UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
DEPARTAMENTO DE ENGENHARIA MECÂNICA

JAIR CURSIOL FILHO
MARCEL CAVALLINI BARBOSA

DESENVOLVIMENTO DE INTERFACE SPI PARA CONTROLE DE UM DIRECT DIGITAL SYNTHESIZER AD9912

SÃO CARLOS

2012

Jair Cursiol Filho
Marcel Cavallini Barbosa

DESENVOLVIMENTO DE INTERFACE SPI PARA CONTROLE DE UM DIRECT DIGITAL SYNTHESIZER AD9912

Orientador: *Prof. Dr. Daniel Varela Magalhães*

Trabalho de conclusão do curso de Engenharia Mecatrônica apresentado à Escola de Engenharia de São Carlos como parte dos requisitos necessários à aprovação na disciplina Trabalho de Graduação.

São Carlos
2012

Cursiol Filho, Jair e Cavallini Barbosa, Marcel
Desenvolvimento de interface SPI para controle de um Direct Digital Synthesizer AD9912 / Jair Cursiol Filho e Marcel Cavallini Barbosa - São Carlos: USP, 2012.
67f. : il.

Inclui Bibliografia.
Trabalho de Conclusão de Curso (Engenharia Mecatrônica) Universidadede São Paulo, Escola de Engenharia de São Carlos, Departamento de Mecânica.
Orientador: Prof. Dr. Daniel Varela Magalhães, Departamento de Mecânica.

1 - Sintetização de frequência 2 - Eletrônica 3 - Programação
4 - Comunicação SPI 5 - Título

FOLHA DE AVALIAÇÃO

Candidatos: Jair Cursiol Filho e Marcel Cavallini Barbosa

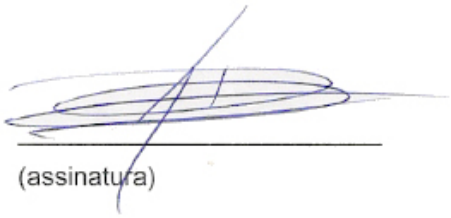
Título: DESENVOLVIMENTO DE INTERFACE SPI PARA CONTROLE DE UM DIRECT DIGITAL SYNTHESIZER AD9912

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos da Universidade de São Paulo Curso de Engenharia Mecatrônica.

BANCA EXAMINADORA

Prof. Dr. Gustavo Deckza Telles

Nota atribuída: 9,0 (nove)



(assinatura)

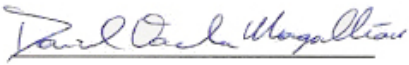
Eng.º João Marcelo Pereira Nogueira

Nota atribuída: 9,0 (nove)

(assinatura)

Prof. Dr. Daniel Varela Magalhães (orientador)

Nota atribuída: 9,0 (nove)



(assinatura)

Média: 9,0 (NOVE)

Resultado: APROVADO

12 de Março de 2012

” Todos morremos. O objetivo não é viver eternamente, mas criar algo que viverá. ”

Chuck Palahniuk

Dedicamos este trabalho aos nossos pais, que nos deram todo o apoio e suporte necessários por todo o curso, aos parentes e aos amigos de faculdade e de nossas cidades.

Agradecimentos

Nossos agradecimentos a todos os que contribuíram, direta ou indiretamente, para a realização deste trabalho, em especial:

- a todos os citados na dedicatória, pelo apoio dado;
- ao nosso orientador, Prof. Dr. Daniel Varela Magalhães, pela paciência e ajuda dada neste ano;
- aos membros da banca;
- a todos que trabalham no LIEPO, pela atenção e idéias dadas;
- ao Denis Henrique Neves, técnico do LIEPO, que nos ajudou a sanar os últimos problemas do protótipo;
- à Mariana Vilela Martins, pela ajuda com a plataforma de texto LaTeX.

