURGE_IN      0x01
#define PURGE_OUT     0x02

#define RTS_CONTROL_ENABLE    0
#define RTS_CONTROL_TOGGLE    0
#define RTS_CONTROL_HANDSHAKE 0
#define PURGE_TXCLEAR 0x0004
#define PURGE_RXCLEAR 0x0008
#define SETDTR 1
#define SETRTS 2
#define CLRDTR 4
#define CLRRTS 8
#define RTS_CONTROL_DISABLE 0x0008  //Added By Oliveira

typedef struct _DCB
{
  DWORD DCBlength;
  DWORD BaudRate;
  DWORD fFlags;
  WORD wReserved;
  WORD XonLim;
  WORD XoffLim;
  BYTE ByteSize;
  BYTE Parity;
  BYTE StopBits;
  char XonChar;
  char XoffChar;
  char ErrorChar;
  char EofChar;
  char EvtChar;
  WORD wReserved1;
  WORD fRtsControl; //Added By Oliveira
  WORD fOutxCtsFlow; //Added By Oliveira
  WORD fOutxDsrFlow; //Added By Oliveira
}DCB,*LPDCB;

typedef struct _COMMTIMEOUTS
{
  DWORD ReadIntervalTimeout;
  DWORD ReadTotalTimeoutMultiplier;
  DWORD ReadTotalTimeoutConstant;
  DWORD WriteTotalTimeoutMultiplier;
  DWORD WriteTotalTimeoutConstant;
}COMMTIMEOUTS,*LPCOMMTIMEOUTS;
```

```
typedef enum
{
  DEVBR_110,
  DEVBR_300,
  DEVBR_600,
  DEVBR_1200,
  DEVBR_2400,
  DEVBR_4800,
  DEVBR_9600,
  DEVBR_14400,
  DEVBR_19200,
  DEVBR_38400,
  DEVBR_56000,
  DEVBR_57600,
  DEVBR_115200,
  DEVBR_128000,
  DEVBR_256000
}DEV_BR;

//excerpt from kernel32
#define INVALID_ATOM (0)
#define IGNORE      0
#define INFINITE    0xFFFFFFFF
#define NOPARITY    0
#define ODDPARITY   1
#define EVENPARITY  2
#define MARKPARITY  3
#define SPACEPARITY 4
#define ONESTOPBIT  0
#define ONE5STOPBITS      1
#define TWOSTOPBITS 2
#define CBR_110     110
#define CBR_300     300
#define CBR_600     600
#define CBR_1200    1200
#define CBR_2400    2400
#define CBR_4800    4800
#define CBR_9600    9600
#define CBR_14400   14400
#define CBR_19200   19200
#define CBR_38400   38400
#define CBR_56000   56000
#define CBR_57600   57600
#define CBR_115200  115200
#define CBR_128000  128000
#define CBR_256000  256000
int SetCommState(HANDLE devHandle, struct termios * newtio);
int GetCommState(HANDLE devHandle, struct termios * newtio);
int PurgeComm(HANDLE devHandle, DWORD dwFlags);
int EscapeCommFunction(HANDLE devHandle, int set);
int CloseHandle(HANDLE devHandle); //OK Linux Close
#endif

/*
 * Serial configuration struct
 */
typedef struct
{
  unsigned char cTop;
  char device[128]; //corrected by Klockwork
  char model[32];
```

