UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

ESCOLA DE ENGENHARIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

AQUILES TURCATI

METODOLOGIA DE AUTOTESTE EMBARCADO PARA CONVERSORES AD EM SISTEMAS PROGRAMÁVEIS DE SINAL MISTO

Porto Alegre

2014

AQUILES TURCATI

METODOLOGIA DE AUTOTESTE EMBARCADO PARA CONVERSORES AD EM SISTEMAS PROGRAMÁVEIS DE SINAL MISTO.

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para Graduação em Engenharia Elétrica.

Orientador: Prof. Dr. Tiago Roberto Balen

Porto Alegre
2014

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

ESCOLA DE ENGENHARIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

AQUILES TURCATI

METODOLOGIA DE AUTOTESTE EMBARCADO PARA CONVERSORES AD EM SISTEMAS PROGRAMÁVEIS DE SINAL MISTO.

Este trabalho foi julgado adequado para fazer jus aos créditos da Disciplina de "Projeto de Diplomação" do Departamento de Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____

Prof. Dr. Tiago Balen, UFRGS

Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Banca Examinadora:

Prof. Dr. Tiago Balen, UFRGS

Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Gilson Inácio Wirth, UFRGS

Doutor pela Universität Dortmund – Dortmund, Alemanha

Dr. André Borin Soares, UFRGS

Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Porto Alegre

2014

AGRADECIMENTOS

Agradeço aos meus pais, minha avó Leonor, meus amigos e demais familiares pelo apoio e incentivo deste o início desta jornada.

Agradeço ao Professor Tiago Balen pela orientação neste trabalho.

RESUMO

Este trabalho apresenta uma Metodologia de autoteste embarcado de conversores AD em sistemas programáveis de sinal misto, caracterizando o conversor AD de interesse quanto ao erro de offset, erro de ganho, não linearidade integral (INL), não linearidade diferencial (DNL) por meio do método do histograma. A metodologia aplicada consiste em gerar e aplicar um sinal de excitação, tipo rampa triangular com função densidade de probabilidade conhecida, no conversor AD sob teste. Armazenar o número de ocorrência de cada código de saída constituindo o histograma, estimar as características de desempenho e exportar os resultados para um computador.

O autoteste tem como base o sistema programável PSoC1 (Programmable System on Chip 1) presente no kit de desenvolvimento CY3214 da Cypress Semiconductor.

Palavras-chave: Autoteste embarcado, conversor AD, teste do histograma, sistema programável de sinal misto.

ABSTRACT

This paper presents a test strategy for Built-in Self Test (BIST) of an Analog to Digital converter embedded in a programmable mixed-signal system. The test, performed through the histogram method, is focused on static parameter of the AD converter, such as the offset error, gain error, integral non-linearity (INL) and differential non-linearity (DNL). The methodology is based on the application of a signal with known probability density function (like a triangular ramp) to the AD converter under test as input stimulus. The number of occurrences of each output code is stored to generate the histogram and to estimate the performance characteristics that are exported to an external computer.

The converter under test is embedded on the PSoC1 (Programmable System on Chip 1) present in CY3214 development kit from Cypress Semiconductor.

Keywords: System-on-chip, built-in self-test, test strategies, mixed-signal testing, histogram test.

SUMÁRIO

LISTA DE FIGURAS

1. INTRODUÇÃO

O Autoteste embarcado ou *Built-In-Self-Test* (BIST) de Conversores Analógicos-Digitais (ADC) em circuitos integrados de sinal misto que possuem Conversores Digitais-Analógicos (DAC) geralmente utiliza uma metodologia digital, onde a reconfiguração e conexão dos blocos aplicados no autoteste é obtida por meio de conexões internas, reduzindo o uso de conexões externas. Com o teste do ADC é possível determinar os parâmetros de desempenho do conversor. Parâmetros estáticos geralmente via método do histograma e parâmetros dinâmicos via a análise espectral da saída (FLORES, 2003) e (KOOK, 2011).

O método do histograma é uma técnica clássica de teste de conversores AD utilizada para determinar os parâmetros de desempenho estáticos, bem como o erro de offset, o erro de ganho e as não linearidades. Este método consiste em aplicar um determinado sinal na entrada no conversor AD em teste e armazenar o número de vezes que cada código binário da saída é gerado (KOOK, 2011). Uma vez que o sinal de entrada é conhecido, o resultado da conversão também será, assim ao comparar o número de vezes em que cada código aparece na saída do conversor com o número de vezes esperado que cada código apareça é possível determinar o erro de ganho, erro de offset e as não linearidades integral e diferencial (FLORES, 2003). Para obter resultados estatisticamente satisfatórios esta técnica exige um grande número de amostras do sinal entrada, resultando em um tempo de teste longo e exigência de hardware que dificultam a aplicação BIST.

1.1. OBJETIVO

Este trabalho tem como objetivo caracterizar um conversor analógico-digital embarcado em um sistema programável de sinal misto quanto às características estáticas de desempenho. Para tanto será utilizado o método do histograma onde será aplicado um sinal periódico do tipo rampa na entrada do conversor em teste.

As características de desempenho dinâmicas do conversor analógico-digital serão abordadas brevemente, pois não é o foco do trabalho.

1.2. ORGANIZAÇÃO DO TRABALHO

O trabalho foi dividido em capítulos que abordam os conceitos que foram utilizados para a execução, a implementação do sistema de autoteste embarcado e os

resultado bem como a avaliação dos resultados. Desta forma tem-se a seguinte organização:

- O Capítulo 2 introduz o sistema programável de sinal misto, em linhas gerais, e o sistema utilizado como base para o desenvolvimento deste trabalho;
- O Capítulo 3 descreve os conversores AD quanto aos parâmetros de desempenho estáticos e dinâmicos e apresenta o conversor AD a ser testado;
- O Capítulo 4 trata dos métodos de caracterização dos conversores AD com ênfase no método a ser aplicado neste trabalho, o método do histograma, que visa caracterizar o conversor quanto aos parâmetros estáticos de desempenho;
- O Capítulo 5 descreve a implementação do autoteste;
- O Capítulo 6 apresenta os resultados;
- O Capítulo 7 apresenta as conclusões oriundas da implementação do autoteste.

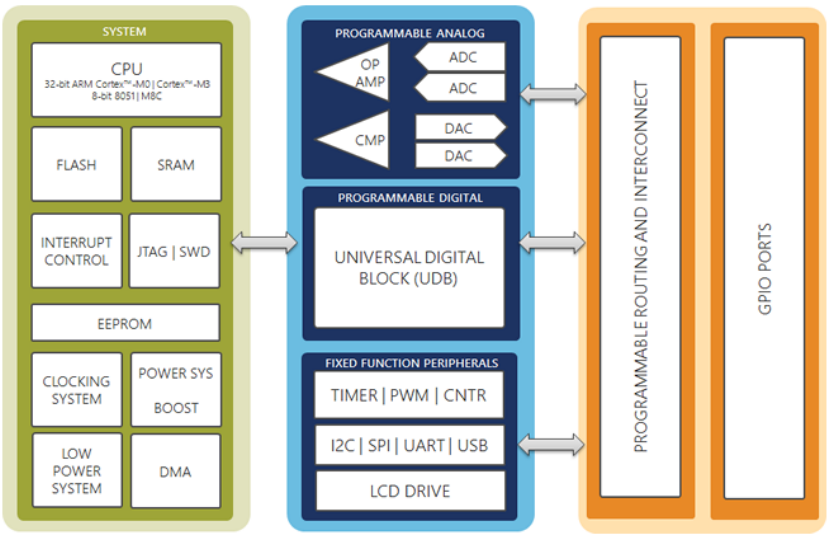
2. INTRODUÇÃO AO SISTEMA PROGRAMÁVEL DE SINAL MISTO

Um sistema programável de sinal misto, *Programmable System-on-Chip* (PSoC), possui blocos digitais programáveis e analógicos programáveis. Os blocos analógicos conforme configurados são conectados internamente reduzindo assim o número de periféricos.

Pode-se pensar o PSoC como um microcontrolador com mais graus de liberdade, onde o chip é customizável de acordo com a necessidade, pois os blocos analógicos e digitais são reconfiguráveis agregando uma flexibilidade ao sistema. A vantagem do uso deste tipo de sistema está associada ao curto ciclo de design de novos produtos, resultando em um menor custo de projeto e tempo de desenvolvimento.

A arquitetura de um PSoC de forma genérica é apresentada na Figura 1 onde percebe-se que o sistema pode ser dividido em três grandes blocos, um deles corresponde ao *System* que seria o microcontrolador em si, um bloco que compreende os blocos digitais e analógicos programáveis e o terceiro grupo as interconexões programáveis e as portas globais de entrada e saída de informação do chip.

Figura 1: Arquitetura genérica de um PSoC..



Fonte: CYPRESS WEB SITE, 2014.

Uma das características básicas da plataforma PSoC é a união dos blocos analógicos e digitais, sendo possível a configuração dos blocos utilizando uma biblioteca de funções disponibilizada pelo fabricante ou criar as próprias funções.

Dentre as famílias de PSoC da fabricante Cypress tem-se processadores de 8 bits como o M8C presente no PSoC1, o processador consagrado 8051 presente no PSoC3 e processadores de 32 bits como o Cortex M0 no PSoC4 e Cortex M3 no PSoC5LP (CYPRESS WEB SITE, 2014).

Para o desenvolvimento deste trabalho, tem-se disponível a plataforma de desenvolvimento CY3214, que trata-se de um PSoC1 com o chip CY8C2494, possuindo o processador proprietário da Cypress M8C. O M8C é um processador de 8 bits com arquitetura Harvard, com possibilidade de seleção de *clock* de 93,7kHz até 24MHz permitindo que este processador se enquadre em características de desempenho e consumo conforme requisitos da aplicação final como produto. Possui três espaços de memória, ROM, RAM e registradores acessíveis via barramento de dados. O endereço da ROM inclui a memória Flash onde é armazenado o programa. O microcontrolador possui 16kB de memória Flash, 1kB de memória SRAM (CYPRESS TRM, 2013).

3. CONVERSORES AD

3.1. Parâmetros de desempenho

Para o procedimento de teste de conversores AD torna-se conveniente dividir o processo em duas categorias condizentes com as características estáticas e dinâmicas do conversor: caracterização estática e caracterização dinâmica. A caracterização estática trata de especificar a não linearidade diferencial (differential non-linearity DNL), não linearidade integral (integral non-linearity INL), erro de offset e erro de ganho, sendo estes os principais erros estáticos. A Caracterização dinâmica trata de mensurar a relação sinal ruído (signal-to-noise ratio SNR), distorção harmônica total (total harmonic distortion THD), número efetivo de bits (effective number of bits ENOB), a relação sinal ruído e distorção (signal-to-noise and distortion ratio SiNAD), faixa dinâmica livre de componentes espúrios (spurious-free dynamic range SFDR), sendo estes os principais erros dinâmicos (AN 641, 2014) e (KOOK, 2011).

As características estáticas são inerentes ao próprio conversor AD relacionadas com imperfeiçoes no processo de fabricação. Tais características podem ser verificadas através do método do histograma, que tem por base a análise do espaçamento entre os códigos de saída do conversor. Os erros dinâmicos estão associados com a forma que o sinal analógico de entrada varia ao longo do tempo. Em geral, estes erros são representados pela distorção harmônica dos estágios analógicos, efeitos dinâmicos nos estágios de comparação e amplificação e as variações dependentes da frequência no espaçamento dos níveis de quantização.

Este trabalho, conforme mencionado anteriormente, tem por objetivo tratar de características estáticas do conversor AD. As características dinâmicas serão abordadas de forma sucinta na seção 3.1.2.

3.1.1. Parâmetros estáticos em conversores AD

As características estáticas de desempenho de um conversor AD são estimadas utilizando um sinal DC ou de baixa frequência como estimulo de entrada. Conhecer tais características é importante, pois elas representam de certa forma o quão preciso é o processo de conversão de um sinal analógico em um código digital (KOOK, 2011).

Aplicando um sinal de entrada do tipo rampa que excursione por toda a faixa de valores do conversor AD tem-se como resposta a curva característica que geralmente é utilizada para avaliar as características estáticas.

Para um conversor AD de N bits a curva característica teórica é dada por uma linha reta representando a função de transferência ideal, enquanto a curva característica real é formada por uma escala de 2^N níveis. A Figura 2 ilustra a função de transferência ideal e a curva característica para um conversor AD de 3 bits, como exemplo. Uma fração de valores de tensão da entrada analógica corresponde a um código digital do conversor AD, sendo esta fração definida como *code width*. O valor do sinal analógico no ponto de transição para o próximo código digital é definido como *code transition point*. Um *code width* de um conversor AD ideal é chamado de "bit menos significativo" (*least-significant-bit* ou LSB), que é a razão entre o fundo de escala da entrada analógica pelo número de códigos digitais (KOOK, 2011).

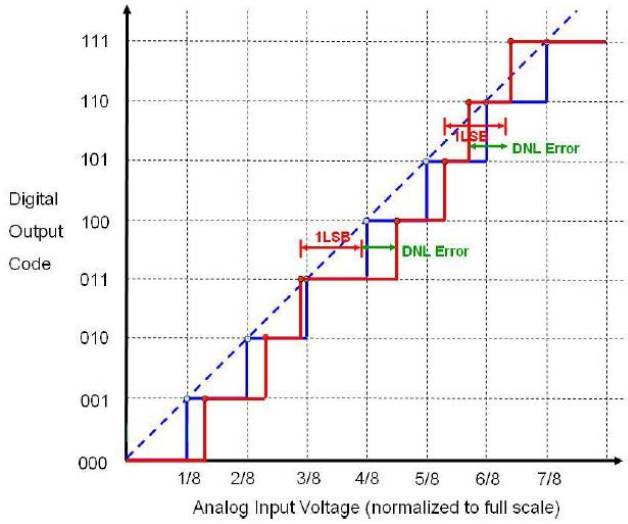
De um modo geral, os *code widths* não são todos iguais dentro da faixa de conversão do AD, devido a imperfeições inerentes ao processo de fabricação, fato este que resulta em uma função de transferência com erros de linearidade, por exemplo.

Figura 2: Função de transferência ideal e curva característica de um conversor AD de 3 bits.

Fonte: KOOK, 2011.

A não linearicade diferencial (DNL) de um código é a diferença entre o *code width* real e o *code width* ideal. Qualquer desvio no LSB é definido como erro de DNL, que pode apresentar um valor positivo ou negativo (KOOK, 2011). A Figura 3 ilustra a não linearidade diferencial de um conversor AD de 3 bits.

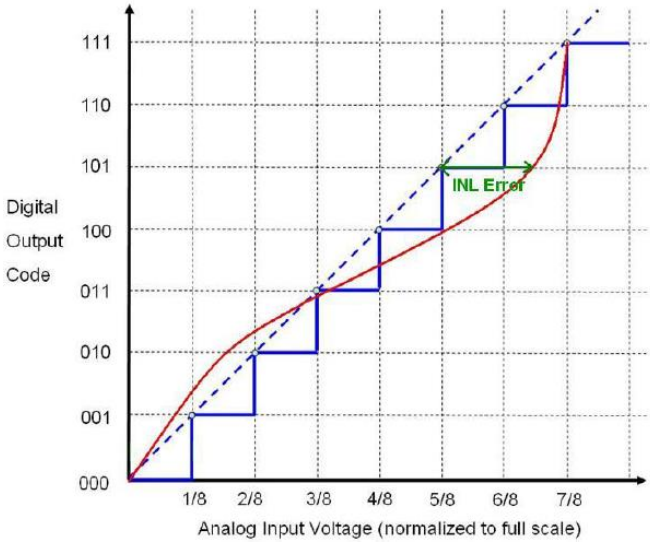
Figura 3: DNL em um conversor AD de 3 bits.



Fonte: KOOK, 2011.

A Figura 4 apresenta a não linearidade integral de um conversor AD, neste caso um conversor de 3 bits. O erro de INL de um código é o desvio da função de transferência real em relação à função de transferência ideal de um conversor AD.