Resumo

Esta dissertação tem como objetivo delinear os métodos para a criação de uma interface de controle do sintetizador digital de frequência utilizado no laboratório do CePOF (Centro de Pesquisa em Óptica e Fotônica) para geração de sinal de referência de um relógio atômico tipo Fountain. Partindo de uma placa de avaliação do sintetizador controlada por USB em um software do fabricante (*Analog Devices*), foi sugerida pelo professor orientador a criação de um sistema que utilizasse apenas o circuito integrado AD9912.

Com isto, espera-se reduzir os custos de equipamentos, além de diminuir o espaço ocupado pelos mesmos. Adicionalmente, será possível criar o controle desta interface no mesmo ambiente de desenvolvimento que os sistemas do relógio atômico utilizam, desenvolvidos em LabVIEW®, facilitando assim programação e o desenvolvimento do novo sistema.

Abstract

This monography will show and explain the methods employed in the development of a control interface to be used in the direct digital frequency synthesizer employed in CePOF's (Optics and Photonics Research Center - University of São Paulo) fountain-type atomic clock as a reference signal generator. Using an USB-controlled evaluation board made by Analog Devices of a direct digital synthesizer (DDS) as a starting point, our goal is the development of a system that could use only the integrated DDS circuit AD9912 instead of the entire board. Expected results are the reduction of equipment costs and space usage by such devices. Additionally, a user interface for controlling the device will be created in the same development environment currently employed in the atomic clock systems (LabVIEW®) in order to make a versatile and integrated solution.

Sumário

Lista de Tabelas

Lista de Figuras

CAPÍTULO 1: Introdução

A definição universal de segundo passou por diversas mudanças, desde 1/86.400 de um dia solar médio até a fração de 1/31.556.925,9747 do ano solar a partir do dia 0 de janeiro de 1900 às 12 horas.[1]Com o surgimento de métodos mais precisos de medição, graças aos relógios atômicos, desde 1967 a unidade de tempo segundo é definida da seguinte forma:

> ”O segundo é a duração de 9 192 631 770 períodos da radiação correspondente à transição entre os dois níveis hiperfinos do estado fundamental do átomo de césio 133.” [INMETRO 2001, p. 22].

O relógio atômico da USP, do tipo *fountain*, é realizado em conjunto pelo Instituto de Física de São Carlos (IFSC) e pela Escola de Engenharia de São Carlos (EESC). Este tipo de relógio permite um erro provável de 1 segundo a cada 3 bilhões de anos. Para que isto seja possível, raios laser resfriam a até quase zero Kelvin átomos de Césio 133 e assim diminuem drasticamente a sua velocidade, formando uma nuvem de átomos que é lançada então na vertical – levando ao nome “chafariz de átomos”. Estes átomos passam por uma câmara de micro-ondas, que pode ou não fazê-los alterar seu estado atômico. Os átomos que têm o estado atômico alterado emitem fótons e estes são medidos por um detector.

Sintoniza-se o sinal de microondas visando a maximização da quantia de átomos que têm o estado atômico alterado e, portanto, a luminosidade medida pelo detector. Esta é frequência de ressonância do Césio, 9.192.631.770 Hz. Assim, com a exata frequência sintonizada, e utilizando a definição dada anteriormente, é possível medir com grande exatidão o segundo.

O relógio tipo *fountain* é o segundo passo na evolução do projeto, que se iniciou em 1996 com um relógio do tipo térmico linear. O próximo passo é a diminuição do tamanho do atual relógio atômico. Tendo isto em mente, a proposta deste trabalho é contribuir nesta miniaturização com o desenvolvimento de um controle em LabVIEW®para o sintetizador digital direto de frequência AD 9912 produzido pela Analog Devices.

Um sintetizador digital direto, como o nome já diz, sintetiza frequências de maneira digital a partir de uma frequência fixa de referência de *clock*. Como a maioria das operações de um DDS (sigla para *Direct Digital Synthesizer*) são digitais existem vantagens com o tamanho físico, que é menor do que um sintetizador analógico, com a resolução, geralmente fina, com o controle de fase e com os ruídos.