```
  long int baudrate;
  char flowctl;
  char parity;
  int databits;
  int stopbits;
  BOOL blocking;
  #ifdef __LINUX__
  struct termios oldtio, newtio;
  #else
  DCB oldtio, newtio; //Deve estar no arquivo config.c
  #endif
  unsigned char cBot;
} SERIAL_CFG;
#ifdef __LINUX__
int SetCommTimeouts( HANDLE devHandle, COMMTIMEOUTS * timeouts, SERIAL_CFG
*serialCfg);
int GetCommTimeouts( HANDLE devHandle, COMMTIMEOUTS * timeouts, SERIAL_CFG
*serialCfg);
#endif
typedef struct tErrorStruct
{
      int iError;
      char * pstrErName;
      char * pstrErDesc;
}tCommErrorTable;
static tCommErrorTable tCommError[30] =
//iError                        strErName                       strErDesc
{{COMM_RETURN_OK,                "COMM_RETURN_OK",               "OK"},\
 {COMM_IO_ERROR,                 "COMM_IO_ERROR",                "Erro E/S"},\
 {COMM_OPEN_ERROR,               "COMM_OPEN_ERROR",              "Erro Abertura
Comm"},\
 {COMM_INVALID_COMMAND,          "COMM_INVALID_COMMAND",         "Erro Comando
Invalido"},\
 {COMM_INVALID_PARAMETER,       "COMM_INVALID_PARAMETER",       "Erro Param.
Invalido"},\
 {COMM_RSP_FORMAT_ERROR,        "COMM_RSP_FORMAT_ERROR",        "Erro Formato da
Resposta Invalido"},\
 {COMM_RSP_CKSUM_ERROR,         "COMM_RSP_CKSUM_ERROR",         "Erro checksum da
Resposta incorreto"},\
 {COMM_TX_IO_ERROR,             "COMM_TX_IO_ERROR",             "Erro Transmissao"},\
 {COMM_RX_IO_ERROR,             "COMM_RX_IO_ERROR",             "Erro Recepcao"},\
 {COMM_RX_NAK_RECEIVED,         "COMM_RX_NAK_RECEIVED",         "Erro Recebeu NAK"},\
 {COMM_RX_DATA_BLOCK_TOO_BIG,   "COMM_RX_DATA_BLOCK_TOO_BIG",   "Erro Bloco muito
Grande"},\
 {COMM_RX_BYTE_STUFFING_ERROR, "COMM_RX_BYTE_STUFFING_ERROR","Erro Formato do
Bloco de recepcao invalido"},\
 {COMM_TXTIMEOUT,               "COMM_TXTIMEOUT",               "Erro Tempo Esgotado
- Transmissao"},\
 {COMM_RXTIMEOUT,               "COMM_RXTIMEOUT",               "Erro Tempo Esgotado
- Recepcao"},\
 {COMM_CFG_ERROR,               "COMM_CFG_ERROR",               "Erro de
Configuracao"},\
 {COMM_RXFLUSH_ERROR,           "COMM_RXFLUSH_ERROR",           "Erro Limpesa
Recepcao"},\
 {COMM_TXFLUSH_ERROR,           "COMM_TXFLUSH_ERROR",           "Erro Limpesa
Transmissao"},\
 {COMM_STATUS_ERROR,            "COMM_STATUS_ERROR",            "Erro Estado da
Maquina"},\
 {COMM_ACK_STATE_ERROR,         "COMM_ACK_STATE_ERROR",         "Erro na Maquina de
Estado"},\
 {COMM_ENQ_STATE_ERROR,         "COMM_ENQ_STATE_ERROR",         "Erro na Solicitacao
ENQ"},\
```

```
 {COMM_NO_HANDLER,              "COMM_NO_HANDLER",             "Porta Fechada"},\
 {COMM_NO_CMD_DATA,             "COMM_NO_CMD_DATA",            "Erro Comando sem
Dados"},\
 {COMM_MEMORY_ALLOC_FAIL,       "COMM_MEMORY_ALLOC_FAIL",      "Erro Alocacao de
Memoria"},\
 {COMM_DEVICE_BUSY,             "COMM_DEVICE_BUSY",            "Erro Dispositivo
Ocupado"},\
 {COMM_UNKNOWN_ERROR,            "COMM_UNKNOWN_ERROR",          "Erro
Desconhecido"},\
 {9999,                          NULL,                          NULL}};

char glstrErrorDetail[100]; //Global Error Details

//Config BaudRate,DataBits,Parity,StopBits,FlowControl into  DCB newtio struct
BOOL SetBaudrate(unsigned long int baudrate, SERIAL_CFG *ptCfg);
BOOL SetDataBits(int databits, SERIAL_CFG *ptCfg);
BOOL SetParity(char parity, SERIAL_CFG *ptCfg);
BOOL SetStopBits(int stopbits, SERIAL_CFG *ptCfg);
BOOL SetFlowControl(char flowctl, SERIAL_CFG *ptCfg);
//
// SERIAL COMM_FUNCTIONS
//
/*
 * Close serial communication
 * Param commHandler :
 *         communication handler
 * Return values :
 *         void
 */
void CloseSerialComm(HANDLE *devHandle, SERIAL_CFG *serialCfg);

/*
 * Clear To Send end data set ready
 * Param serialCfg:
 *         configuration values.
 * Param commHandler :
 *         return communication handler.
 * Return values :
 *         RETURN_OK - complete
 *         anything else is error
 */
int OpenSerialComm(HANDLE *devHandle, SERIAL_CFG *serialCfg);

/*
 * Clear To Send end data set ready
 * Param cts values :
 *         return 0 - not clear;
 *         return 1 - clear;
 * Param dsr values :
 *         return 0 - not ready;
 *         return 1 - ready;
 * Return values :
 *         RETURN_OK - complete
 *         anything else is error
 */
int GetCTSDSR(HANDLE devHandle, int *cts, int *dsr,SERIAL_CFG *ptCfg);