Figura 4: INL de um conversor AD de 3 bits.

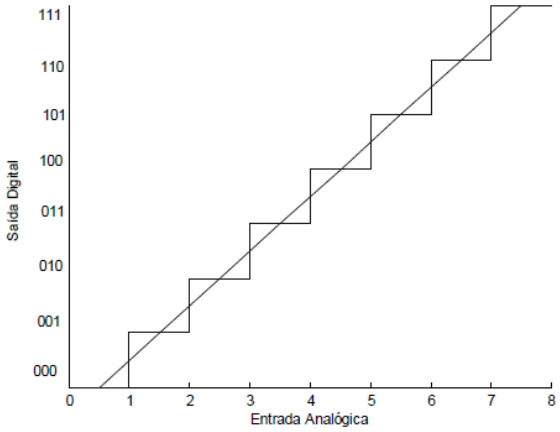


Fonte: KOOK, 2011.

O erro de *Offset* de um conversor AD pode ser descrito como o deslocamento horizontal da curva característica e pode ser medido pela diferença entre o valor real e o

valor teórico de entrada que fornece o LSB na saída (IEEE STD 1241, 2010). A Figura 5 ilustra o erro de *Offset* de um conversor de 3 bits.

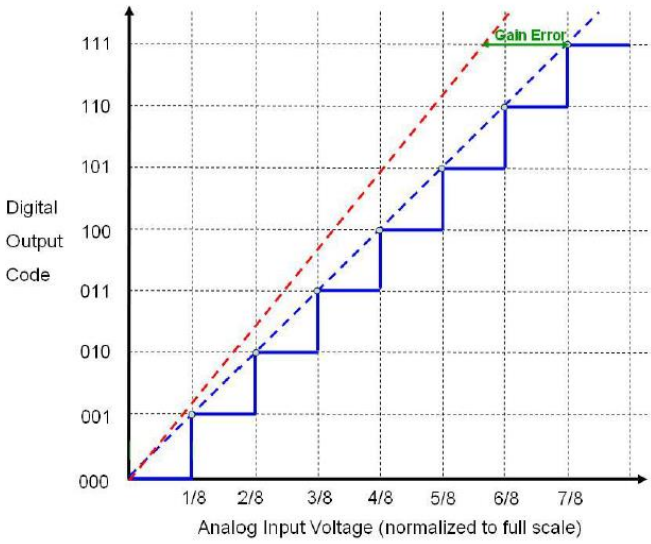
Figura 5: Erro de *Offset* de um conversor AD de 3 bits.



Fonte: FLORES, 2003.

O erro de ganho é o desvio do valor de fundo de escala real em relação ao valor de fundo de escala teórico para um determinado número de bits, ou seja, corresponde à rotação de toda a função de transferência em relação à origem, conforme ilustrado na Figura 6.

Figura 6: Erro de ganho de um conversor AD de 3 bits.



Fonte: KOOK, 2011.

Quantos aos erros de não linearidade, uma vez que são conhecidos, podem ser corrigidos de certa forma posteriormente por processamento digital.

Estimar os parâmetros estáticos de desempenho do conversor AD não significa conhecer por completo seu comportamento quando forem aplicados sinais que variem com o tempo, sinais AC. Fica por parte da caracterização dinâmica estimar o desempenho do conversor em relação a distorções harmônicas e ruído quando um sinal AC for aplicado.

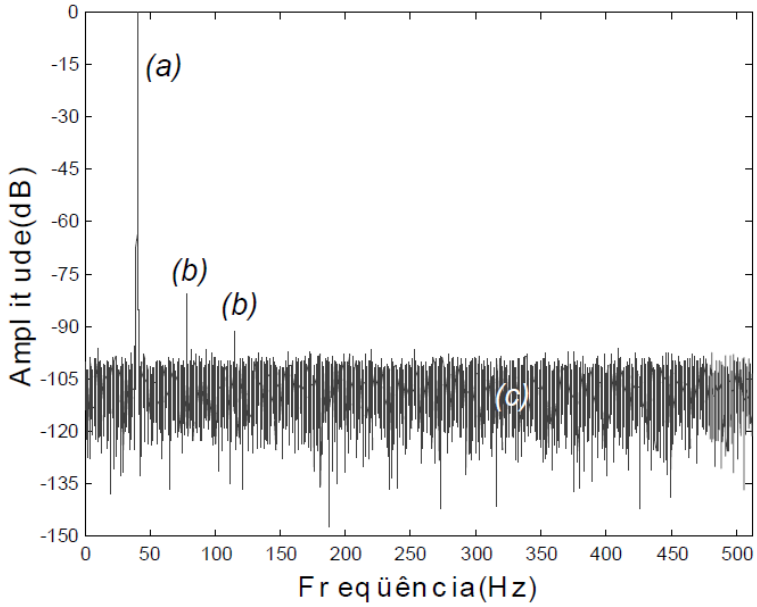
3.1.2. Parâmetros dinâmicos em conversores AD

Para aplicações onde o sinal de entrada do conversor varie lentamente a caracterização estática do conversor AD é satisfatória, todavia em aplicações em que o sinal de entrada apresente componentes de frequência elevada a caracterização estática do conversor não é suficiente, para tanto é necessário caracterizar os parâmetros de desempenho dinâmicos do conversor, tal caracterização é denominada como caracterização dinâmica.

Geralmente a caracterização dinâmica tem por base a aplicação de uma excitação senoidal de fase, amplitude e frequência conhecidas e os parâmetros são estimados baseados no espectro do sinal. A Figura 7 apresenta o espectro de frequência típico da saída de um conversor AD não ideal de 16 bits onde é possível visualizar a frequência fundamental (a), as componentes harmônicas (b) e o ruído de quantização do conversor (c).

Uma vez que o nível de ruído, a distorção harmônica e a potência do sinal forem medidos a partir do espectro de frequência obtido na saída do conversor AD, as características de desempenho dinâmicas podem ser estimadas.

Figura 7: Análise espectral da saída de um ADC: (a) frequência fundamental do sinal; (b) componentes harmônicas do sinal; (c) ruído de quantização do conversor.



Fonte: FLORES, 2003.

A Relação Sinal Ruído (*Signal Noise to Ratio – SNR)* de um conversor AD quando aplicado um sinal senoidal puro de entrada é definida como a razão entre o valor efetivo (rms) da amplitude do sinal de entrada e o valor efetivo do ruído de quantização inerente ao processo de conversão. Este parâmetro não inclui as componentes harmônicas que são utilizadas para determinar a distorção harmônica total (THD). Segundo (IEEE STD 1241, 2010) um bom estimador para a SNR de um conversor de N bits é dado pela Equação 1.

$$SNR_{dB} = 6{,}02 \cdot N + 1{,}76 \tag{1}$$

A relação sinal ruído e distorção (*Signal-to-Noise and Distortortion Ratio – SINAD)* é a razão entre o valor efetivo (rms) da amplitude do sinal de entrada e o valor efetivo do ruído total, composto pelo ruído de quantização inerente ao processo de conversão e levando em consideração a distorção harmônica e os efeitos da amostragem, de acordo com (IEEE STD 1241, 2010) a SINAD é definida pela Equação 2.

$$SINAD_{dB} = 20 \cdot \log_{10}\left[\frac{rms(sinal)}{rms(ruidoTotal)}\right] = 10 \cdot \log_{10}\left[\frac{P_{sinal}}{P_{ruidoTotal}}\right] \tag{2}$$

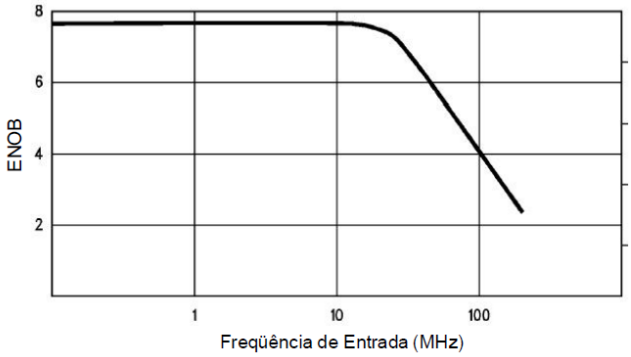
O Número Efetivo de Bits (*Effective Number of Bits – ENOB)* é o parâmetro de avaliação da resolução efetiva do conversor AD real. Este parâmetro relaciona o ruído total do conversor real com ruído de quantização de um conversor ideal. Segundo (IEEE STD 1241, 2010) a Equação 3 define o ENOB.

$$ENOB \cong N - \log_2\left[\frac{rms(ruidoTotal)}{rms(ruidoQuantização)}\right] \quad (3)$$

onde N é o número de bits do conversor real.

De acordo com (MELKONIAN, 1991) o ENOB decresce com o aumento da frequência do sinal de entrada, conforme apresentado na Figura 8.

Figura 8: Variação do ENOB em relação a frequência de entrada.



Fonte: MELKONIAN, 1991.

A Distorção Harmônica Total (*Total Harmonic Distortion – THD)* relaciona o valor rms do sinal de entrada com o somatório do valor rms das harmônicas presentes no sinal de saída do conversor AD. A Equação 4 define o THD de acordo com (IEEE STD 1241, 2010).

$$THD_{dB} = 20 \cdot \log_{10}\left(\frac{\sqrt{\sum_k H_k^2}}{S}\right) = 10 \cdot \log_{10}\left(\frac{Potência\ das\ Harmônicas}{Potência\ do\ sinal\ de\ entrada}\right) \quad (4)$$

onde H_k é o valor efetivo da k-ésima componente harmônica do sinal de entrada presente no sinal de saída, S é o valor efetivo do sinal de entrada.

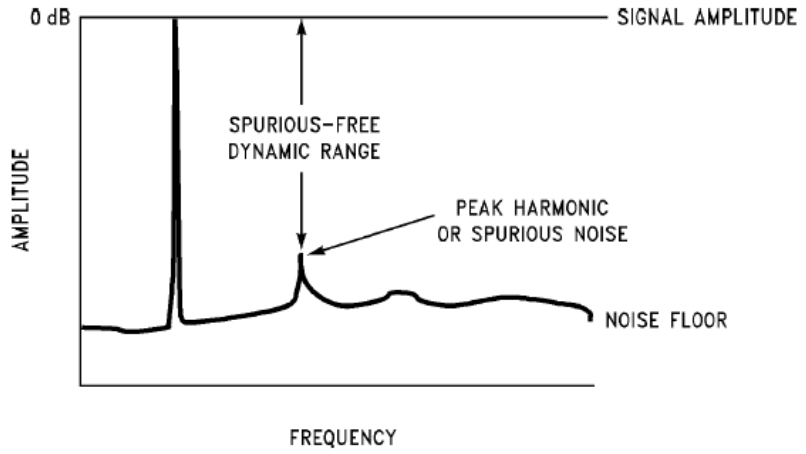
O *Spurious Free Dynamic Range (SFDR)* é a razão entre a amplitude do sinal de entrada pela amplitude da maior componente espectral do sinal de saída dentro do

intervalo de Nyquist, seja uma das componentes harmônicas ou um componente espúrio. A definição deste parâmetro segundo (IEEE STD 1241, 2010) é dada pela Equação 5.

$$SFDR_{dB} = 20 \cdot \log_{10}\left(\frac{A_{input}}{\max\{A_{H(\max)}\ ou\ A_{S(max)}\}}\right) \quad (5)$$

onde A_{input} é a amplitude da componente fundamental da senóide de entrada, $A_{H(max)}$ e $A_{S(max)}$ é a maior amplitude das componentes harmônicas e a maior amplitude das componentes espúrias, respectivamente. A Figura 9 ilustra a definição do SFDR.

Figura 9: SFDR indicando o nível que é possível distinguir o sinal de interesse do ruído e distorções.



Fonte: MELKONIAN, 1991.

Uma vez que alguns dos principais parâmetros de desempenho dinâmicos foram introduzidos, um detalhamento maior destas características é descrita em (IEEE STD 1241, 2010).

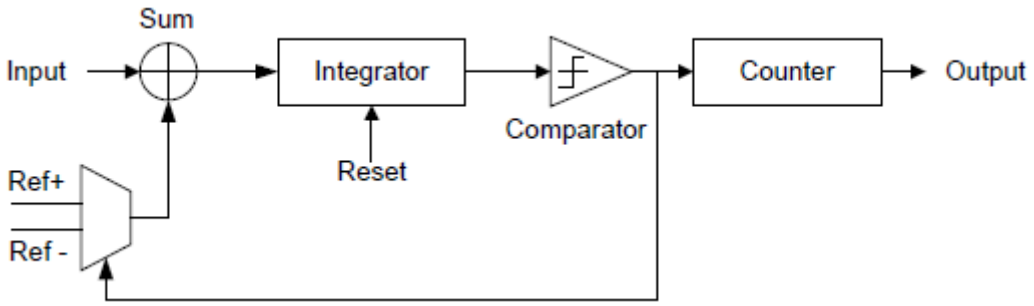
3.2. Conversor AD sob teste

O PSoC1 possui conversores AD com diferentes características de resolução, frequência de amostragem e relação sinal ruído. Dentre eles tem-se um conversor de aproximações sucessivas de 6 bits (SAR6), um conversor Delta Sigma e um conversor Incremental (ADCINC) com resolução de 6 a 14 bits.

O conversor a ser testado é o ADCINC apresenta arquitetura similar com um Delta Sigma ADC, mas operacionalmente é equivalente a um *Integrating* ADC (AN2096, 2014).

O ADCINC é um conversor do tipo *Averaging* que amostra o sinal de entrada muitas vezes para produzir um código de *n* bits de saída. Este conversor trabalha como um integrador que é resetado no início de cada conversão, a Figura 10 apresenta o diagrama simplificado do funcionamento deste conversor.

Figura 10: Circuito equivalente do conversor ADCINC.



Fonte: AN2096, 2014.

Assume-se que a entrada de sinal (*input*) é constante durante o período de conversão e positiva. Ao começo de cada conversão ocorre o *Reset* do integrador e do comparador, na sequência 2^n ciclos são executados, onde *n* é o número de bits do conversor. Sempre que a entrada do comparador excede o valor de referência a saída dele é colocada em nível lógico alto, o valor *Ref-* é somado ao sinal de entrada e o valor do contador é incrementado em uma unidade, inicia-se um novo ciclo, a entrada começa a ser integrada até atingir o valor de comparação do integrador. Ao final dos 2^n ciclos a valor presente no contador é o resultado da conversão AD (MÁRKUS *et. al.*, 2004) e (AN2096, 2014).

A frequência de amostragem (*SampleRate)* é dada de acordo com a Equação 6 (AN2096, 2014).

$$SampleRate = \frac{DataClock}{256 \cdot (2^{n-6}+1)} \quad (6)$$

Onde *n* é o número de bits, *DataClock* deve ser entre 125kHz e 8MHz e a frequência de amostragem deve ser limitada entre 125sps e 31.25ksps como garantia de funcionamento e também limitada em relação ao número de bits conforme a Figura 11.

Figura 11: Resolução versus Frequência máxima de amostragem do ADCINC.

Resolution	Maximum Sample Rate
6-bit	15.6 ksps
7-bit	10.4 ksps
8-bit	6.25 ksps
9-bit	3.4 ksps
10-bit	1.8 ksps
11-bit	976 sps
12-bit	480 sps
13-bit	242 sps
14-bit	121 sps

Este conversor quando operando em 5 volts, 8 bits, DataClock de 8MHz e com modulador de primeira ordem apresenta DNL de 0,6LSB e INL de 0,7LSB e *offset* de 5mV, tipicamente.