Esta proposta visa acabar com a necessidade de gasto de dinheiro e de espaço utilizado pelos sintetizadores comerciais e também visa a não-utilização dos kits de desenvolvimento da *Analog Devices*. Estes kits permitem um controle simples e rápido dos chips,

[1]0 de janeiro de um ano é o dia anterior a 1º de janeiro deste, ou seja, 31 de dezembro do ano anterior.

tanto através de um programa próprio que reconhece a placa via conexão USB com o computador quanto por controle direto do chip, fazendo um *bypass* da parte da placa responsável pela comunicação USB. A primeira opção não permite programação no LabVIEW®, algo desejável tendo em vista que todo o controle do relógio se dá neste ambiente de programação. A segunda opção, que se resume em utilizar apenas o chip presente na placa de desenvolvimento, se inviabiliza pelo custo – uma placa de kit de desenvolvimento chega a custar US$ 500,00 (quinhentos dólares) enquanto o chip isoladamente não passa de US$ 70,00 (setenta dólares).

Uma outra forma de controle do AD9912 se dá pelo uso de comunicação SPI, que permite a gravação de valores nos registradores do DDS. Isto pode ser feito através de duas ou três portas lógicas, sendo que uma é usada para ditar a frequência em que será trocada a informação (*master clock*) e as outras são usadas como dutos de dados. Para que isto fosse implementado foram feitos testes com dois tipos de circuitos integrados neste projeto, uma para comunicação por porta Serial e outra por comunicação via porta USB. No final, decidimos pelo uso de um microcontrolador (PIC, ou Programmable Integrated Circuit) da fabricante Microchip com comunicação Serial RS-232. Uma solução de baixo nível (mais próxima do hardware) como esta permite uma plataforma mais flexível, além de ser menos custosa e permitir mais fácil adaptação a novas necessidades que poderão surgir com o tempo (como controle de amplitude e fase).

CAPÍTULO 2: Conceitos

Esta capítulo se dedica a explicar de maneira sucinta alguns dos conceitos mais importantes utilizados neste trabalho.

2.1 Síntese digital direta

A sítese digital direta consiste em sintetizar uma forma de onda analógica de maneira a gerar um sinal digital oscilante a partir de uma frequência fixa de referência de *clock* e depois fazer uma conversão digital-analógica. O DDS gera uma função (geralmente seno) utilizando *samples* armazenados em sua memória. A cada pulso do *clock* o *sample* correto é escolhido da memória e então gerado. A figura 2.1 exemplifica a síntese de um sinal de 21 MHz com um clock de 100 MHz:

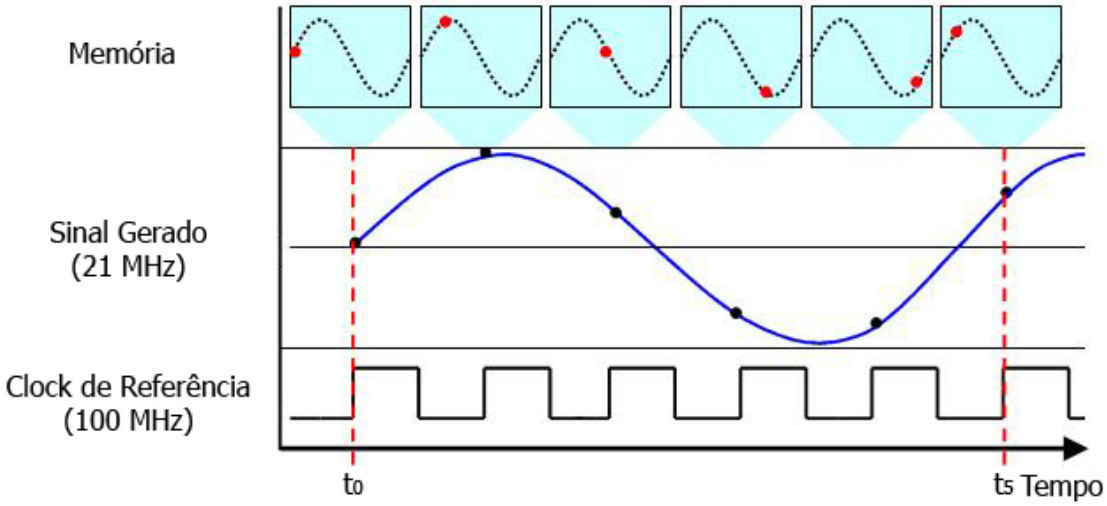


Figura 2.1: Síntese de sinal de 21 MHz a partir de um clock de 100 MHz.