/*
 * Set request to send end data terminal ready
 * Param rts values :
 *         0 - data terminal not ready;
 *         1 - reverse operation;
```

```
 * Param dtr values :
 *         0 - data terminal not ready;
 *         1 - data terminal ready;
 * Return values :
 *         RETURN_OK - complete
 *         anything else is error
 */
int SetRTSDTR(HANDLE devHandle, int rts, int dtr, SERIAL_CFG *ptCfg);

/*
 * Flush serial communication channel
 * Param channel values :
 *         0 - flush both;
 *         1 - flush data received but not read;
 *         2 - flush data written but not transmitted
 * Return values :
 *         RETURN_OK - complete
 *         anything else is error
 */
int FlushSerialComm(HANDLE devHandle, int channel, SERIAL_CFG *ptCfg);

/*
 * Determine the amount of data that can be read atomically from socket
 * Param dataSize :
 *         return size of data that can be read.
 * Return values :
 *         RETURN_OK - complete
 *         anything else is error
 */
int GetSerialCount(HANDLE devHandle, int *dataSize, SERIAL_CFG *ptSerialCfg);

/*Adicionado por Arthur Morales Sampaio*/
int SetCTS(HANDLE devHandle, SERIAL_CFG *ptSerialCfg);

int ReadGetKeyData(HANDLE devHandle, int iBytesToRead, char *data, unsigned long
ulTimeout,SERIAL_CFG *ptSerialCfg);

/*Adicionado por Joao Marcelo T. e S. Torres*/
int ReadData(HANDLE devHandle, int szRXBuffer, char *data, unsigned long
ulTimeout,SERIAL_CFG *ptSerialCfg);

/*Adicionado por Joao Marcelo T. e S. Torres*/
int WriteData(HANDLE devHandle, int szTXBuffer, char *data, unsigned long
ulTimeout, SERIAL_CFG *ptSerialCfg);

/* Adicionado por Daniel Moraes Bittar para correção de ausência */
int SetRTSDTR(HANDLE devHandle, int rts, int dtr, SERIAL_CFG *ptSerialCfg);

/* Adicionado por Jose Carlos de Oliveira 18-05-2011 */
int SetDTR(HANDLE devHandle, int dtr, SERIAL_CFG *ptSerialCfg);

/* Adicionado por Danilo Almeida Maletta 31-05-2012*/
int SetDTR2(HANDLE devHandle, int dtr,SERIAL_CFG * ptSerialCfg);

/* Adicionado por Jose Carlos de Oliveira 18-05-2011 */
int SetRTS(HANDLE devHandle, int rts, SERIAL_CFG *ptSerialCfg);

//Concatenate [Header + CMD + Param + Folder + Checksum]  then Send >
DeviceHandler
int SendCMD(HANDLE devHandle, int iProtocol, char *pCMD, int iLen, char* pParam,
int iLenParam, unsigned long long ulTxTimeout, SERIAL_CFG *ptSerialCfg);
```

```
//Receive [Header + RSP)Rsult + Folder + Checksum], compute Checksum,
//        Remove Headers & Folder and return only pRspBuff data and it Len
int RecvRSP(HANDLE devHandle, int iProtocol, char* pRspBuff, int *iLen, unsigned
long long ulRxTimeout,SERIAL_CFG *ptSerialCfg);
int InsertCMDPayLoad(int iProtocol, char *pCMD_Parm, int *iLen,SERIAL_CFG
*ptSerialCfg);
int RemoveRSPPayLoad(int iProtocol, char *pRSPin, char *pRSPout, int
*iLen,SERIAL_CFG *ptSerialCfg);
int devbr_to_OSbr(int devbr);
int devpar_to_OSpar(char devpar);
int devsb_to_OSsb(int devsb);
int devdb_to_OSdb(int devdb);
char * GetCommErrorDetail(int iError);

/*Adicionado por Arthur Morales Sampaio
 * Wrapper para corrigir deprecation usleep. */
#ifdef __WINDOWS__
void my_usleep(int waitTime);
#endif
/*int my_usleep(unsigned long uSec);
void nsleep(long us);
*/
```

utils.c

```
/*****************************************************************************
 * Autor          : Arthur Morales Sampaio, Jose Carlos de Oliveira        *
 *****************************************************************************/
#include <utils.h>

bConsoleLogsEnabled = 1;
bTraceLogsEnabled = 1;