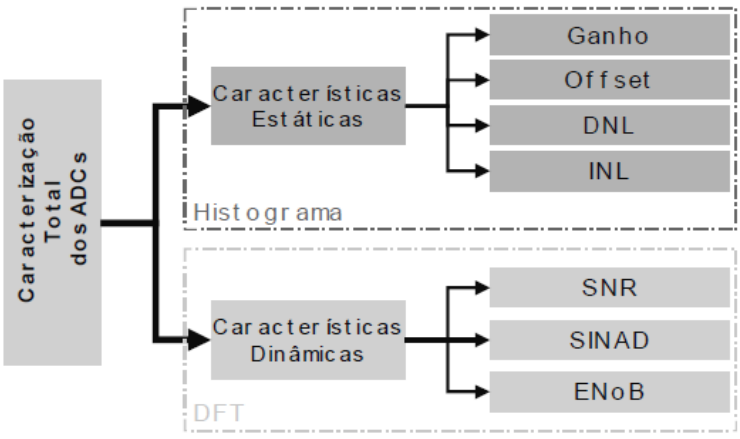
4. MÉTODOS DE CARACTERIZAÇÃO DOS CONVERSORES AD

Os métodos de caracterização de conversores analógicos digitais podem ser distinguidos basicamente em duas linhas de processo, a primeira delas envolve o teste do circuito, neste caso o conversor AD, de forma isolada por meio de equipamentos dedicados e projetados para esta tarefa onde o circuito alvo do teste recebe alterações visando uma melhor testabilidade. (*Design For Testability – DFT)* (FLORES, 2003). Neste arranjo é necessário ter acesso externo a determinados pontos do circuito. Na outra linha de abordagem o concersor AD é testado pelo próprio sistema em que ele faz parte (*Built-in Self Test – BIST),* ou seja, os circuitos de geração da excitação, controle e avaliação da resposta fazem parte do mesmo sistema que o circuito sob teste (FLORES, 2003).

Com o intuito de obter a caracterização dos conversores analógico-digitais quanto às características de desempenho estáticas e dinâmicas utilizando o conceito BIST tem-se como base o uso de dois métodos distintos: O método do histograma baseado no *Code Transition Density (CTD)* para características estáticas e o método da análise espectral onde é geralmete aplicado a transformada rápida de Fourier (*FFT)* para analisar o espectro de frequência do sinal de saída para determinar as características de desempenho dinâmicas.

A Figura 12 ilustra o os métodos de caracterização do conversor bem como as características mensuradas via cada método.

Figura 12: Métodos de caracterização de conversores AD.



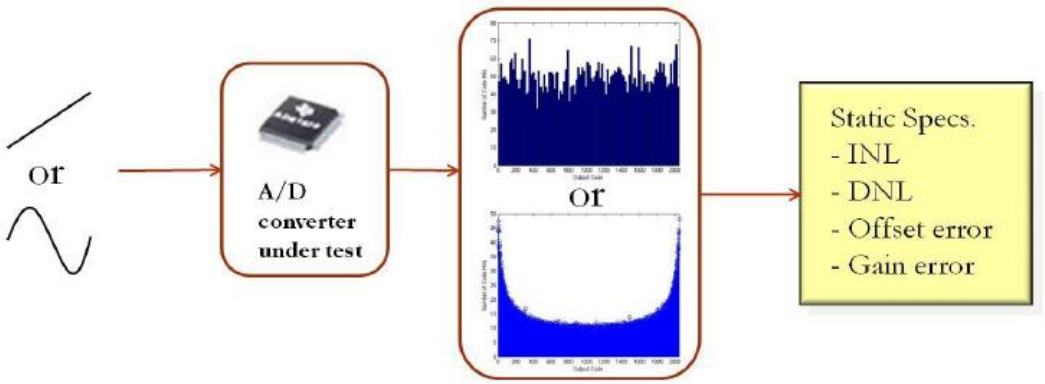
Fonte: FLORES, 2003.

O Método do histograma será abordado de forma mais detalhada na seção seguinte.

4.1. Método do Histograma

Segundo (DOERNBERG, 1984) com o método do histograma é possível determinar com uma ótima acuidade as características estáticas de desempenho do conversor AD. Este método envolve a aplicação de um sinal com distribuição de probabilidade conhecida e o armazenamento no número de ocorrência de cada código na saída do conversor, ou seja, aplicando um sinal com densidade de probabilidade conhecida a saída seguirá esta função de densidade de probabilidade. Se ocorrer alguma alteração na função de densidade de probabilidade de saída pode-se afirmar que o conversor apresenta falha (DOERNBERG, 1984) e estas variações de códigos estão relacionadas com as características estáticas do conversor. A Figura 13 ilustra o sinal de excitação que pode ser uma senóide ou um sinal triangular e os respectivos histogramas bem como as características estáticas de desempenho mensuradas com este método.

Figura 13: Método do histograma, sinais de excitação e características estáticas.



Fonte: KOOK, 2011.

Para obter resultados estatisticamente satisfatórios o método do histograma exige muitas amostras, seja com amostragem determinística como apresentado em (LIBERALI *et. al.*, 1996) ou com amostragem aleatória conforme (DOERNBERG, 1984) para garantir que um número mínimo de amostras por código binário seja obtido. Pode-se dizer que para um conversor de 8 bits seriam necessário 1 milhão de amostras aleatórias ou 64 mil amostras determinísticas para atingir resultados estatísticos satisfatórios. Estas amostras adquiridas são então comparadas com resultados teóricos provenientes da função de densidade de probabilidade conhecida do sinal de entrada. Se

o sinal de entrada for uma senóide será exigido maior potencial de cálculo e memória para estimar as características quando comparado com a situação em que o sinal de entrada for um sinal triangular. Na sequência este tema será abordado com mais detalhe.

Muitos fatores devem ser levados em consideração no momento de aplicar o método do histograma, como mencionado anteriormente, o número de amostras necessárias é um fator muito importante para estimar com precisão o DNL e INL.

Consideramos, como exemplo, o método de teste BIST onde o próprio circuito sob teste gera o sinal de excitação, assumindo que o conversor sob teste seja de 12 bits e com 1Msps e que o interesse seja obter 20 amostras por código de saída, desta forma é possível ter uma resolução de 0,05LSB. Isto implica que a rampa de excitação seja capaz de cobrir os 4096 níveis de leitura do sinal de entrada. Assim um total de 81920 amostras seriam necessárias, equivalendo a 20 amostras para cada um dos 4096 códigos. Dando sequencia no exemplo, assume-se que a rampa seja gerada por um conversor DA de 16 bits, isto significa que o DAC divide cada um dos códigos de 12 bits em 16 níveis, resultando em 0,06LSB. A incerteza na medida do DNL para este exemplo que foi exposto por (KESTER, 2005) é dada pela soma de 0,05LSB com 0,06LSB resultando em 0,11LSB.

Gerar o sinal de excitação utilizando um DAC é impraticável para testes de conversores AD com resolução superior a 12bits, abrindo espaço para utilizar um gerador de sinal externo com um sinal de excitação de frequência não sub harmônica da frequência de amostragem do conversor sob teste. Desta forma a obtenção do número total de amostras necessárias torna-se um problema estatístico (KESTER, 2005).

Por fim, outro fator impactante no processo de implementação do método do histograma refere-se ao *buffer* de memória necessário para lidar com certa velocidade os dados referentes à saída do conversor AD, bem como espaço de memória para trabalhar com variáveis relacionadas com o alto número de amostra que se faz necessário dependendo da metodologia de teste.

4.1.1. Número de amostras necessárias.

Utilizando o método convencional de amostragem estocástica, regido por fundamentos estatísticos, a frequência de amostragem e o sinal de entrada não devem ser correlacionados e a precisão deste método depende do número de amostras adquiridas. Por exemplo, um conversor AD de n bits com β LSB de precisão para

estimar o erro de DNL com um nível de confiança de $(1-\alpha)\cdot 100\%$ o número mínimo de amostras é dado pela Equação 7 (LIBERALI *et. al.*, 1996).

$$N_s = \frac{Z^2_{\alpha/2}\cdot\pi\cdot 2^{n-1}}{\beta^2} \quad (7)$$

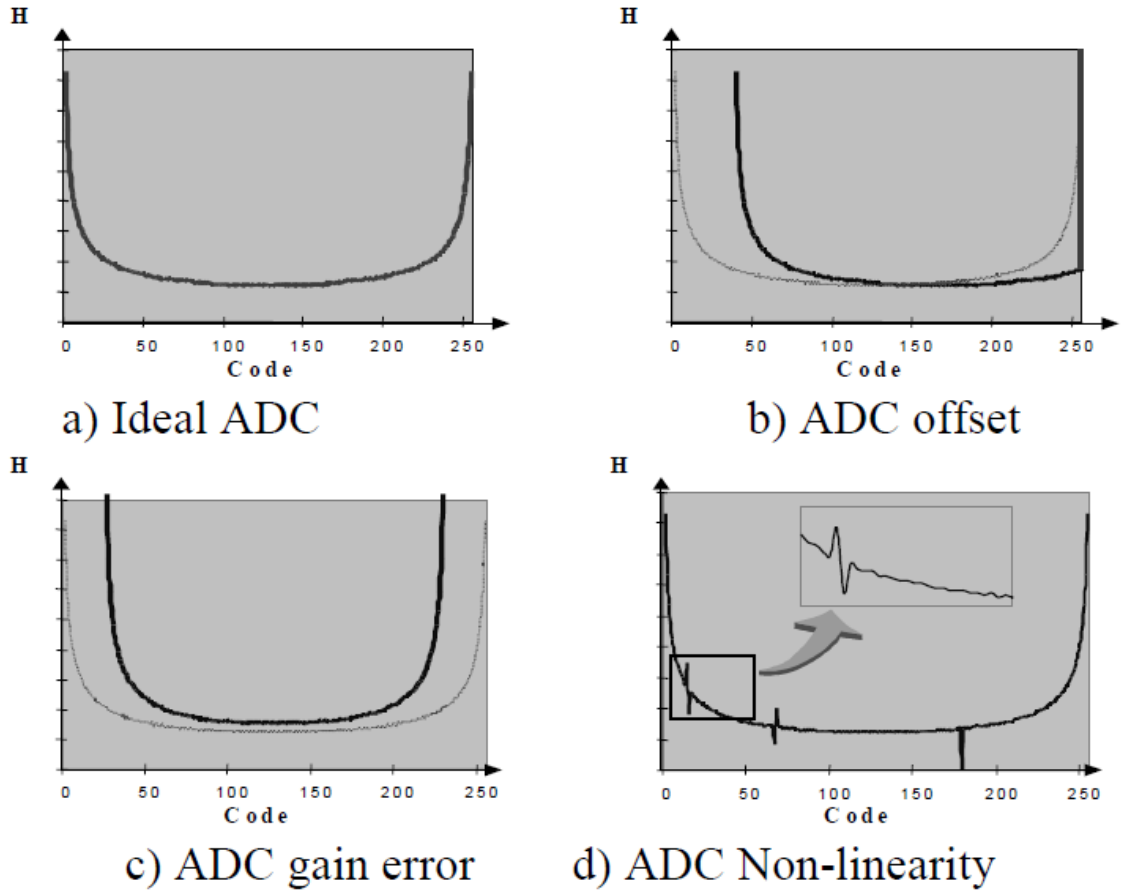
Onde $Z\alpha/2$ é o valor da variável da distribuição normal de probabilidades com o nível de confiança de $(1-\alpha)$.

Utilizando o método de amostragem determinístico a amostragem deve ser sincronizada com o sinal de entrada, por exemplo, utilizando um PLL (phase-locked loop), possibilitando que as amostras sejam adquiridas de forma uniforme dentro do período do sinal de excitação. O número de amostras necessárias para um conversor de *n* bits pode ser obtido por meio da função de probabilidade da amplitude do sinal de entrada, que para uma rampa é constante (LIBERALI *et. al.*, 1996). Por exemplo, para um conversor AD com frequência de amostragem de 100kHz, com 10mV de LSB, para obter-se uma resolução de 0,01LSB é necessário ter 100 amostras por código binário. Sendo assim a duração da rampa por código é de 1ms (100x1/100kHz) resultando na inclinação da rampa de excitação de 10mV/ms.

4.1.2. Cálculo das características de desempenho

Uma vez que o sinal de entrada pode ser qualquer desde que seja conhecida a função de densidade de probabilidade, a Figura 14a ilustra o histograma ideal obtido quando aplicado uma excitação de característica senoidal no conversor AD. Sabe-se que a caracterização é obtida comparando o histograma ideal com o real sendo assim possível perceber que a perda de amostras em determinados códigos refletem os erros estáticos de ganho, offset e não linearidade. A Figura 14b ilustra o efeito do erro de offset onde o histograma sofre um deslocamento lateral; a Figura 14c ilustra o erro de ganho e a Figura 14d ilustra o efeito do DNL no histograma.

Figura 14: Comportamento do histograma baseado em uma excitação senoidal quanto aos erros estáticos.



Fonte: RENOVELL *et. al.*, 2000.

4.1.2.1. Erro de Offset

O primeiro ponto de transição do código de saída em um conversor ideal ocorre na tensão de 0,5LSB. O erro de *offset* é o desvio deste ponto de transição em relação ao ponto de transição ideal. Por meio do teste do histograma o erro de *offset* é calculado conforme a Equação 8 segundo (NOVAK *et. al.*, 2012):

$$Offset = \frac{H(2^n - 1) - H(0)}{2 \cdot H_{ideal}} \quad (8)$$

Onde H_{ideal} representa o número de ocorrência ideal de cada código.

4.1.2.2. Erro de Ganho

Para um conversor AD ideal o ganho é unitário e a contagem de códigos não extremos é igual a H_{ideal}. A razão entre a contagem de qualquer valor não extremo e o respectivo H_{ideal} determina o valor do ganho para este código. Se o valor da contagem de um determinado código for menor que o valor ideal o ganho neste ponto é maior do

que o valor unitário, por exemplo. O ganho do conversor AD é determinado pela média do ganho de um determinado número *m* de códigos, não necessariamente de todos os códigos, conforme a Equação 9 (NOVAK *et. al.*, 2012).

$$Ganho^{-1} = \frac{\sum_{i=N1}^{N2} H(i)}{mH_{ideal}} \quad (9)$$

4.1.2.3. Não Linearidades

A não linearidade diferencial DNL é definida conforme a Equação 10 (NOVAK *et. al.*, 2012).

$$DNL(i) = \frac{H(i) - H(i)_{ideal}}{H(i)_{ideal}} \quad (10)$$

Onde $H(i)$ é o valor do histograma para um determinado código *i* e $H(i)_{ideal}$ é o respectivo valor ideal.

A não linearidade integral INL de um determinado código *i* é calculada por meio da soma cumulativa do DNL dos códigos anteriores, de acordo com a Equação 11 (NOVAK *et. al.*, 2012).

$$INL(i) = \sum_{j=1}^{i} DNL(j) \quad (11)$$

As equações acima consistem em somas, subtrações e divisões. Para simplificar a divisão seria interessante que o valor de $H(i)_{ideal}$ e *m* fossem potencias de 2.

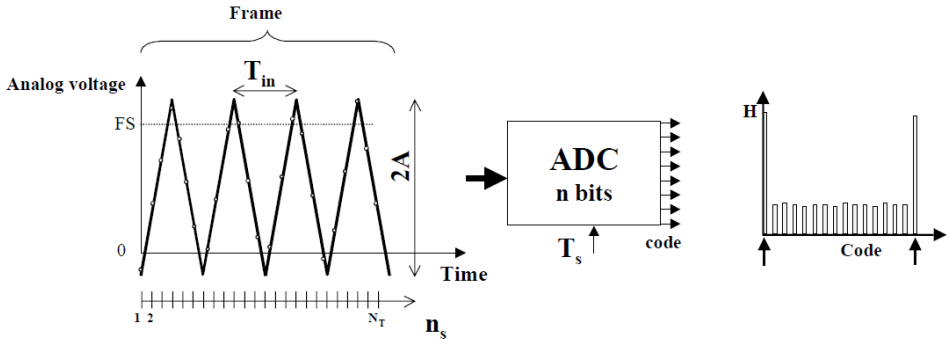
4.1.3. Sinal de excitação e hardware necessário.

A aplicação de uma metodologia BIST utilizando um sinal de excitação senoidal implicaria em 2^n registradores para armazenar o histograma ideal, 2^n registradores para armazenar o histograma experimental, sendo n o número de bits, e um núcleo DSP (*Digital Signal Processor*) para o processamento dos cálculos, tornando assim restrita a aplicação quanto ao hardware disponível.