A resolução da frequência gerada depende diretamente da palavra binária de definição de frequência, a FTW (sigla para *Frequency Tuning Word*). Esta palavra varia de tamanho de acordo com o chip utilizado, geralmente entre 24 e 48 bits e define a frequência gerada de acordo com a equação 2.1, onde N é o número de bits da FTW:

$$f_{OUT} = \begin{cases} \frac{FTW}{2^N} \cdot f_{SYSCLK}, \text{ se } 0 \leq f_{OUT} < 2^{N-1} \\ (1 - \frac{FTW}{2^N}) \cdot f_{SYSCLK}, \text{ se } f_{OUT} \geq 2^{N-1} \end{cases} \tag{2.1}$$

A fase do sinal DDS pode ser digitalmente controlada pela *phase offset word* (POW). O ângulo relativo de *offset* $\Delta\theta$ pode ser definido pela equação 2.2, sendo M o número de

bits da POW:

$$\Delta\theta = \begin{cases} 2\pi\frac{POW}{2^M}\ (rad) \\ 360\frac{POW}{2^M}\ (graus) \end{cases} \tag{2.2}$$

Por fim, em alguns DDS, a amplitude relativa pode ser escalada digitalmente por um fator de escala de amplitude, ASF (sigla para *Amplitude Scale Factor*). Este fator pode definido pela equação 2.3, sendo K o número de bits do ASF.

$$Amplitude\ Scale = \begin{cases} \frac{ASF}{2^K}\ (\text{fração}) \\ 20\log(\frac{ASF}{2^K})\ (dB) \end{cases} \tag{2.3}$$

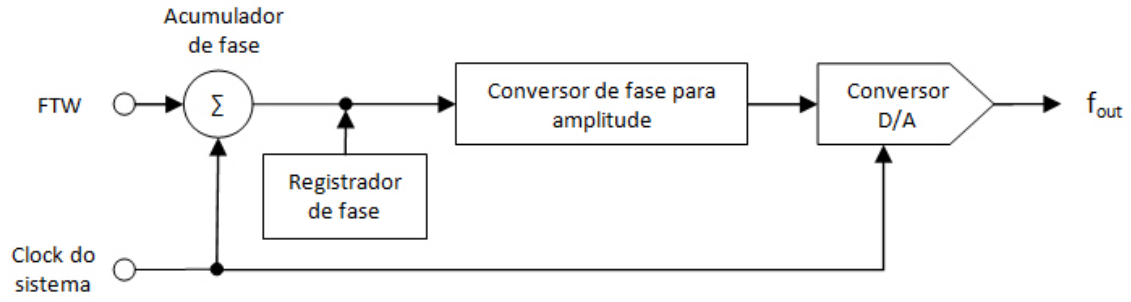


Figura 2.2: Diagrama básico de um DDS.

A figura 2.2 mostra de maneira básica como funciona a geração de uma frequência de saída através de um DDS: a FTW computa um salto angular a cada período do clock e esses saltos são adicionados no acumulador de fase. O conversor de fase para amplitude utilizará uma tabela para converter esta fase em um valor digital de amplitude, no caso o seno do ângulo. E por fim, o conversor D/A transformará esse sinal digital em um valor analógico de tensão ou corrente.