//
// Trace Log
// iLogType = 0  Log Disabled
//            1  ERROR    Error Only
//            2  WARNING  Error and Warnning only
//            3  MESSAGES Error, Warning and Function Menssages
//            4  TRACE
int TraceLog(int iLogType, char *format, ...)
{
#ifdef __LINUX__
      time_t now;
      struct tm st;
      time(&now);
      localtime_r(&now, &st);
      struct timespec tp;
      clock_gettime(CLOCK_REALTIME, &tp);
#else
      SYSTEMTIME st;
      GetSystemTime(&st);
#endif
      FILE *hLogFile = NULL;
      va_list args;
```

```
	if (iLogType != 0)
	{
		if (((bErrorLogsEnabled) && (iLogType == ERR))
				|| ((bWarningLogsEnabled) && (iLogType == WARNING))
				|| ((bMessageLogsEnabled) && (iLogType == MESSAGE))
				|| (bTraceLogsEnabled))
		{
#ifdef __LINUX__
			if ((st.tm_hour - 3) <= 0)
			{
				st.tm_hour = (st.tm_hour - 3) + 24;
			}
			else
			{
				st.tm_hour = st.tm_hour - 3;
			}
#else
			//Ajuste de Fuso Horario GMT -3 BRZ
			if ((st.wHour - 3) <= 0)
			{
				st.wHour = (st.wHour - 3) + 24;
			}
			else
			{
				st.wHour = st.wHour - 3;
			}
#endif
			hLogFile = fopen(DEFAULT_LOG_FILE_NAME, "a");
			if (hLogFile != NULL )
			{
#ifdef __LINUX__
				fprintf(hLogFile, "[%02d:%02d:%02d.%03d] ", st.tm_hour,
st.tm_min, st.tm_sec, (int) tp.tv_nsec / 1000000);
#else
				fprintf(hLogFile, "[%02d:%02d:%02d.%03d] ", st.wHour,
					st.wMinute, st.wSecond, st.wMilliseconds);
#endif
				va_start(args, format);
				vfprintf(hLogFile, format, args);
				va_end(args);
				fflush(hLogFile);
				fclose(hLogFile);
			}
		}
	}
	if (bConsoleLogsEnabled)
	{
		if (((bErrorLogsEnabled) && (iLogType == ERR))
				|| ((bWarningLogsEnabled) && (iLogType == WARNING))
				|| ((bMessageLogsEnabled) && (iLogType == MESSAGE))
				|| (bTraceLogsEnabled))
		{
			va_start(args, format);
			vprintf(format, args);
			va_end(args);
		}
	}
	return 0;
}

// Alteração no TraceLog! Arthur Morales Sampaio
// Consiste em Logar em arquivos diferentes!
```

```
//
// Trace Log
// iLogType = 0  Log Disabled
//            1  ERROR    Error Only
//            2  WARNING  Error and Warnning only
//            3  MESSAGES Error, Warning and Function Menssages
//            4  TRACE
int TraceFileLog(char * sLogFileName, int iLogType, char *format, ...)
{
#ifdef __LINUX__
      time_t now;
      struct tm st;
      time(&now);
      localtime_r(&now, &st);
      struct timespec tp;
      clock_gettime(CLOCK_REALTIME, &tp);
#else
      SYSTEMTIME st;
      GetSystemTime(&st);
#endif
      char strLogType[6];
      FILE *hLogFile = NULL;
      va_list args;
      if (iLogType != 0)
      {
            if (((bErrorLogsEnabled) && (iLogType == ERR))
                        || ((bWarningLogsEnabled) && (iLogType == WARNING))
                        || ((bMessageLogsEnabled) && (iLogType == MESSAGE))
                        || (bTraceLogsEnabled))
            {
                  switch (iLogType)
                  {
                  case ERR:
                        strcpy(strLogType, "Err");
                        break;
                  case WRN:
                        strcpy(strLogType, "Wrn");
                        break;
                  case INFO:
                        strcpy(strLogType, "Inf");
                        break;
                  case MSGS:
                        strcpy(strLogType, "Msg");
                        break;
                  case TRC:
                        strcpy(strLogType, "Trc");
                        break;
                  default:
                        strcpy(strLogType, "Unk");
                        break;
                        break;
                  }
#ifdef __LINUX__
                  if ((st.tm_hour - 3) <= 0)
                  {
                        st.tm_hour = (st.tm_hour - 3) + 24;
                  }
                  else
                  {
                        st.tm_hour = st.tm_hour - 3;
                  }
```