Uma forma mais básica de aplicar o método do histograma é excitar o conversor com um sinal triangular periódico com mesmo tempo de subida e descida tendo como consequência uma função de densidade de probabilidade constante, fato este verdadeiro quando o sinal triangular apresenta amplitude ligeiramente maior que a máxima

amplitude de conversão de forma que toda a faixa de conversão é abrangida (RENOVELL *et. al.*, 2000). A Figura 15 ilustra o histograma baseado em uma excitação triangular, demonstrando que quantidade de códigos intermediários é constante, reduzindo assim a quantidade de memória para armazenar o histograma ideal e diminuindo a complexidade do processamento necessário para estimar os erros.

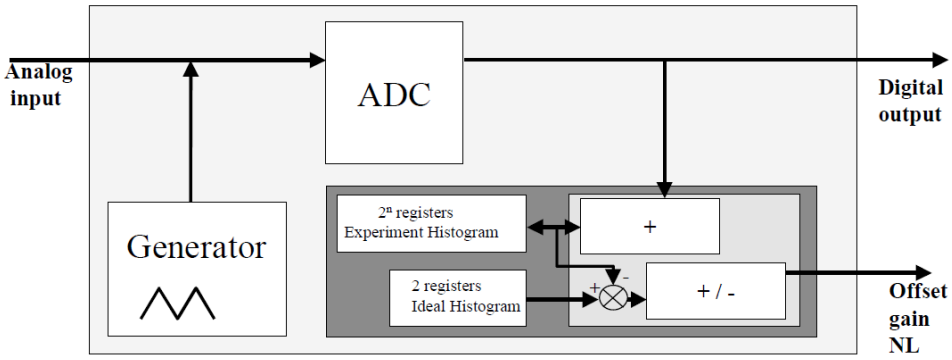
Figura 15: Histograma baseado em uma excitação triangular.



Fonte: RENOVELL *et. al.*, 2000.

Neste caso seriam necessários dois registradores para armazenar o histograma ideal e 2^n registradores para armazenar o histograma experimental onde n é o número de bits do conversor, conforme a Figura 16.

Figura 16: Estrutura BIST genérica considerando excitação triangular.



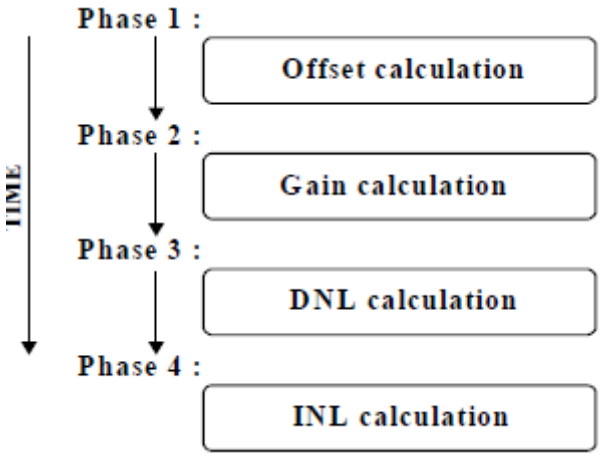
Fonte: RENOVELL *et. al.*, 2000.

Vale ressaltar que ainda assim é necessário armazenar uma grande quantidade de informação proveniente do histograma experimental, tornando complicada aplicação BIST.

Como alternativa, (RENOVELL *et. al.*, 2000) propõem um a Técnica de Decomposição do Tempo que divide o autoteste em diferentes fases que estimam

características de desempenho diferentes, conforme ilustra a Figura 17, diminuindo assim o processamento paralelo de informação.

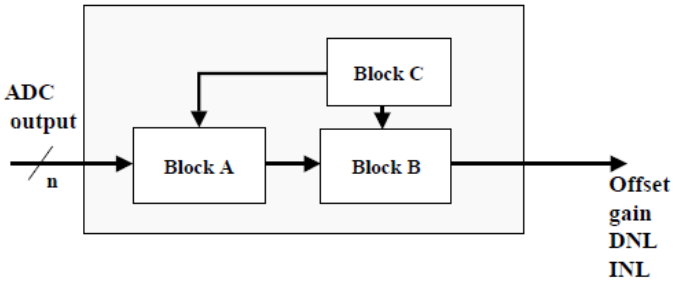
Figura 17: Fases da Técnica de Decomposição no Tempo.



Fonte: RENOVELL *et. al.*, 2000.

Segundo o método proposto por (RENOVELL *et. al.*, 2000) o *hardware* necessário para cálculo das características estáticas de desempenho pode ser dividido basicamente em três blocos conforme Figura 18, onde o Bloco A é um comparador, Bloco B um registrador e o Bloco C uma máquina de estados para gerenciar as diferentes fases do cálculo.

Figura 18: Hardware necessário aplicando método de divisão temporal do histograma.



Fonte: RENOVELL *et. al.*, 2000.

O pseudo código demonstrado a seguir, segundo (RENOVELL *et. al.*, 2000), faz referência à lógica de cálculo da não linearidade diferencial DNL.

```
for i = 0 to 2^n -1
    R1 = 0
      for n_s = 1 to N_T
      if code = i then  R = R+1
    H(i) = R
```

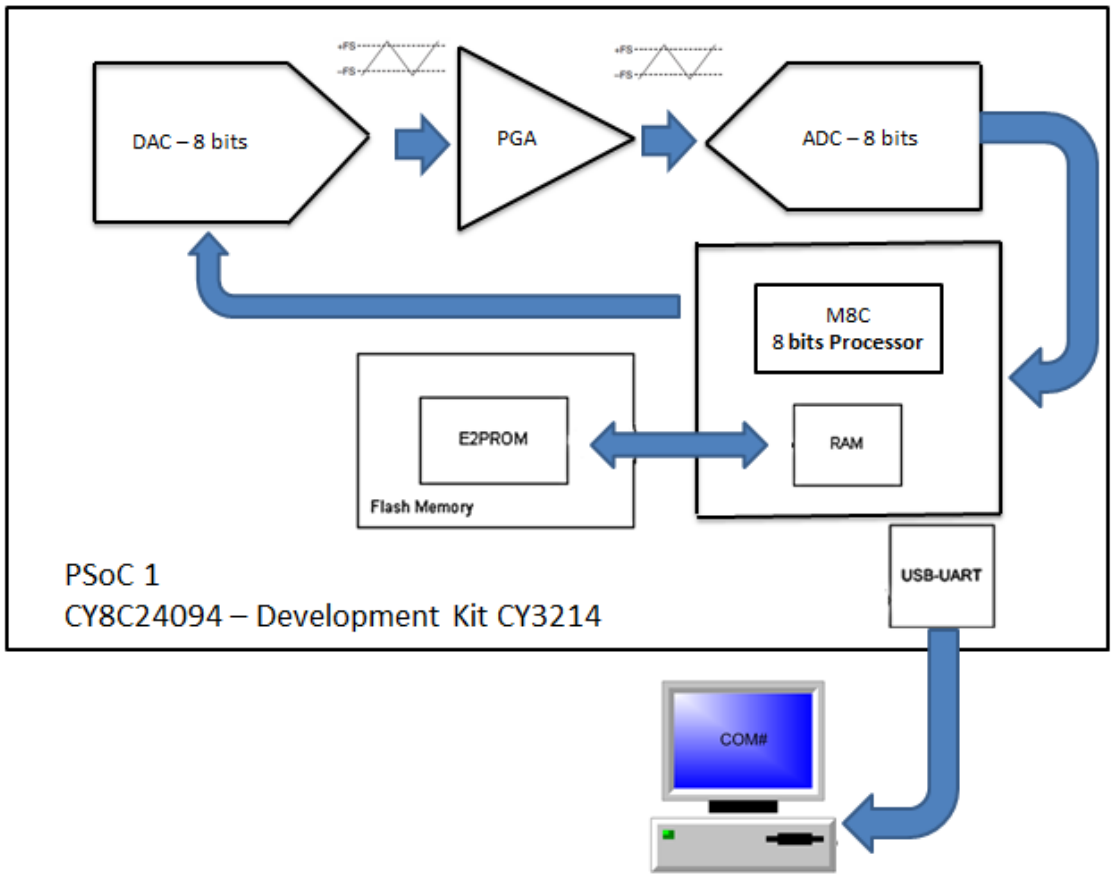
Onde para i variando de 0 até $2^n - 1$, se o código de saída do conversor for igual ao valor i então o registrador R é incrementado resultando ao final das N_T iterações referentes às N_T amostras necessárias o valor de $H(i)$, assim o DNL pode ser calculada para este código conforme a Equação 10 apresentada anteriormente. Note que seriam necessárias novas N_T amostras para cada um dos códigos seguintes.

5. MÉTODO DE TESTE UTILIZADO

O método de teste do histograma a ser executado consiste em gerar internamente um sinal triangular e aplicar na entrada do conversor AD sob teste. Foi utilizado um conversor DA de oito bits para gerar o sinal de excitação, sinal este que na sequência passa por um amplificador de ganho programável (PGA) com ganho unitário e então direcionado à entrada do conversor AD de oito bits do tipo incremental (ADCINC) sob teste. O uso do PGA foi necessário por motivo de ligação interna dos blocos analógicos do PSoC utilizado. O conversor sob teste foi descrito na seção 3.2.

A Figura 19 apresenta o diagrama de blocos do método de teste utilizado, neste caso o BIST (*Built-in self test*), note que uma vez que o histograma é processado os resultados são enviados via USBUART para o computador e assim apresentá-los. O USBUART utiliza a interface USB para emular uma COM port.

Figura 19: Diagrama de blocos do método de teste BIST.

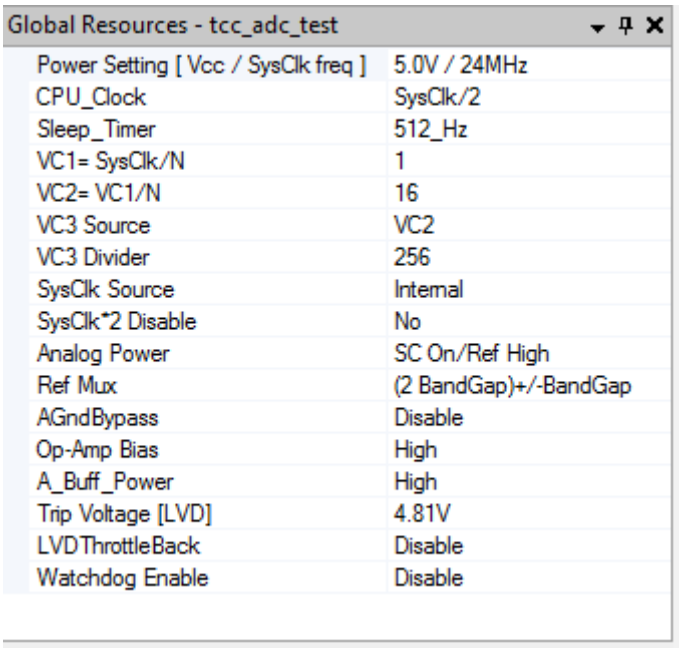


Para a programação do PSoC 1, a fabricante Cypress disponibiliza gratuitamente o software PSoC Designer.

Para início de utilização e programação é necessário configurar as características globais do chip (*Global Resources*) que afetam diretamente o funcionamento dos blocos. A maioria destas configurações são empregadas durante o *power-up* e não são alteradas posteriormente, mas elas podem ser alteradas posteriormente, dependendo da aplicação, acessando determinados registradores.

A Figura 20 apresenta a janela de configuração dos parâmetros globais do PSoC, têm-se as configurações de *clock* do sistema, alimentação, definição de alguns níveis de operação e tensões de referência.

Figura 20: Configurações Globais do PSoC1.



Dentre os parâmetros apresentados na Figura 20, o que deve-se deter mais atenção é o *Ref Mux*, que provavelmente é o mais importante.

O parâmetro *Op-Amp Bias* configurado como *High* faz com que os amplificadores operacionais operem com mais corrente e também amplia a largura de banda e aumenta a frequência de chaveamento dos capacitores chaveados.

Voltando para o parâmetro *Ref Mux*, ele determina a faixa e a precisão de cada componente que utiliza blocos de capacitores chaveados bem como os ADC's e DAC's. Para a medição de tensões é necessário o uso de referências constantes, como o *ground* ou outra tensão de referência, que impactam na precisão e desempenho do sistema. O PSoC oferece algumas possibilidades de referência. Mais detalhes sobre que referência

escolher e do porquê usar referência interna diferente do *Ground* são abordadas em (AN2219, 2014) e (AN74170, 2014) disponibilizados pela fabricante do PSoC1.

O PSoC trabalha apenas com sinais positivos, mas algumas aplicações exigem leituras de sinais com variações positivas e negativas em reação a determinada referência, para tanto o conceito de um *Artificial Ground* (terra virtual) é empregado no chip, possibilitando assim a leitura de sinais de ambas polaridades, esta referência é denominada *Analog Ground.*

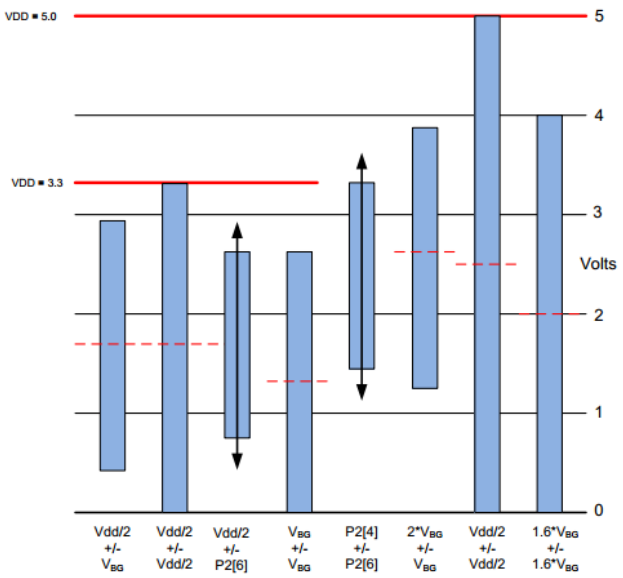
A Tabela 1 apresenta as opções de configuração do parâmetro Ref Mux bem como o *Analog Ground* oriundo desta escolha. Esta escolha afeta a faixa de leitura do ADC e a faixa de operação do DAC, este impacto é apresentado na Figura 21.

Tabela 1: Possibilidades de escolha do parâmetro global do Ref Mux.

Ref Mux Option	V_{DD}	RefLo	AGND	RefHi
(V_{DD}/2)+/- BandGap	3.3 V	0.350 V	1.65 V	2.95 V
	5.0 V	1.2 V	2.5 V	3.7 V
(V_{DD}/2)+/-(V_{DD}/2)	3.3 V	0.0 V (V_{SS})	1.65 V	3.3 V (V_{DD})
	5.0 V	0.0 V (V_{SS})	2.5 V	5.0 V (V_{DD})
BandGap +/- BandGap	3.0 V to 5.0 V	0.0 V (V_{SS})	1.30 V	2.60 V
(1.6*BandGap)+/-(1.6*BandGap)	> 4.16 V	0.0 V (V_{SS})	2.08 V	4.16 V
(2*BandGap) +/- BandGap	> 3.9 V	1.3 V	2.6 V	3.9 V
(2*BandGap) +/- P2[6]	3.0 V to 5.0 V	2.6V – P2[6]	2.6 V	2.6 V + P2[6]
P2[4] +/- BandGap	3.0 V to 5.0 V	P2[4] – 1.3 V	P2[4]	P2[4] + 1.3 V
P2[4] +/- P2[6]	3.0 V to 5.0 V	P2[4] – P2[6]	P2[4]	P2[4] + P2[6]

Fonte: AN74170, 2014.