A figura 2.3 mostra também de maneira básica as alterações do sinal através do DDS: conforme já vimos, a FTW computa um salto angular a cada período do clock e esses saltos são adicionados no acumulador de fase, gerando uma rampa digital. Esta rampa é convertida para o seno do ângulo e por fim, o conversor D/A transforma esse sinal digital em um valor analógico.

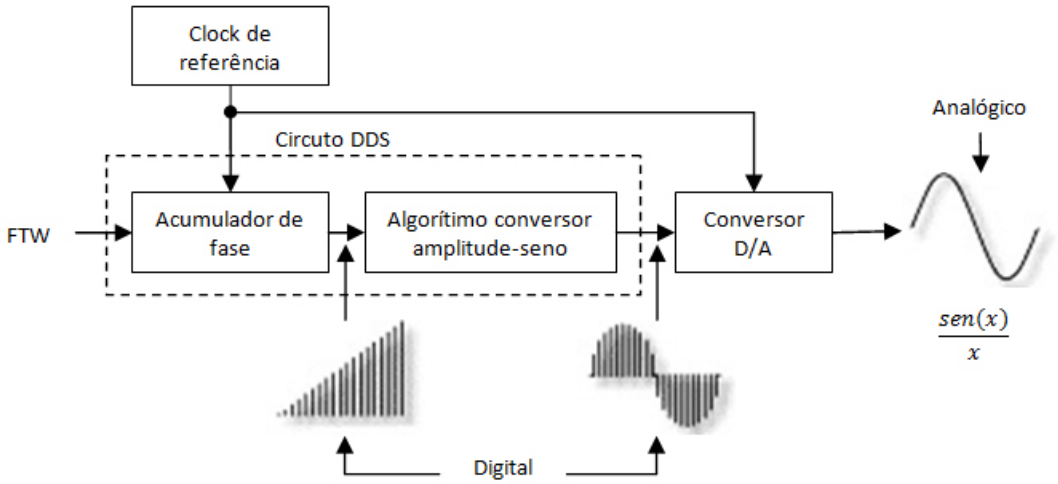


Figura 2.3: Sinal através de um DDS.

2.2 Comunicação SPI

SPI bus, ou *Serial Peripheral Interface bus*, é um padrão de comunicação serial síncrono criado pela Motorola que opera em *full-duplex* – o que implica que a comunicação pode ser feita em duas direções, assim como no *half-duplex*, mas diferente deste no *full-duplex* ela ocorre ao mesmo tempo.

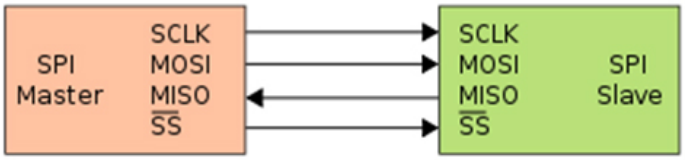


Figura 2.4: Comunicação entre dispositivo SPI Master e SPI Slave.

A figura 2.4 mostra o esquema típico entre um dispositivo atuando como SPI Master e um dispositivo atuando como SPI Slave, o que significa que o Master controlará o Slave. As setas indicam o sentido do fluxo de dados. Podemos ver na imagem os quatro fios da comunicação SPI, que são:

- **SCLK:** *Serial Clock*, também visto como SCK ou CLK;
- **MOSI:** *Master Output – Slave Input*, onde os dados são enviados pelo Master;
- **MISO:** *Master Input – Slave Output*, onde os dados são enviados pelo Slave;

- **$\overline{\text{SS}}$**: *Slave Select*, utilizado para permitir que o Slave transmita dados para o Master e também receba dados do Master. É ativo em nível lógico baixo e permite que apenas um Master controle mais do que um Slave. Algumas vezes visto também como $\overline{\text{CS}}$ (*Chip Select*).

Alternativamente podemos chamar as duas linhas de transmissão de dados(MOSI e MISO) de um dispositivo SPI como:

- **SDI:** *Serial Data In*, onde os dados são recebidos pelo dispositivo. É o MISO de um SPI Master e o MOSI de um SPI Slave;
- **SDO:** *Serial Data Out*, onde os dados são enviados pelo dispositivo. É o MOSI de um SPI Master e o MISO de um SPI Slave.