```
#else
                    //Ajuste de Fuso Horario GMT -3 BRZ
                    if ((st.wHour - 3) <= 0)
                    {
                          st.wHour = (st.wHour - 3) + 24;
                    }
                    else
                    {
                          st.wHour = st.wHour - 3;
                    }
#endif
                    hLogFile = fopen(sLogFileName, "a");
                    if (hLogFile != NULL )
                    {
#ifdef __LINUX__
                          fprintf(hLogFile, "[%02d:%02d:%02d.%03d]%s ",
st.tm_hour, st.tm_min, st.tm_sec, (int) tp.tv_nsec / 1000000,strLogType);
#else
                          fprintf(hLogFile, "[%02d:%02d:%02d.%03d]%s ", st.wHour,
                                st.wMinute, st.wSecond, st.wMilliseconds,
strLogType);
#endif
                          va_start(args, format);
                          vfprintf(hLogFile, format, args);
                          va_end(args);
                          fflush(hLogFile);
                          fclose(hLogFile);
                    }
             }
      }
      if (bConsoleLogsEnabled)
      {
             if (((bErrorLogsEnabled) && (iLogType == ERR))
                          || ((bWarningLogsEnabled) && (iLogType == WARNING))
                          || ((bMessageLogsEnabled) && (iLogType == MESSAGE))
                          || (bTraceLogsEnabled))
             {
                    va_start(args, format);
                    vprintf(format, args);
                    va_end(args);
             }
      }
      return 0;
}

//END ALTERAÇÃO TRACELOG!

//
=============================================================================
=
// Converts a AscII Hex Array to Binary Array
// at Entry: usAscIIHex    & iLen
// at  Exit: usBinaryArray & piOutLen
// Example: usAsciiHex    = '1234\x3A\x3B   iOutLen = 6
//          usBinaryArray = 0x12 0x34 0xAB  iBinLen = 3
//          if (ok)   ret = *piOutLen;
//          if (err)  ret = -1
//
=============================================================================
=
int convAsciiHexBin(unsigned char *pusAsciiHex, int iLen,
             unsigned char *pusBinArray, int *piOutLen)
```

```
{
	int i, o = 0;
	unsigned char ucHdig = 0;

	for (i = 0; i < iLen; i++)
	{
		if ((pusAsciiHex[i] >= '0') && (pusAsciiHex[i] <= '9'))
		{
			//AscII Number 0-9
			ucHdig = pusAsciiHex[i] & 0x0F;
		}
		else if (((pusAsciiHex[i] >= 'a') && (pusAsciiHex[i] <= 'f'))
				|| ((pusAsciiHex[i] >= 'A') && (pusAsciiHex[i] <= 'F')))
		{
			//AscII Chars A-F
			ucHdig = (pusAsciiHex[i] & 0x0F) + 9;
		}
		else
		{
			//Invalid AscII Hex
			*piOutLen = o;
			return -1;
		}
		if ((i & 0x00000001) == 0)
		{
			//UpperDigit (Do not Increment)
			pusBinArray[o] = ucHdig << 4;
		}
		else
		{
			//LowerDigit (Increment o now)
			pusBinArray[o++] |= ucHdig;
		}
	}
	*piOutLen = o;
	//TraceFileLog("bitinfo.log", MSGS, "[%04d] %s iLen[%d] *piOutLen[%d]
o[%d]\n", __LINE__, __FUNCTION__,iLen,*piOutLen,o);
	return o;
}

int str_removec(char * sData, char c)
{
	char * pch;

	char * sDataAux = (char*) calloc(1, strlen(sData));

	pch = strchr(sData, c);
	while (pch != NULL )
	{
		//printf("found at %d\n", pch - sData + 1);
		//printf("Substr to use 0-[%d]", pch - sData);
		strncpy(sDataAux, sData, pch - sData);

		pch = strchr(pch + 1, '\n');

		memset(sData, 0, sizeof(sData));
		strcpy(sData, sDataAux);

	}

	free(pch);
	free(sDataAux);
```

```
	return 0;
}

/*
 * Create a new string with [substr] being replaced by [replacement] in [string]
 * Returns the new string, or NULL if out of memory.
 * The caller is responsible for freeing this new string.
 */
char *
str_replace(const char *string, const char *substr, const char *replacement)
{
	char *tok = NULL;
	char *newstr = NULL;

	tok = strstr(string, substr);
	if (tok == NULL )
		return strdup(string);
	newstr = malloc(strlen(string) - strlen(substr) + strlen(replacement) + 1);
	if (newstr == NULL )
		return NULL ;
	memcpy(newstr, string, tok - string);
	memcpy(newstr + (tok - string), replacement, strlen(replacement));
	memcpy(newstr + (tok - string) + strlen(replacement), tok + strlen(substr),
			strlen(string) - strlen(substr) - (tok - string));
	memset(newstr + strlen(string) - strlen(substr) + strlen(replacement), 0,
			1);
	return newstr;
}

int file_exists(const char *fname)
{

	FILE *file;
	if (file = fopen(fname, "r"))
	{
		fclose(file);
		return 1;
	}
	return 0;