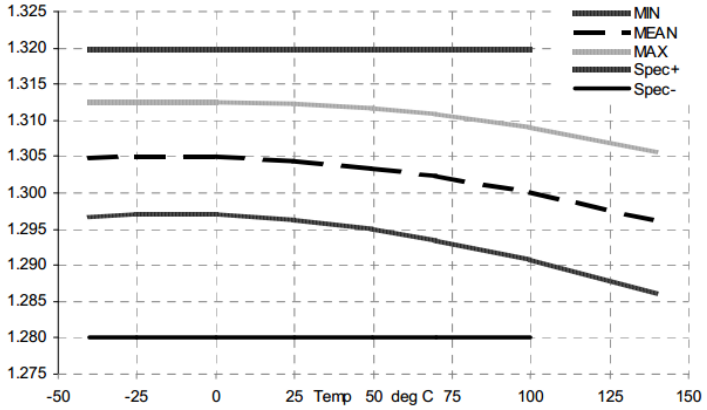
Figura 21: Referência de tensão do DAC e faixa de leitura do ADC.



Fonte: AN74170, 2014.

Para esta aplicação foi escolhido o Ref Mux relacionado com a tensão *Band Gap*, conforme pode ser observado na Figura 20 apresentada anteriormente. A variação desta tensão de referência em função da variação da temperatura de operação pode ser observada na Figura 22, onde é possível observar certa estabilidade térmica.

Figura 22: Variação da tensão de referência em função da variação de temperatura.



Fonte: AN2219, 2014.

Esta faixa de leitura do ADC, via escolha do Ref Mux, foi escolhida pois a maioria das aplicações que envolvem leitura de sensores, como termistores, apresentam certo *offset*.

Após efetuar as configurações globais, foram inseridos os blocos a serem utilizados neste projeto, como os conversores AD e DA, o amplificador de ganho programável PGA, os módulos E2PROM para manipulação e armazenamento dos dados gerados e o módulo USBUART para envio dos resultados via USB. Tais blocos são apresentados na janela *Workspace Explorer* do PSoC Designer conforme a Figura 23. Para este projeto o bloco referente ao display LCD foi inserido somente para Debug, sendo desnecessário para a metodologia de teste.

A Figura 24 mostra o ambiente de trabalho do software de programação do PSoC1, o PSoC Designer, nesta figura é possível observar as possibilidades de posicionamento e conexões dos blocos digitais, a possibilidade de posicionamento e conexões dos blocos analógicos, bem como as portas de entrada e saída do chip. Alguns blocos analógicos possuem posicionamento pré determinado, devido ao hardware dispendido por ele.

Figura 23: Blocos do PSoC utilizados.

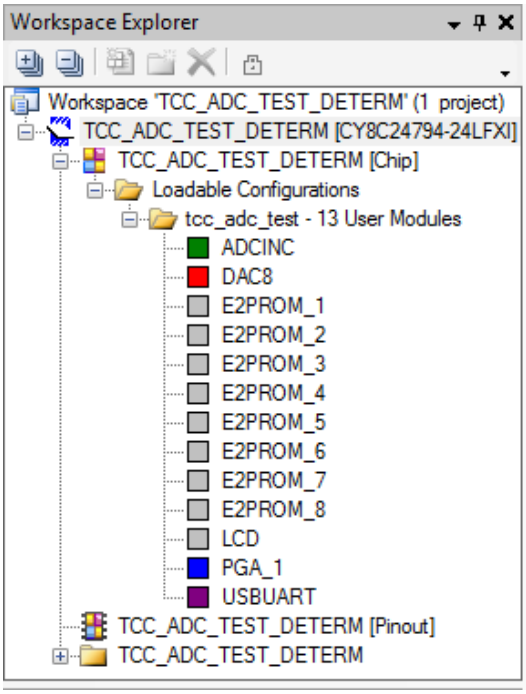
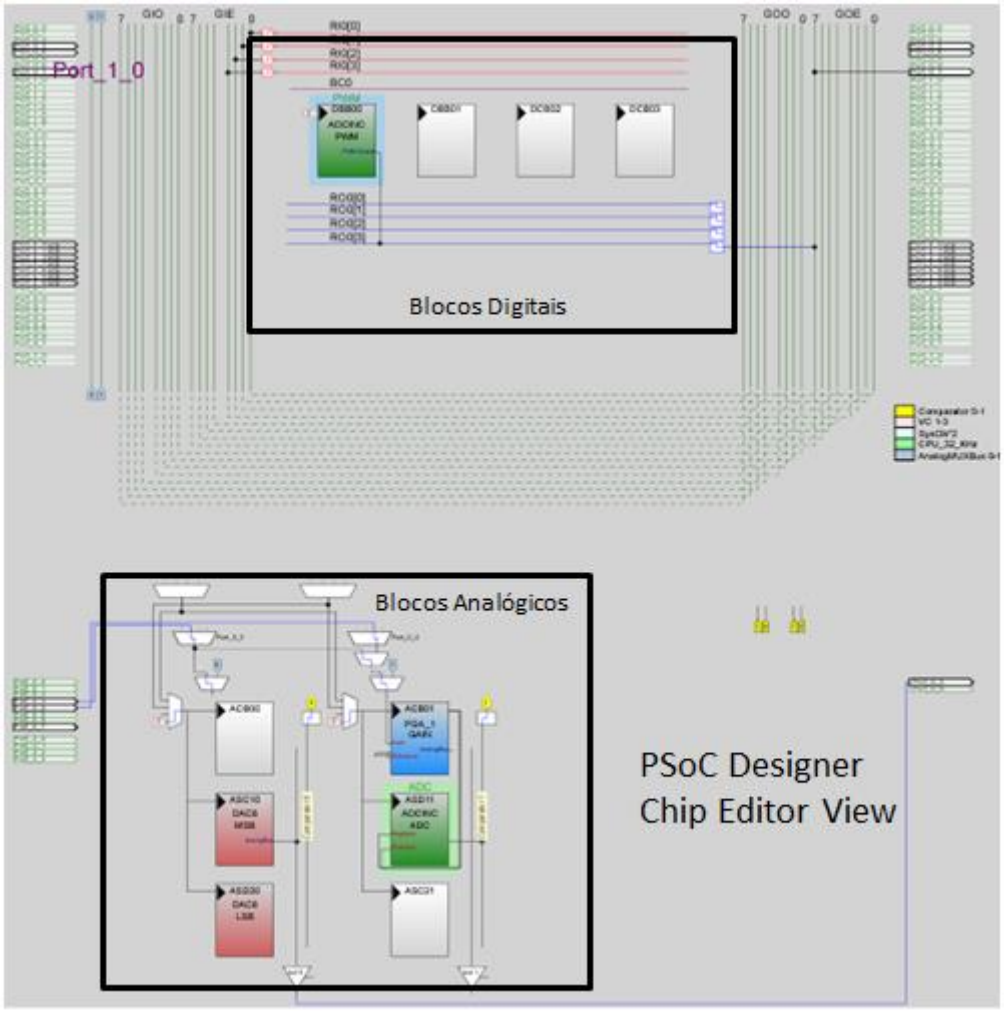
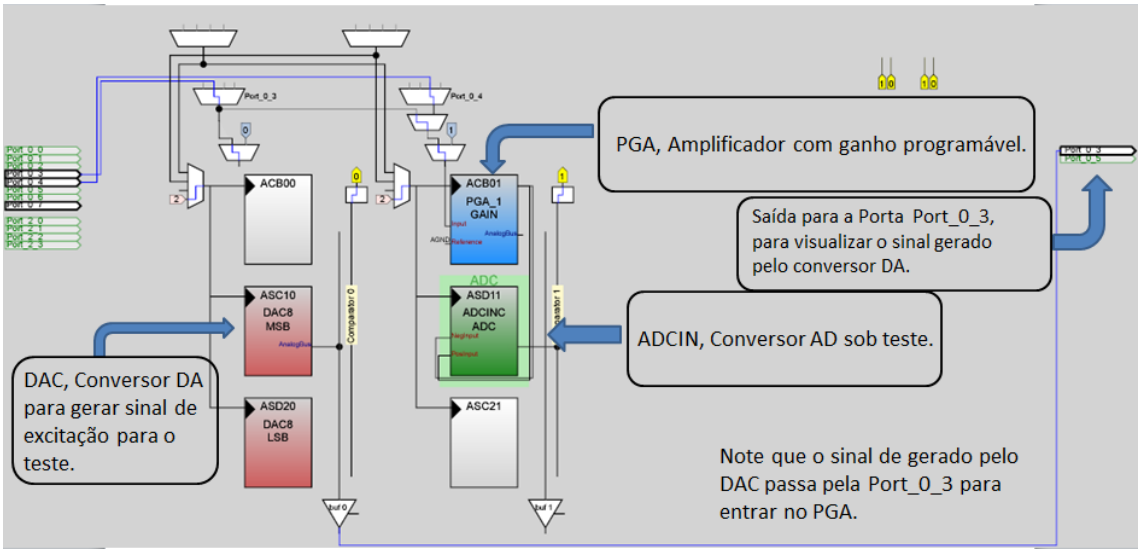


Figura 24: PSoC Designer, janela de edição e conexão dos blocos analógicos e digitais.



Na Figura 25 tem-se o detalhe das conexões dos blocos analógicos utilizados nesta metodologia de teste. É possível visualizar o PGA, ADC, DAC com conexões de acordo com o apresentado no diagrama de blocos da Figura 19. Note que os blocos dispendidos para o uso do conversor DA são um total de dois blocos e de posicionamento limitado.

Figura 25: Configuração dos blocos analógicos para o BIST.



Depois de inseridos os blocos foram configurados, a Figura 26 apresenta a janela de configuração do conversor AD utilizado, nesta janela foram selecionados os números de bits, a referência de *clock* como sendo o VC2 e o tipo de dados de saída como *Unsigned*.

Figura 26: Configuração ADCINC.

Parameters - ADCINC

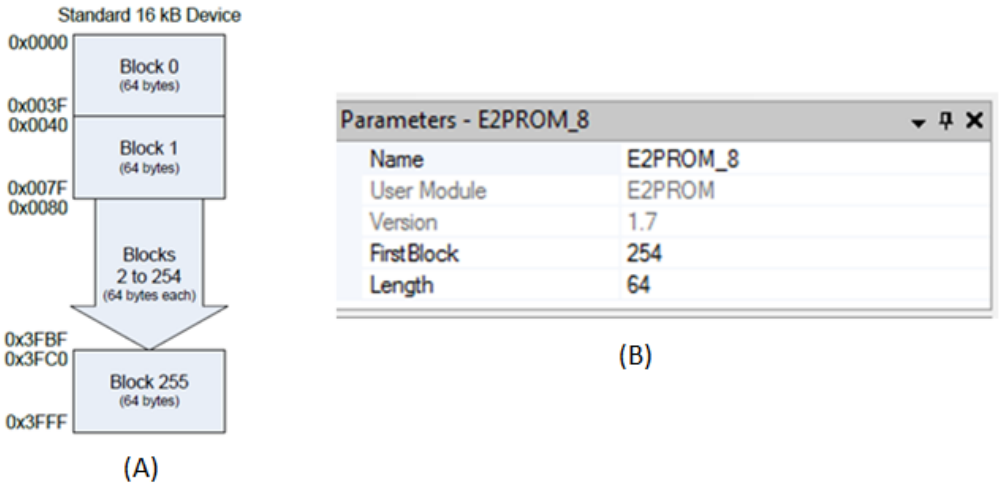
Name	ADCINC
User Module	ADCINC
Version	1.20
DataFormat	Unsigned
Resolution	8 Bit
Data Clock	VC2
ClockPhase	Normal
PosInput	ACB01
NegInput	ACB01
NegInputGain	Disconnected
PulseWidth	1
PWM Output	Row_0_Output_3

Com o valor *Data Clock* da Figura 26 sendo de 1,5MHz configurado de acordo com a Figura 20, e o conversor configurado para 8 bits de resolução, aplicando essas informações na Equação 6 tem-se que a frequência de amostragem é de aproximadamente 1,17ksps. A faixa de conversão do sinal analógico de tensão foi configurada de acordo com a Figura 20 e equivale ao intervalo de 1,3V até 3,9V.

O uso do bloco E2prom permite abstrair da gravação de blocos de dados de 64 Bytes inerentes ao uso da arquitetura flash deste dispositivo para a escrita e leitura de Bytes induvidual, por exemplo. Vale ressaltar que o modulo E2prom continua a escrever os 64Bytes do bloco, mas o usuario não se envolve diretamente com esta manipulação.

A Figura 27a ilustra a divisão do espaço de memoria Flash de 16kB deste dispositivo em 256 blocos de 64 bytes com os respectivos endereços. A Figura 27b apresenta a janela de configuração do modulo E2PROM_8 onde é possivel perceber que este foi configurado para acessar o bloco 254 com um tamanho de 64 bytes.

Figura 27: Blocos de memória Flash (A) e configuração do modulo E2prom(B).



Por definição o compilador começa a alocar os dados referentes ao programa nos endereços de memória mais baixos, por consequência é interessante que os módulos E2prom sejam alocados nos endereços de memória mais altos para que não ocorra alteração dos dados de programa.

Um fator importante quando for utilizada a escrita na memória Flash, seja de forma direta ou usando o módulo E2prom é o parâmetro temperatura. Se o dispositivo

for operar na faixa de 0°C até 50°C as variações de comportamento nesta faixa podem ser desprezadas e a temperatura pode ser configurada como 25°C, mas se o dispositivo for utilizado na faixa industrial de -40°C até 85°C deve ser agregado ao dispositivo a capacidade de medir a temperatura ambiente e então utilizá-la como parâmetro de entrada no processo de configuração do módulo E2prom, por exemplo.

Nesta aplicação foi utilizado o parâmetro temperatura como sendo 25°C e foram utilizados os blocos E2prom para salvar o histograma. A Tabela 2 apresenta o mapa de memória da memória E2prom bem como os respectivos Blocos e a faixa de códigos do histograma que é armazenada.

Tabela 2: Mapa de memória, armazenamento do histograma.

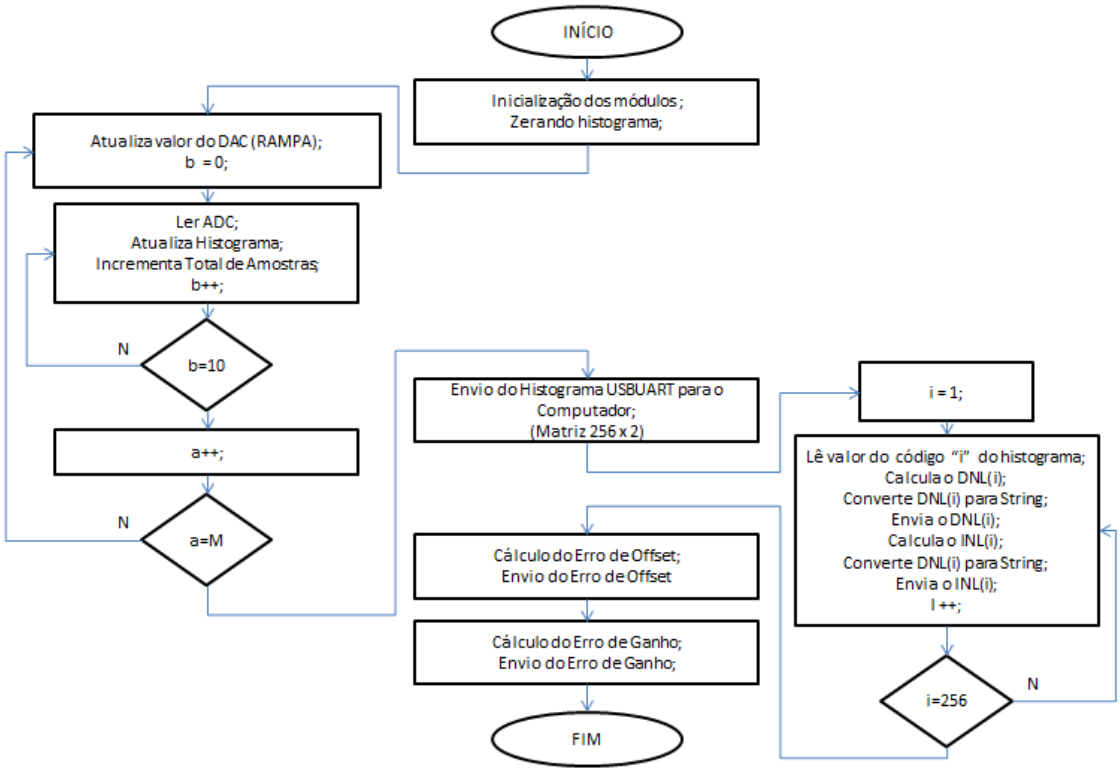
E2PROM	Bloco Flash	Endereço bloco Flash	Intervalo dos Códigos
E2PROM_1	247	3DC0	0 - 31
E2PROM_2	248	3E00	32 - 63
E2PROM_3	249	3E40	64 - 95
E2PROM_4	250	3E80	96 - 127
E2PROM_5	251	3EC0	128 - 159
E2PROM_6	252	EF00	160 - 191
E2PROM_7	253	EF40	192 - 223
E2PROM_8	254	EF80	224 - 255

O conversor DA , bloco DAC8, utilizado permite a escrita de códigos entre 0 e 254, onde 0 representa o menor valor de tensão de saída e o código 254 o maior valor de tensão alcançado pela saída deste conversor.