}
```

utils.h

```
/*****************************************************************************
 * Autor          : Arthur Morales Sampaio, Jose Carlos de Oliveira        *
 *****************************************************************************/
#ifndef LOG_H_
#define LOG_H_

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

#ifdef __WINDOWS__
//Windows Only C Libraryes
#include <windows.h>  //C Windows Library
#endif

#ifdef __LINUX__
#include <string.h>
#endif

#ifdef __LINUX__
#define DEFAULT_LOG_FILE_NAME  "/log/drv/cts_atm_startuplog.log"
#else
#define DEFAULT_LOG_FILE_NAME  "\\log\\drv\\cts_atm_startuplog.log"
#endif

#ifdef __WINDOWS__
//Linux to Windows Time Function
//extern void usleep( unsigned long usleepTime);
#define DEVBR_110    1
#define DEVBR_300    2
#define DEVBR_600    3
#define DEVBR_1200   4
#define DEVBR_2400   5
#define DEVBR_4800   6
#define DEVBR_9600   7
#define DEVBR_14400  8
#define DEVBR_19200  9
#define DEVBR_38400  10
#define DEVBR_56000  11
#define DEVBR_57600  12
#define DEVBR_115200 13
#define DEVBR_128000 14
#define DEVBR_256000 15
#endif

//TraceLog Definitions
FILE * logFile;
BOOL bConsoleLogsEnabled;
BOOL bErrorLogsEnabled;
BOOL bWarningLogsEnabled;
BOOL bMessageLogsEnabled;
BOOL bTraceLogsEnabled;
#define DISABLED 0     //No Log
#define ERR      1     //Error Logs Enabled
#define WARNING  2     //Warning Logs Enableds
```

```
#define WRN      2
#define MESSAGE  3     //Message Logs Enableds
#define MSGS     3
#define INFO     4     //Message Logs Enableds
#define INF      4     //Message Logs Enableds
#define TRACE    5     //Trace (All Logs Enableds
#define TRC      6
#define DBG      7     //Debug Logs Enabled

int TraceLog(int iLogType, char *format, ...);
int TraceFileLog(char * sLogFileName, int iLogType, char *format, ...);
int convAsciiHexBin(unsigned char *pusAsciiHex, int iLen, unsigned char
*pusBinArray, int *piOutLen);
//Remove apenas uma ocorrencia do caractere solicitado!
int str_removec(char * sData, char c);
//Substitue a substring substr por replacement em string.
char * str_replace(const char *string, const char *substr, const char
*replacement);

int file_exists(const char *fname);


#endif /* LOG_H_ */
```

commerr.h

```
/*****************************************************************************
 * Autor          : Arthur Morales Sampaio, Jose Carlos de Oliveira        *
 *****************************************************************************/
#define COMM_RETURN_OK                          0
#define COMM_IO_ERROR                             -1
#define COMM_OPEN_ERROR                        -2
#define COMM_INVALID_COMMAND                   -3
#define COMM_INVALID_PARAMETER         -4
#define COMM_RSP_FORMAT_ERROR                  -5
#define COMM_RSP_CKSUM_ERROR                   -6
#define COMM_TX_IO_ERROR                       -7
#define COMM_RX_IO_ERROR                       -8
#define COMM_RX_NAK_RECEIVED             -9
#define COMM_RX_DATA_BLOCK_TOO_BIG      -10
#define COMM_RX_BYTE_STUFFING_ERROR     -11
#define COMM_TXTIMEOUT                         -12
#define COMM_RXTIMEOUT                         -13
#define COMM_CFG_ERROR                             -14
#define COMM_RXFLUSH_ERROR                        -15
#define COMM_TXFLUSH_ERROR                        -16
#define COMM_STATUS_ERROR              -17
#define COMM_ACK_STATE_ERROR             -18
#define COMM_ENQ_STATE_ERROR             -19
#define COMM_NO_HANDLER                  -20
#define COMM_NO_CMD_DATA                 -21
#define COMM_MEMORY_ALLOC_FAIL           -22
#define COMM_DEVICE_BUSY                 -23
#define COMM_UNKNOWN_ERROR                        -999
//Serial Printer Errors ESC-POS
//
// Serial Printer - Errors Code
```