O fato apresentado acima tem grande impacto no autoteste, pois deste modo foi verificado que o código 0 não é lido no conversor AD, bloco ADCINC.

Uma vez que os blocos foram inseridos no esquemático do PSoC Designer, a etapa seguinte consiste em desenvolver o código em linguagem de programação C para realizar o autoteste. A Figura 28 apresenta o fluxograma elaborado para a metodologia de teste BIST, onde o dispositivo gera o sinal de excitação, aplica – o ao conversor sob AD sob este, processa as informações e transmite os resultados.

Figura 28: Fluxograma BIST.



Nesta configuração foram adquiridas 280 amostras por código, resultando em uma incerteza de 1/280LSB que equivale a 0,003LSB, uma incerteza associada ao DAC de 0,125LSB, resultando em uma incerteza total de 0,13LSB.

O código completo implementado baseado no fluxograma apresentado na Figura 28 está no APÊNDICE A. Trechos e comentários sobre código são apresentados na sequência.

Inicialmemte tentou-se utilizar a memória SRAM para armazenar o histograma, agregando maior velocidade e facilidade de implementação, mas o dispositivo não apresenta memória SRAM suficiente para armazenar as 256 variáveis do tipo *int* (2 bytes) do histograma, portanto fez-se necessário o uso da memória Flash. Uma das formas de armazenar na memória flash é a escrita direta do bloco inteiro, os 64 bytes, a outra forma é usar o módulo E2prom que emula uma memória E2prom dentro da memória Flash, conforme mensionado anteriormente. A alternativa escolida foi o uso do Módulo E2prom pois proporciona um uso mais eficiente da memoria Flash.

Vale lembrar que para escrever e ler os blocos da memoria flash é necessário alterar o o arquivo *flashsecurity.txt* acessível no PSoC Designer.

Um artifício interessante para manipular as variáveis foi declarar variáveis de tipos diferentes de tal forma que ocupem o mesmo endereço de memória, isto foi feito conforme o código apresentado no Quadro 1.

Quadro 1: Declaração da variável de manipulação do histograma.

```
union Histograma{
        unsigned int int_countcode;
        unsigned char char_countcode[2];
}countcode;
```

Este procedimento facilitou o uso da função de escrita e leitura de dados na memória Flash por meio do bloco E2prom, pois esta função utiliza variáveis do tipo *char* neste processo e durante a contagem do número de ocorrência de cada código é mais conveniente tratar os dados como variáveis do tipo *int*.

Como a variável que conta o número de ocorrência de cada código binário do conversor AD é do tipo *int*, dois bytes, antes de transmitir é necessário converter este valor para uma variável do tipo *string*. Para tanto foi utilizada a função *csprintf(..., "%d",...)*. a função *Atualizacontagem()* agora é utilizada para ler o valor do histograma, pois a *flag_histOUdnl* recebeu o valor 1, o trecho do código responsável por transmitir o histograma é apresentado no Quadro 2.

Quadro 2: Código de envio do histograma.

```
//******************** ENVIANDO HISTOGRAMA ************************
        flag_histOUdnl = 1;
        bdata = 0;
        for (a = 0; a < 256; a++){
                // bdata de 0 ate 255 para cobrir todos os bins
                atualizacontagem(); // le o valor e escreve em countcode.int_countcode
                while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                csprintf(stringTOpc4,"%d",countcode.int_countcode);
                USBUART_PutString(stringTOpc4);
                while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                USBUART_CPutString(",0 \n\r"); // envia zero para completar matriz 2x2
matlab
                bdata++;
        }
```

Vale ressaltar que o formato de envio dos dados é feito de tal forma que constitua uma matriz de duas colunas, por isso é enviado uma virgula e um zero logo após o envio de cada valor do histograma, de acordo com o comando *USBUART_CPutString(",0 \n\r")*. Isto é feito para que no momento de processar os dados no computador o processo seja facilitado.

O Cálculo do DNL é realizado em ponto flutuante, variável de quatro bytes, é necessário converter esta variável para o formato string para que se possa enviá-la via USBUART. Infelizmente a função *csprintf(...,"%f",...)* não pode ser usada por que ela necessita de grande espaço de memória de programa, espaço este que é escasso e limitado para esta aplicação. Para contornar essa adversidade foi utilizado a função *ftoa()* que converte valores do tipo *float* para *string* necessitando um menor espaço de memória de programa, mas em contrapartida esta função apresenta certas limitações e não converte valores menores que 0,0000001, então foi necessário inserir algumas linhas de código e artifícios para poder utilizá-la, por exemplo, quando a variável apresenta valor negativo o caracter "-" é enviado via USBUART e na sequência a variável *float* é multiplicado por "-1" e convertida usando a função *ftoa()* e por fim enviada, conforme apresentado no Quadro 3

Quadro 3: Cálculo e envio do DNL.

```
DNL.float_DNL = (countcode.int_countcode - histograma_ideal)/histograma_ideal;
INL.float_INL = DNL.float_DNL + INL.float_INL;
// ********** Enviar DNL ***************
if (DNL.float_DNL < 0.000000){ // função ftoa somente para positivos
        DNL.float_DNL = -1.0*DNL.float_DNL;
        if (DNL.float_DNL != 0.0){
                while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                USBUART_CPutString("-");// enviar sinal negativo
        }
}
if (DNL.float_DNL > 0.000001){
        pointer = ftoa ((float )DNL.float_DNL,&sstatus );
        strncpy(asciistring , pointer, 6 );
        //csprintf(stringTOpc8,"%f",DNL.float_DNL);
        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
        USBUART_PutString(asciistring);
}
else { // enviar 0.0
        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
        USBUART_CPutString("0.0");// enviar sinal negativo
}
// ********** Enviar separador da matriz ***************
while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
USBUART_CPutString(",");
```

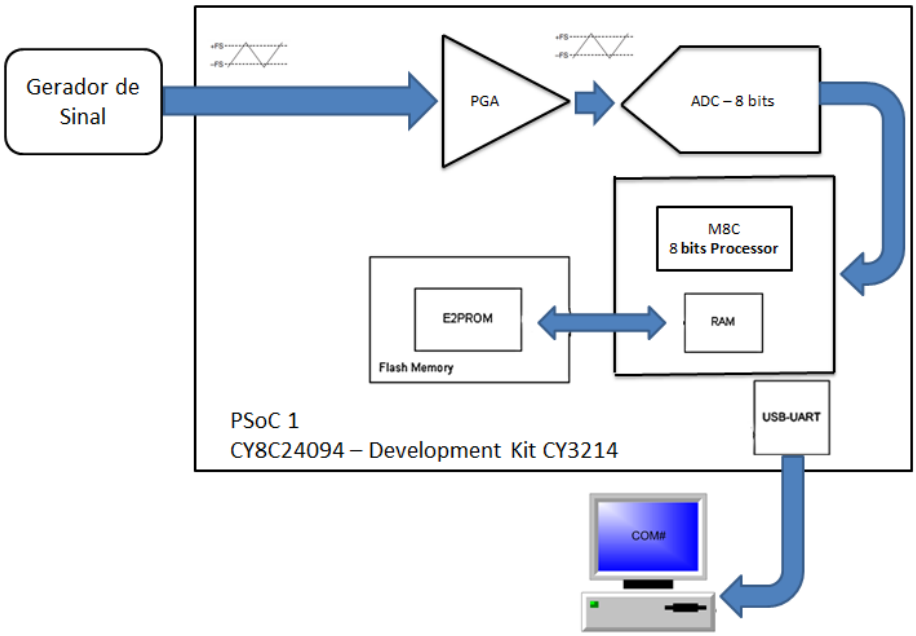
Novamente é enviada a vírgula como separador da matriz, mas nesse caso o valor que é enviado na sequência é o INL do respectivo código binário que foi calculado o DNL e não o valor 0 como no caso do envio do histograma. O procedimento de envio

do INL é semelhante ao descrito anteriormente para o envio do DNL, pois ambas variáveis são do tipo *float*, contendo 32 bits.

O código em sua versão final ocupou cerca de 94% da área de memória de programa que está situada na memória Flash, parte destes 6% restante é destinado para armazenar o histograma. Isto indica parte da complexidade em aplicar o BIST.

A outra abordagem para implementar o método do histograma neste dispositivo seria aplicar uma excitação externa, método este e características descritas na seção 4. A Figura 29 apresenta o diagrama de blocos desta abordagem. Vale notar que a alteração que ocorreu em relação ao diagrama de blocos apresentado na Figura 19 é a não utilização do bloco DAC, os demais blocos são os mesmos e executam a mesma lógica de funcionamento.

Figura 29: Diagrama de blocos método histograma com excitação externa.



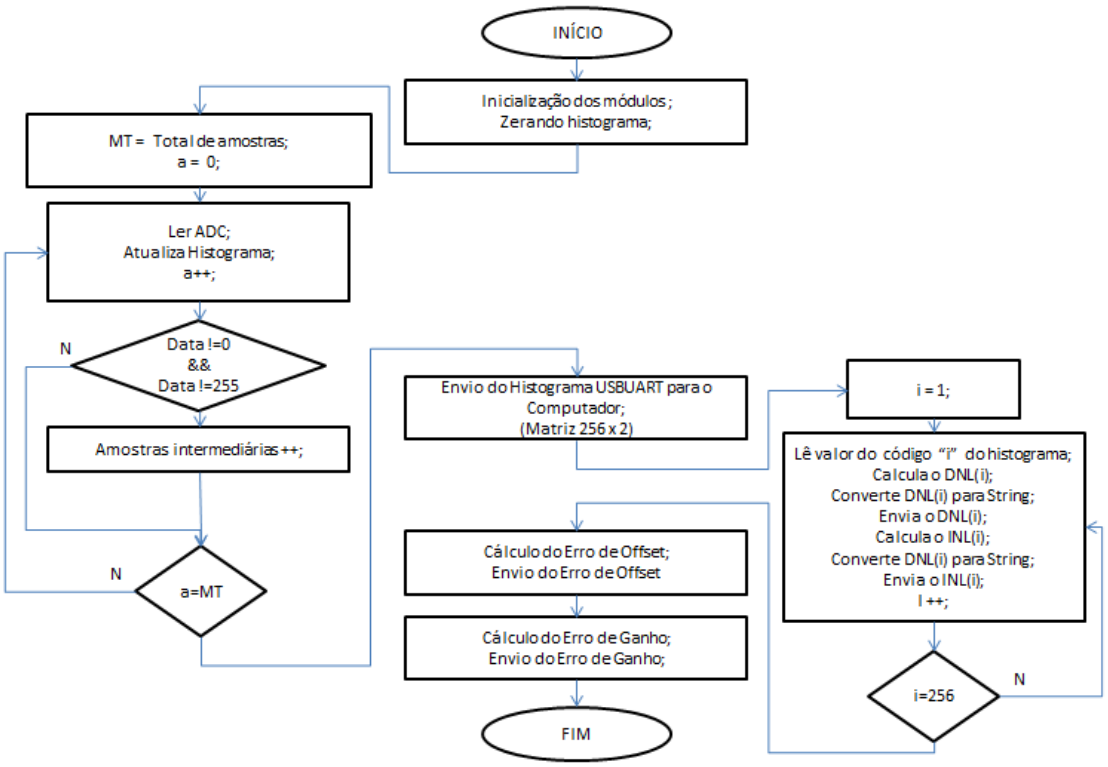
Para uma resolução de 0,1LSB e um nível de confiança de 99%, aplicando na Equação 7 temos um total de 266840 amostras necessárias para um conversor de 8 bits. Com o intuito de garantir que todo o *Range* de entrada seja estimulado pelo sinal externo do tipo rampa, um *Overdrive* de cerca de 10% para mais e para menos é tipicamente aplicado segundo (KESTER, 2005). Com isso um total de 332800 amostras deveriam ser obtidas como garantia de alcançar o número mínimo de amostras por código.

A frequência do sinal de excitação deve ser baixa e não sub harmônica da frequência de amostragem conforme mencionado anteriormente.

A

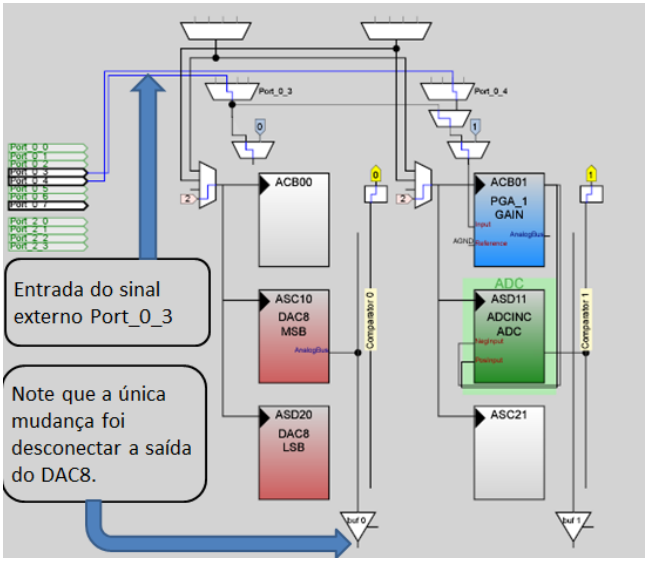
Figura 30 apresenta o fluxograma para a metodologia de teste em que o sinal de excitação do conversor ADC sob teste tem origem externa e não relacionada com a frequência de amostragem do dispositivo sob teste. Durante a parte de aquisição do histograma é exercido um controle das amostras intermediarias, ou seja, o total de amostras menos o número de amostras dos códigos dos extremos, isto é feito para que os códigos extremos não entrem no cálculo das não linearidades, pois o sinal de excitação foi aplicado com certo *Overdrive.*

Figura 30: Fluxograma da metodologia de teste utilizando excitação externa.



Uma das vantagens de se utilizar um PSoC é que a alteração necessária no diagrama de blocos do PSoC Designer utilizado na metodologia BIST para adequar a esta metodologia é desconectar a saída do DAC8, conforme ilustra a Figura 31.

Figura 31: Configuração dos blocos analógicos para método de teste com excitação externa.



O trecho de código apresentado no Quadro 4 faz referência a aquisição do histograma, levando em consideração que a rampa tem origem externa.

Quadro 4: Código de aquisição do histograma utilizando sinal de excitação externo.

```
/************* LOOP para amostragem estocástica *****************/
//totalamostras = 332800 equivale a 1300 amostras por bin
totalamostras = 0;
flag_histOUdnl = 0; // configura escrita do histograma
for (totalamostrastotal=0; totalamostrastotal<332800; totalamostrastotal++){
        ADCINC_GetSamples(1); //leio ADC
        while (ADCINC_fIsDataAvailable()==0);// Loop until value ready
        bdata = ADCINC_cClearFlagGetData();
        if (bdata != 0 && bdata != 255){
                totalamostras ++;  // controle do num amostras entre 1 - 254
        }
        atualizacontagem(); // atualiza valor bdata lido no histograma da flash
        LCD_Position(0,14);
        LCD_PrHexByte(bdata);// Escreve no LCD valor lido no ADC
        ltoa (stringTOpc16,totalamostrastotal, 10);
        LCD_Position(1,4);
        LCD_PrString(stringTOpc16);
}
ltoa (stringTOpc16,totalamostras, 10);
LCD_Position(0,7);
LCD_PrString(stringTOpc16);
/************* FIM LOOP amostragem estocástica *****************/
```

Vale notar que a variável *totalamostras* não contabiliza a contagem dos códigos extremos.

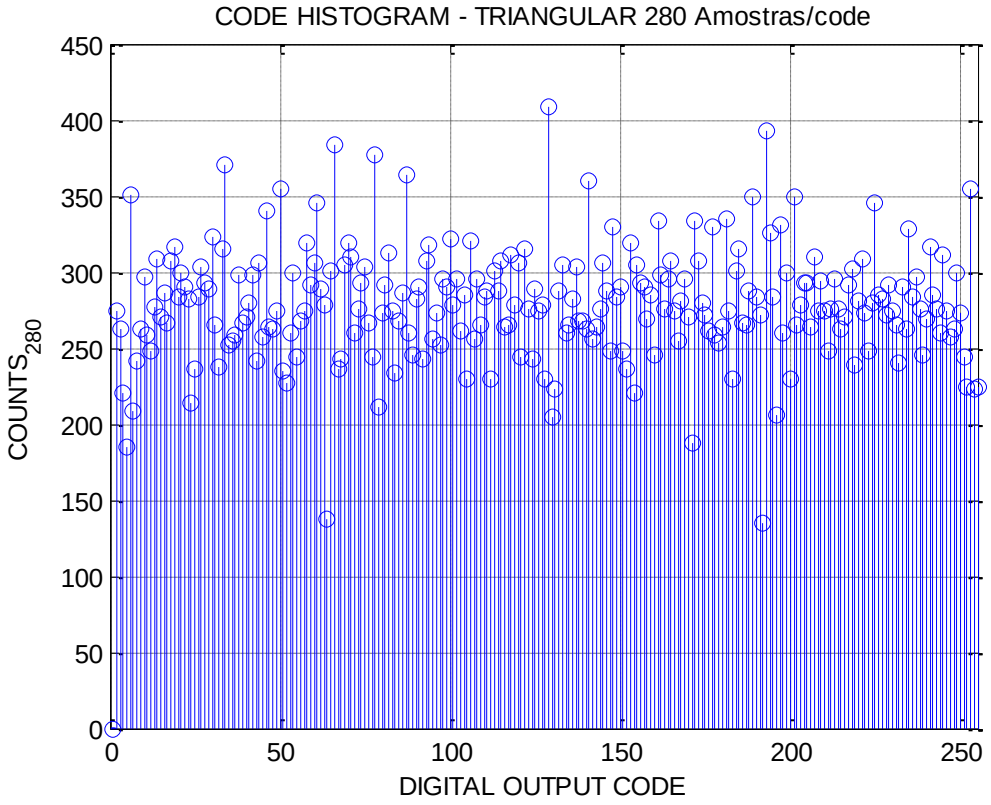
Em suma, as alterações necessárias para aplicar este método em relação ao método BIST é desconectar a saída do DAC8, trocar o código responsável pelo armazenamento do histograma.

6. RESULTADOS

Após concluir as configurações dos módulos do PSoC e implementar o firmware que realiza o BIST conforme descritos anteriormente, ao executar o programa os dados são adquiridos, processados e enviados para o computador. Para visualizar os dados recebidos monitorou-se a porta USB do computador por meio do software TeraTerm e foi gerado um arquivo contendo os dados recebidos. Estes dados foram processados pelo software Matlab, código presente no Apêndice B, os gráficos foram gerados.

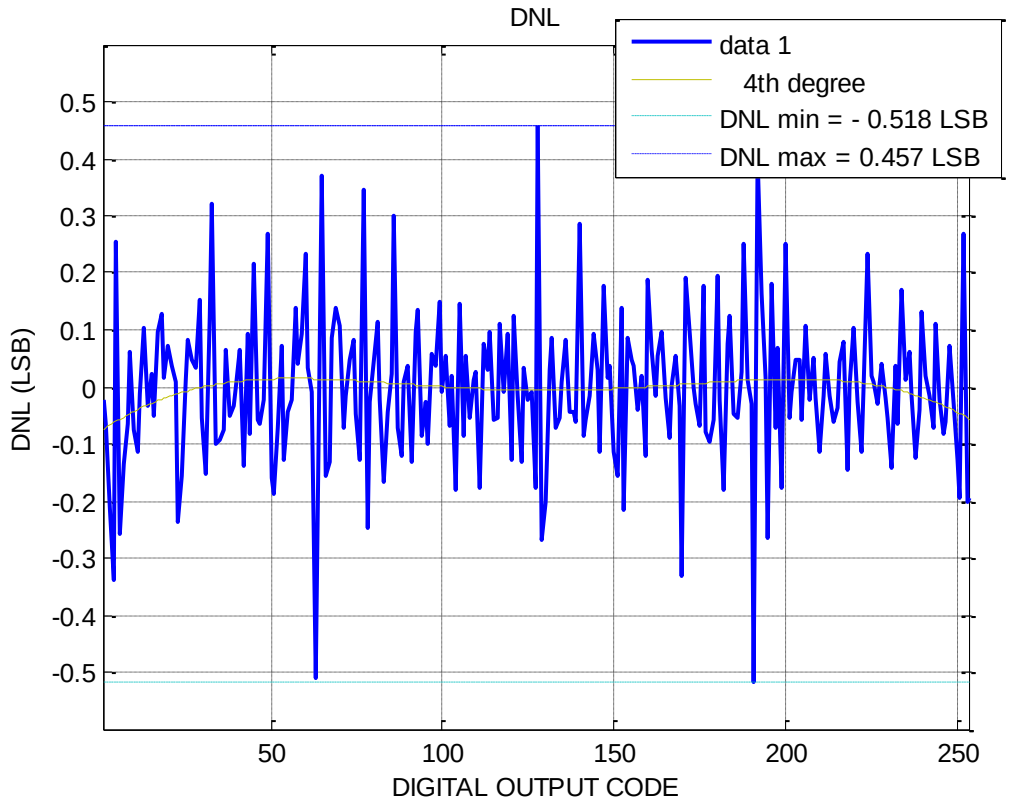
A Figura 32 apresenta o histograma do autoteste embarcado realizado, o número de amostras ideal por código nesta situação foi de 280 amostras/código.

Figura 32: Histograma obtido via BIST referente ao número de amostras para cada código de saída.



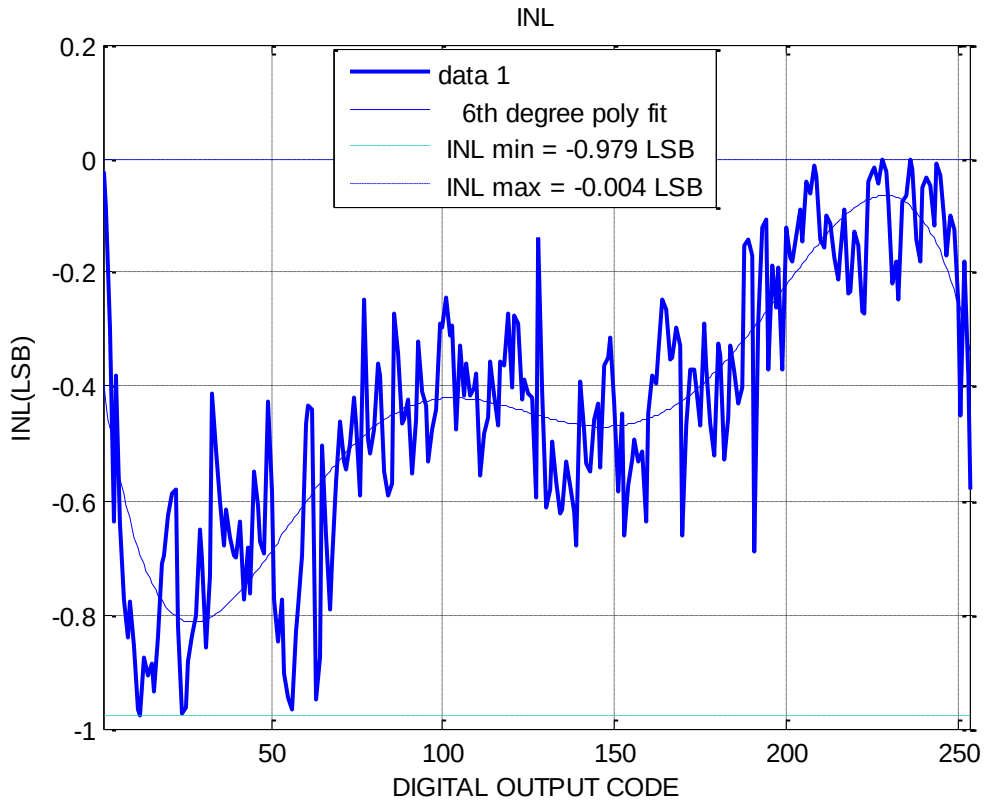
A Figura 33 apresenta o DNL do conversor DA testado no conceito BIST, vale notar que o maior DNL em módulo foi de 0,518LSB e que segundo o Datasheet deste conversor o valor típico de DNL é de 0,6LSB.

Figura 33: DNL do conversor ADCINC via BIST.



O INL para o ADCIN testado está ilustrado na Figura 34, onde é possível observar que o maior valor do INL em módulo é de 0,978LSB e a título de comparação o valor típico para esta não linearidade informado no Datasheet deste componente é de 0,7LSB.

Figura 34: INL do Conversor ADCINC via BIST.



O erro de *offset* estimado com este método de teste foi de 0,302LSB que equivale a 3mV na faixa em que o conversor foi configurado (1,3V até 3,9V) . O valor típico para este parâmetro para uma faixa de 0V até 5V é de 5mV. O ganho estimado foi de 1,001, que é muito próximo ao valor ideal que é unitário.

7. CONCLUSÕES

Este trabalho tratou de caracterizar um conversor AD presente no PSoC1, um sistema programável de sinal misto, quanto às características estáticas de desempenho utilizando o método do histograma e aplicando o conceito BIST, onde o próprio dispositivo é responsável por gerar o sinal de excitação, gerenciar o autoteste, tratar os resultados e enviá-los. O inconveniente foi o longo tempo de teste, devido a limitações do dispositivo, como tratamento dos dados relacionado ao espaço de memória e também por se tratar de um processador de 8 bits que manipulou dados de 32bits.

No Capítulo 2 deste trabalho foi descrito em linhas gerais o sistema programável de sinal misto utilizado, o PSoC1, bem como algumas de suas características relevantes para esta aplicação.

No Capítulo 3 foi abordado o conversor AD em que foi realizado o teste para obter os parâmetros de interesse.

O Capítulo 4 teve como tema o método do histograma e suas peculiaridades de aplicação, foram apresentadas as equações para o cálculo do *offset*, do ganho, do DNL e INL. Foram apresentadas as diferenças em se gerar o sinal de excitação internamente por meio de DAC e externamente via gerador de sinais, foi abordado também um dos fatores mais importantes para a execução do método do histograma que é a quantidade de amostras necessárias bem como o tipo de sinal de excitação e suas implicações.

No Capítulo 5 foi descrito como o método de autoteste foi aplicado no PSoC1 visando a obtenção das características estáticas de desempenho.

O Capítulo 6 apresentou os resultados obtidos, sendo eles: *offset* de 3mV, Ganho praticamente unitário, DNL máximo de 0,518LSB e INL máximo de 0,98LSB, resultados estes considerados satisfatórios e a metodologia de teste cumpriu com o objetivo.

8. REFERÊNCIAS BIBLIOGRÁFICAS

AN 641, Maxim IC Application Note, “ADC and DAC glossary”, 2014;

AN2096, Cypress Application Note, “PSoC1 Using the ADCINC ADC”, 2014.

AN2219, Cypress Application Note, “PSoC1 Selecting Analog Ground and Reference”, 2014;

AN74170, Cypress Application Note, “PSoC 1 Analog Structure and Configuration With PSoC Designer”, 2014;

CYPRESS TRM, “PSoC Technical Reference Manual (TRM)”, Cypress Semiconductor Corporation, Document No. 001-14463 Rev. *H, 2013;

CYPRESS WEB SITE, <http://www.cypress.com/>, Cypress Semiconductor Corporation Web Site, acesso em Outubro de 2014.

DOERNBERG, J. “Full-Speed Testing of A/D Converters”. IEEE Journal of Solid-State Circuits, pp.820-827, 1984;

FLORES, Maria da Glória Cataldi Couto, “Teste Embarcado de Conversores Analógico-Digitais”, Dissertação de Mestrado, UFRGS, Porto Alegre, 2003;

IEEE STD 1241, “IEEE standard for terminology and test methods for analog-to-digital converters”, Institute of Electrical and Electronics Engineers Inc., 2010;

MÁRKUS, János Márkus, José Silva, Gabor C. Temes. “Theory and Applications of Incremental Delta-Sigma Converters”, IEEE Transactions on circuits and systems—i: regular papers, vol. 51, no. 4, APRIL 2004

KESTER, Walt, “Data Conversion Handbook”, Analog Devices, 2005, ISBN: 0-7506-7841-0;

KOOK, Se Hun, “Low-Cost Testing of High-Precision Analog-to-Digital Converters”, Dissertation Thesis for the Degree Doctor of Philosophy , Georgia Institute of Technology, Atlanta, GA, 2011;

LIBERALI, V. Liberali, F. Maloberti and M. Stramesi, "ADC Characterisation Using the Code Density Test Method With Deterministic Sampling", Proc. International Mixed Signal Testing Workshop, pp.113-18, 1996;

MELKONIAN, L. “Dynamic Specifications for Sampling AD Converters”. National Semiconductors, application note nº769, 1991;

NOVAK, Franc Novak, Peter Mrak, Anton Biasizzo, “Test Strategies for Embedded ADC Cores in a System-On-Chip, a Case Study” Computer Systems

Department, Joˇzef Stefan Institute Computing and Informatics, Vol. 31, 2012, 411–426;

RENOVELL, M. Renovell, F. Azaïs, S. Bernard and Y. Bertrand “Hardware Resource Minimization for Histogram-Based ADC BIST”. Proceedings of 18th IEEE VLSI Test Symposium, pp.247-252, 2000;