```
//
#define SERIAL_PRINTER_OK               0
#define SERIAL_PRINTER_LIB_NOT_LOADED -701
#define SERIAL_PRINTER_FUNC_NOT_FOUND -702
#define SERIAL_PRINTER_INIT_FAIL      -703
#define SERIAL_PRINTER_UNINITIALIZED  -704
#define SERIAL_PRINTER_INVALID_PARAM  -705
#define SERIAL_PRINTER_INVALID_DEVICE -706
#define SERIAL_PRINTER_POWER_OFF      -707
#define SERIAL_PRINTER_CABLE_OUT      -708
#define SERIAL_PRINTER_OFFLINE        -709
#define SERIAL_PRINTER_ERROR          -710
#define SERIAL_PRINTER_BUSY           -711
#define SERIAL_PRINTER_PAPER_END      -712
#define SERIAL_PRINTER_CRC_ERROR      -713
#define SERIAL_PRINTER_NAK_RCVD       -713
//
// Serial Printer Status
//
#define SERIAL_PRINTER_STATUS_POWER_OFF 1
```

Implementação do Software Embarcado no Microcontrolador

```
#include <msp430.h>

/*
 * Arthur Morales Sampaio - Universidade de Brasilia
 *
 * main.c
 */

#define BYTES_QTY_TO_SEND 124

#define LOWBYTE(v)   ((unsigned char) (v) )
#define HIGHBYTE(v)  ((unsigned char) (((unsigned int) (v)) >> 8))
#define START_ADDRESS 0x200


void main() {

	WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer

	register unsigned int bitCount = 0;
	register unsigned int bytesCounter = 0;
	register unsigned char * pDataByte = ( unsigned char *) START_ADDRESS;


	P1OUT &= 0x00;
	P1DIR = 0x03; // P1.0 ==> DataTX - P1.1 ==> DataAvailable  - P1.2 <==
DataRead

	/*Piscar a luz para dizer que iniciou o programa*/
	for( bitCount = 0; bitCount < 10; bitCount++) {
		P1OUT ^= 0x01;
		_delay_cycles(65536);
	}

	bitCount = 0;
	bytesCounter = 0;

	// We have 8 bits (1B) to send.
	while( bytesCounter < BYTES_QTY_TO_SEND )
	{

		//if( bytesCounter <= BYTES_QTY_TO_SEND )
		//{

			if( bitCount < 8 )
			{
				if ( *pDataByte & ( 0x01 << bitCount ) ) // If the bit
we intend to send is 1...
				{
					P1OUT |= 0x01; //Send 1
				}
				else
				{
					P1OUT &= 0xFE; // Send 0
				}

				// After we sent data set DataAvailable pin
				P1OUT |= 0x02;

				while( !(P1IN & (0x01 << 2) ) );// Wait for DataRead

				P1OUT &= 0xFD; // DataRead received. Drop DataAvailable
```

```
                        while( (P1IN & (0x01 << 2)) );// Wait for DataRead DROP.

                        bitCount++; //Increment bitCount

                  }
                  else if (bitCount >= 8)
                  {
                        bitCount = 0;
                        pDataByte++;
                        bytesCounter++;
                  }

            //} else {
                  //Done sending memory dump
            //    break;
            //}
      }

      SD16CTL = SD16REFON + SD16SSEL_1;          // 1.2V ref, SMCLK
      SD16INCTL0 = SD16INCH_6;                   // A6+/-
      SD16CCTL0 = SD16SNGL + /*SD16IE + */SD16UNI;  // Single conv, interrupt,
Unipolar

      SD16CCTL0 |= SD16SC;                      // Start SD16 conversion

      while( !(SD16CCTL0 & 0x04) ); //Wait until there is temperature data in
SD16MEM0 - When SD16IFG is set continue.

      pDataByte = (unsigned char*)START_ADDRESS;

      //There's new data to be read in SD16MEM0
      *pDataByte = LOWBYTE(SD16MEM0);

      bitCount = 0;

      /*Start sending HIGH BYTE of Temperature! Seria melhor escrever uma funcao
que enviasse um byte pela porta serial
       * mas nao podemos utilizar STACK nesse programa, entao nao pode haver
nenhuma chamada a funçao alguma! */
      while(1) {
            if( bitCount < 8 )
            {
                  if ( *pDataByte & ( 0x01 << bitCount ) ) // If the bit we
intend to send is 1...
                  {
                        P1OUT |= 0x01; //Send 1
                  }
                  else
                  {
                        P1OUT &= 0xFE; // Send 0
                  }