9. APÊNDICE A – Código Principal

```
/*
ADC_TEST_V8
24/11/2014
- Port_0_7 PWM ADCIN
Powered by Aquiles
*/
//-------------------------------------------------------------------------
// C main line
//-------------------------------------------------------------------------

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define  TEMPERATURE       25

//***************************** VARIÁVEIS ************************************
unsigned char bdata; // variavel de atualizacontagem
unsigned char i;    // Variable for zerarregistradores, Delay loop DAC
unsigned char b; // loop for deterministico
unsigned char flag_dac = 0; // flag de controle da rampa subida e descida dac
unsigned char flag_histOUdnl; //flag para ler escrever hist ou ler para dnl
unsigned char writebcode = 0; // variavel de programação para escrita na DAC, rampa
unsigned char bError;
char stringTOpc4[4]; // para envio do histograma
char stringTOpc16[16];
char asciistring[8];
char * pointer;
int sstatus;
int  ADDRESS_OFFSET = 0; // variavel de controle da posição dos blocos e2prom
int flag_ftoa; //varialve uso na conversão de float to string
int H1;
unsigned int a;// loop for deterministico
unsigned long totalamostras;
float histograma_ideal;
float INLtoPC;
float offset;
float ganho;
union DNL_data{
        float float_DNL;
        unsigned char char_DNL[4];
}DNL;
union INL_data{
        float float_INL;
        unsigned char char_INL[4];
}INL;
union Histograma{
        unsigned int int_countcode;
```

```
        unsigned char char_countcode[2];
}countcode;

//****************************** FUNÇÕES ***************************************

void atualizacontagemB1(int ADDRESS_OFFSET){ //bloco 1 e2prom...codigos 0 ate 63
        ADDRESS_OFFSET = ADDRESS_OFFSET*2;
        E2PROM_1_E2Read(ADDRESS_OFFSET,(unsigned char*)&countcode.char_countcode ,sizeof
(countcode.char_countcode));
        if (flag_histOUdnl == 0){
                countcode.int_countcode ++;
                bError = E2PROM_1_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
        }
}
void atualizacontagemB2(int ADDRESS_OFFSET){ //bloco 2 e2prom...codigos 64 ate 95
        ADDRESS_OFFSET = (ADDRESS_OFFSET-32);
        ADDRESS_OFFSET = ADDRESS_OFFSET*2;
        E2PROM_2_E2Read(ADDRESS_OFFSET,(unsigned char*)&countcode.char_countcode ,sizeof
(countcode.char_countcode));
        if (flag_histOUdnl == 0){
                countcode.int_countcode ++;
                bError = E2PROM_2_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
        }
}
void atualizacontagemB3(int ADDRESS_OFFSET){ //bloco 3 e2prom...codigos 96 ate 127
        ADDRESS_OFFSET = (ADDRESS_OFFSET-64);
        ADDRESS_OFFSET = ADDRESS_OFFSET*2;
        E2PROM_3_E2Read(ADDRESS_OFFSET,(unsigned char*)&countcode.char_countcode ,sizeof
(countcode.char_countcode));
        if (flag_histOUdnl == 0){
                countcode.int_countcode ++;
                bError = E2PROM_3_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
        }
}
void atualizacontagemB4(int ADDRESS_OFFSET){ //bloco 4 e2prom...codigos 128 ate 159
        ADDRESS_OFFSET = (ADDRESS_OFFSET-96);
        ADDRESS_OFFSET = ADDRESS_OFFSET*2;
        E2PROM_4_E2Read(ADDRESS_OFFSET,(unsigned char*)&countcode.char_countcode ,sizeof
(countcode.char_countcode));
        if (flag_histOUdnl == 0){
                countcode.int_countcode ++;
                bError = E2PROM_4_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
        }
}
void atualizacontagemB5(int ADDRESS_OFFSET){ //bloco 5 e2prom...codigos 160 ate 191
        ADDRESS_OFFSET = (ADDRESS_OFFSET-128);
```

```
        ADDRESS_OFFSET = ADDRESS_OFFSET*2;
        E2PROM_5_E2Read(ADDRESS_OFFSET,(unsigned char*)&countcode.char_countcode ,sizeof
(countcode.char_countcode));
        if (flag_histOUdnl == 0){
                countcode.int_countcode ++;
                bError = E2PROM_5_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
        }
}
void atualizacontagemB6(int ADDRESS_OFFSET){ //bloco 6 e2prom...codigos 192 ate 223
        ADDRESS_OFFSET = (ADDRESS_OFFSET-160);
        ADDRESS_OFFSET = ADDRESS_OFFSET*2;
        E2PROM_6_E2Read(ADDRESS_OFFSET,(unsigned char*)&countcode.char_countcode ,sizeof
(countcode.char_countcode));
        if (flag_histOUdnl == 0){
                countcode.int_countcode ++;
                bError = E2PROM_6_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
        }
}
void atualizacontagemB7(int ADDRESS_OFFSET){ //bloco 7 e2prom...codigos 224 ate 255
        ADDRESS_OFFSET = (ADDRESS_OFFSET-192);
        ADDRESS_OFFSET = ADDRESS_OFFSET*2;
        E2PROM_7_E2Read(ADDRESS_OFFSET,(unsigned char*)&countcode.char_countcode ,sizeof
(countcode.char_countcode));
        if (flag_histOUdnl == 0){
                countcode.int_countcode ++;
                bError = E2PROM_7_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
        }
}
void atualizacontagemB8(int ADDRESS_OFFSET){ //bloco 1 e2prom...codigos 0 ate 63
        ADDRESS_OFFSET = (ADDRESS_OFFSET-224);
        ADDRESS_OFFSET = ADDRESS_OFFSET*2;
        E2PROM_8_E2Read(ADDRESS_OFFSET,(unsigned char*)&countcode.char_countcode ,sizeof
(countcode.char_countcode));
        if (flag_histOUdnl == 0){
                countcode.int_countcode ++;
                bError = E2PROM_8_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
        }
}
void atualizacontagem(void){// 0 = atualiza histograma ; 1 = leitura histograma
        //direcionar o codigo BIN para respectivo bloco e contar
        if (bdata>=0&&bdata<=31){
                atualizacontagemB1(bdata);
        }
        if (bdata>=32&&bdata<=63){
                atualizacontagemB2(bdata);
        }
```

```
        if (bdata>=64&&bdata<=95){
                atualizacontagemB3(bdata);
        }
        if (bdata>=96&&bdata<=127){
                atualizacontagemB4(bdata);
        }
        if (bdata>=128&&bdata<=159){
                atualizacontagemB5(bdata);
        }
        if (bdata>=160&&bdata<=191){
                atualizacontagemB6(bdata);
        }
        if (bdata>=192&&bdata<=223){
                atualizacontagemB7(bdata);
        }
        if (bdata>=224&&bdata<=255){
                atualizacontagemB8(bdata);
        }
}
void zerarregistradores(void){
        countcode.char_countcode[0] = 0;
        countcode.char_countcode[1] = 0;
        ADDRESS_OFFSET = 0;
        for (i=0; i!=63; i++){
                bError = E2PROM_1_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
                bError = E2PROM_2_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
                bError = E2PROM_3_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
                bError = E2PROM_4_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
                bError = E2PROM_5_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
                bError = E2PROM_6_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
                bError = E2PROM_7_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
                bError = E2PROM_8_bE2Write(ADDRESS_OFFSET,(unsigned
char*)&countcode.char_countcode, sizeof (countcode.char_countcode), TEMPERATURE);
                ADDRESS_OFFSET++;
        }
}
void atualizaDAC(void){
        if (flag_dac == 0){
                if (writebcode != 254){
                        DAC8_WriteStall(writebcode);
                        writebcode++;
                }
                else {
```

```
                        DAC8_WriteStall(writebcode);
                        flag_dac = 2;
                }
        }
        if(flag_dac == 1){
                if (writebcode != 0){
                        DAC8_WriteStall(writebcode);
                        writebcode--;

                }
                else {
                        flag_dac = 0;
                        DAC8_WriteStall(writebcode);
                }
        }
        if (flag_dac == 2){
                flag_dac = 1;
        }
        for(i = 0xFF; i != 0; i--);// Delay loop , estabilizar DAC
        for(i = 0xFF; i != 0; i--);// Delay loop , estabilizar DAC
}

//****************************** MAIN ********************************************
void main(void)
{
        M8C_EnableGInt ; // Enable Global Interrupts
        USBUART_Start(USBUART_5V_OPERATION); /*Start USBUART 5V operation */
        while(!USBUART_Init()); /* Wait for Device to initialize */
        PGA_1_Start(3); //Start with high power
        ADCINC_Start(3); //Start with high power
        DAC8_Start(3); // Start DAC8 in HIGH power mode
        LCD_Start();
        LCD_Position(0,0);
        LCD_PrCString("ADC_V8 DET         ");
        LCD_Position(1,0);
        LCD_PrCString("Zerando Reg...  ");
        zerarregistradores(); //  escreve zero nos blocos da mem flash
        LCD_Position(0,12);
        LCD_PrCString("D_  ");
        LCD_Position(1,0);
        LCD_PrCString("TOT             ");

        /*************LOOP para amostragem deterministica *****************/
        /******DAC to ADC totaldeamostras 10loopx10read por bin ************/
        totalamostras = 0;
        flag_histOUdnl = 0;
        //7140 > 280 por bin
        for (a=0;a<7140;a++){// pois o dac é de 0 ate 254, multiplos de 255
                atualizaDAC (); // Escreve DAC, sinal triangular
                LCD_Position(0,9);
```

```
                LCD_PrHexByte(writebcode);
                for (b =0 ; b < 10;b ++){ // loop que repete 10 vezes para cada bin code
                        ADCINC_GetSamples(1); //leio ADC
                        while (ADCINC_fIsDataAvailable()==0);// Loop until value ready
                        bdata = ADCINC_cClearFlagGetData();
                        //flag_histOUdnl = 0;
                        atualizacontagem(); // atualiza valor bdata lido no histograma da
flash
                        totalamostras++;
                        LCD_Position(0,14);
                        LCD_PrHexByte(bdata);// Escreve no LCD valor lido no ADC
                }
                ltoa (stringTOpc16,totalamostras, 10);
                LCD_Position(1,4);
                LCD_PrString(stringTOpc16);
        }
        LCD_Position(1,14);
        LCD_PrCString("OK");

        //******************  COMUNICAÇÃO USBUART *************************
        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
    USBUART_CPutString("/* ADC TEST - PSOC 1 Cypress */\n\r");
        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
        USBUART_CPutString("/* Powered by AQUILES V8 */\n\r");
        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
    USBUART_CPutString("/************************/\n\r");
        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
        USBUART_CPutString("/****** MATRIZ [HISTOGRAMA , 0] ******/\n\r");
        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */

        //******************** ENVIANDO HISTOGRAMA ************************
        flag_histOUdnl = 1;
        bdata = 0;
        for (a = 0; a < 256; a++){
                // bdata de 0 ate 255 para cobrir todos os bins
                atualizacontagem(); // le o valor e escreve em countcode.int_countcode
                while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                csprintf(stringTOpc4,"%d",countcode.int_countcode);
                USBUART_PutString(stringTOpc4);
                while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                USBUART_CPutString(",0 \n\r"); // envia zero para completar matriz 2x2
matlab
                bdata++;
        }
        //******************** CALCULANDO E ENVIANDO DNL & CALC INL
************************
//      End Points Eliminated for DNL and INL Calculations
        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
        USBUART_CPutString("/******** MATRIZ [DNL, INL]********/\n\r");
        histograma_ideal = totalamostras/255.0;
```

```
        bdata = 1;
        flag_histOUdnl = 1;
        INL.float_INL = 0.0;
        for (a = 1; a < 255; a ++){
                atualizacontagem(); //ler valor do countcode referente ao bdata
                DNL.float_DNL = (countcode.int_countcode -
histograma_ideal)/histograma_ideal;
                //if (bdata != 0 && bdata != 255){
                INL.float_INL = DNL.float_DNL + INL.float_INL;
                //}
                // ********** Enviar DNL ***************
                if (DNL.float_DNL < 0.000000){ // função ftoa somente para positivos
                        DNL.float_DNL = -1.0*DNL.float_DNL;
                        if (DNL.float_DNL != 0.0){
                                while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                                USBUART_CPutString("-");// enviar sinal negativo
                        }
                }
                if (DNL.float_DNL > 0.000001){
                        pointer = ftoa ((float )DNL.float_DNL,&sstatus );
                        strncpy(asciistring , pointer, 6 );
                        //csprintf(stringTOpc8,"%f",DNL.float_DNL);
                        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                        USBUART_PutString(asciistring);
                }
                else { // enviar 0.0
                        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                        USBUART_CPutString("0.0");// enviar sinal negativo
                }
                // ********** Enviar separador da matriz ****************
                while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                USBUART_CPutString(",");
                // ********** Enviar INL ****************
                while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                if (INL.float_INL < 0.000000){ // função ftoa somente para positivos
                        INLtoPC = -1.0*INL.float_INL;
                        if (INL.float_INL != 0.0){
                                while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                                USBUART_CPutString("-");// enviar sinal negativo
                        }
                }
                else {
                        INLtoPC = INL.float_INL;
                }
                if (INLtoPC > 0.000001){
                        pointer = ftoa ((float )INLtoPC,&sstatus );
                        strncpy(asciistring , pointer, 6 );
                        //csprintf(stringTOpc8,"%f",DNL.float_DNL);
                        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                        USBUART_PutString(asciistring);
```

```
        }
        else {// enviar 0.0
                while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                USBUART_CPutString("0.0");
        }
        //***********  Troca de linha **********
        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
        USBUART_CPutString("\n\r");
        LCD_Position(1,9);
        LCD_PrHexByte(bdata);// Escreve no LCD do DNL bin
        bdata ++;
}
//******************** CALCULANDO E ENVIANDO OFFSET ERROR e GAIN ERROR *********
while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
USBUART_CPutString("/******** MATRIZ [OFFET, GANHO]********/\n\r");
flag_histOUdnl = 1;
bdata = 1;
atualizacontagem(); //ler valor do countcode referente ao bdata
H1 = countcode.int_countcode;
bdata = 255;
atualizacontagem(); //ler valor do countcode referente ao bdata
offset = (countcode.int_countcode - H1)/(2.0*histograma_ideal);
        // ********** Enviar offset ***************
if (offset < 0.000000){ // função ftoa somente para positivos
        offset = -1.0*offset;
        if (offset != 0.0){
                while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                USBUART_CPutString("-");// enviar sinal negativo
        }
}
if (offset > 0.000001){
        pointer = ftoa ((float )offset,&sstatus );
        strncpy(asciistring , pointer, 6 );
        //csprintf(stringTOpc8,"%f",DNL.float_DNL);
        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
        USBUART_PutString(asciistring);
}
else { // enviar 0.0
        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
        USBUART_CPutString("0.0");// enviar sinal negativo
}
// ********** Enviar separador da matriz ****************
while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
USBUART_CPutString(",");
//********** Calculando o erro de ganho **********
flag_histOUdnl = 1;
bdata = 80; // faixa de calculo do ganho ( 80 to 180)
ganho = 0;
for (a = 0; a < 100; a++){
        atualizacontagem(); //ler valor do countcode referente ao bdata
```

```
                ganho = countcode.int_countcode + ganho;
                bdata ++;
        }
        ganho = ganho/(100.0*histograma_ideal);
        // ********** Enviar ganho ***************
                if (ganho < 0.000000){ // função ftoa somente para positivos
                ganho = -1.0*ganho;
                if (ganho != 0.0){
                        while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                        USBUART_CPutString("-");// enviar sinal negativo
                }
        }
        if (ganho > 0.000001){
                pointer = ftoa ((float )ganho,&sstatus );
                strncpy(asciistring , pointer, 6 );
                //csprintf(stringTOpc8,"%f",DNL.float_DNL);
                while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                USBUART_PutString(asciistring);
        }
        else { // enviar 0.0
                while (!USBUART_bTxIsReady()); /* Wait until TX is ready */
                USBUART_CPutString("0.0");// enviar sinal negativo
        }
        //************** FINAL CALCULANDO E ENVIANDO OFFSET ERROR e GAIN ERROR *********
        LCD_Position(1,13);
        LCD_PrCString("OK");

        //******************* FINAL LOOP  ****************************
        for(;;){
                i++;
        }
}
```

10. APÊNDICE B – Código para gerar os gráficos no computador, via MATLAB

```
% ***************** DETERMINISTICO ***************************
clc; clear all;close all;
[A]=textread('teraterm.log','','commentstyle','c','delimiter',',');

histogram = A(1:256,1)';
histogram = histogram;
figure(1);
stem(histogram);
grid on;
title('CODE HISTOGRAM - TRIANGULAR 280 Amostras/code');
xlabel('DIGITAL OUTPUT CODE');
ylabel('COUNTS_2_8_0');
AXIS([0 255 0 450]);

bin = 0:1:255;
bin2= 1:1:254;
DNL = A(257:510,1);
INL = A(257:510,2);

figure(2);
plot(bin2,DNL);
grid on;
title('DNL');
xlabel('DIGITAL OUTPUT CODE');
ylabel('DNL (LSB)');
AXIS([1 254 -0.6 0.6]);
figure(3);
plot(bin2,INL);
grid on;
title('INL');
xlabel('DIGITAL OUTPUT CODE');
ylabel('INL(LSB)');
AXIS([1 254 -1 0.2]);
```