                  // After we sent data set DataAvailable pin
                  P1OUT |= 0x02;

                  while( !(P1IN & (0x01 << 2) ) );// Wait for DataRead

                  P1OUT &= 0xFD; // DataRead received. Drop DataAvailable

                  while( (P1IN & (0x01 << 2)) );// Wait for DataRead DROP.
```

```
                    bitCount++; //Increment bitCount

            }
            else if (bitCount >= 8)
            {
                    break;
            }
      }
      /*End sending HIGH BYTE of temperature! */
      pDataByte++;
      *pDataByte = HIGHBYTE(SD16MEM0);

      bitCount = 0;

      /*Start sending LOW BYTE of Temperature! Seria melhor escrever uma funcao
que enviasse um byte pela porta serial
       * mas nao podemos utilizar STACK nesse programa, entao nao pode haver
nenhuma chamada a funçao alguma! */
      while(1) {
            if( bitCount < 8 )
            {
                    if ( *pDataByte & ( 0x01 << bitCount ) ) // If the bit we
intend to send is 1...
                    {
                            P1OUT |= 0x01; //Send 1
                    }
                    else
                    {
                            P1OUT &= 0xFE; // Send 0
                    }

                    // After we sent data set DataAvailable pin
                    P1OUT |= 0x02;

                    while( !(P1IN & (0x01 << 2) ) );// Wait for DataRead

                    P1OUT &= 0xFD; // DataRead received. Drop DataAvailable

                    while( (P1IN & (0x01 << 2)) );// Wait for DataRead DROP.

                    bitCount++; //Increment bitCount

            }
            else if (bitCount >= 8)
            {
                    break;
            }
      }
      /*End sending HIGH BYTE of temperature! */

      // Change input of SD16_A to read Vdd data from the Internal Voltage Divider
- Channel 5
      SD16INCTL0 = SD16INCH_5;                    // A5+/-

      SD16CCTL0 |= SD16SC;                       // Start SD16 conversion

      _delay_cycles(65536);

      while( !(SD16CCTL0 & 0x04) ); //Wait until there is Vcc data in SD16MEM0 -
When SD16IFG is set continue.

      //There's new data to be read in SD16MEM0
```

```
      pDataByte++;
      *pDataByte = LOWBYTE(SD16MEM0);

      bitCount = 0;

      /*Start sending HIGH BYTE of Temperature! Seria melhor escrever uma funcao
que enviasse um byte pela porta serial
       * mas nao podemos utilizar STACK nesse programa, entao nao pode haver
nenhuma chamada a função alguma! */
      while(1) {
            if( bitCount < 8 )
            {
                  if ( *pDataByte & ( 0x01 << bitCount ) ) // If the bit we
intend to send is 1...
                  {
                        P1OUT |= 0x01; //Send 1
                  }
                  else
                  {
                        P1OUT &= 0xFE; // Send 0
                  }

                  // After we sent data set DataAvailable pin
                  P1OUT |= 0x02;

                  while( !(P1IN & (0x01 << 2) ) );// Wait for DataRead

                  P1OUT &= 0xFD; // DataRead received. Drop DataAvailable

                  while( (P1IN & (0x01 << 2)) );// Wait for DataRead DROP.

                  bitCount++; //Increment bitCount

            }
            else if (bitCount >= 8)
            {
                  break;
            }
      }
      /*End sending HIGH BYTE of temperature! */
      pDataByte++;
      *pDataByte = HIGHBYTE(SD16MEM0);

      bitCount = 0;

      /*Start sending LOW BYTE of Temperature! Seria melhor escrever uma funcao
que enviasse um byte pela porta serial
       * mas nao podemos utilizar STACK nesse programa, entao nao pode haver
nenhuma chamada a função alguma! */
      while(1) {
            if( bitCount < 8 )
            {
                  if ( *pDataByte & ( 0x01 << bitCount ) ) // If the bit we
intend to send is 1...
                  {
                        P1OUT |= 0x01; //Send 1
                  }
                  else
                  {
                        P1OUT &= 0xFE; // Send 0
                  }
```

```
                    // After we sent data set DataAvailable pin
                    P1OUT |= 0x02;

                    while( !(P1IN & (0x01 << 2) ) );// Wait for DataRead

                    P1OUT &= 0xFD; // DataRead received. Drop DataAvailable

                    while( (P1IN & (0x01 << 2)) );// Wait for DataRead DROP.

                    bitCount++; //Increment bitCount

             }
             else if (bitCount >= 8)
             {
                    break;
             }
      }
      /*End sending HIGH BYTE of temperature! */
}
```