Esta nomenclatura alternativa, que é usada em muitos chips e aplicações, pode ser vista na figura 2.5, que mostra também como o uso do *chip select* permite que com apenas um *Master* se controle mais de um *Slave*, conforme já citado anteriormente. Para isso, é necessário que o *SPI Master* tenha mais do que um *chip select*, o que é possível através da programação de um PIC, por exemplo. Já as vias de dados SCLK, MOSI e MISO são as mesmas para todos os *Slaves*, deixando a programação dos *chip selects* para definir qual *Slave* receberá e enviará dados (ou quais, embora isso seja pouco usual).

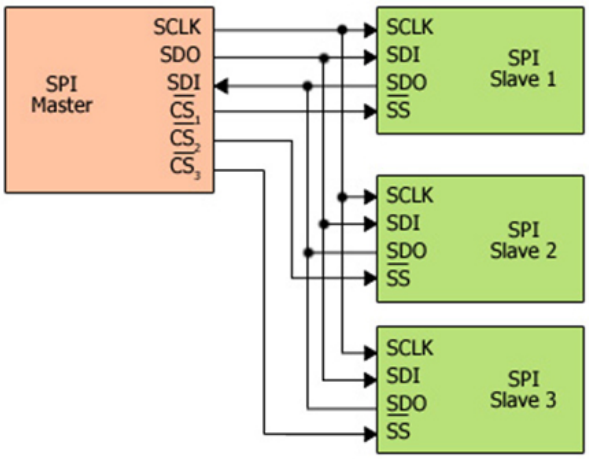


Figura 2.5: Comunicação entre um dispositivo SPI Master e três dispositivos SPI Slave através do uso de chip select.

É possível também usar os *Slaves* de maneira cooperativa, em configuração conhecida por *Daisy chain*. Neste tipo de configuração há apenas um *chip select* no Master e o MISO de cada Slave é ligado no MOSI do Slave seguinte. Sendo N grupos de pulso de clock, os MISOs neste caso devem ser programados para enviarem no Nº grupo de pulsos exatamente o que o *Slave* recebeu via MISO no (N-1)º grupo de pulsos. Apenas alguns produtos e aplicações suportam e/ou requerem tal configuração, sendo pouco usual.

A transmissão de dados utlizando protocolo SPI é síncrona, isto é, uma comunicação entre um transmissor e um receptor com clocks de referência sincronizados. Assim, o reconhecimento dos dados transmitidos se dá na borda de subida (alteração de nível baixo para alto) ou na borda de descida (alteração de nível alto para baixo) do clock.

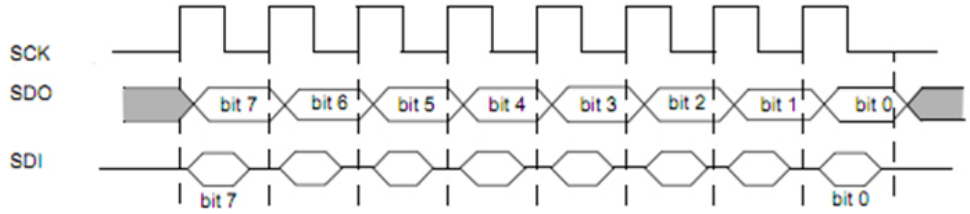


Figura 2.6: Transmissão e recepção de dados síncrona com reconhecimento na borda de subida.

A figura 2.6 mostra de maneira simplificada o conceito de a transmissão e recepção de dados síncrona no caso de reconhecimento na borda de subida. O comportamento é análogo para reconhecimento na borda de descida.

2.3 Phase-Locked Loop

A sigla PLL atende por *Phase-Locked Loop*, um modo de controle que permite a um clock externo de 40 MHz, por exemplo, gerar um sinal de 400 MHz. Um Phase-locked Loop (PLL) é um sistema de controle que gera um sinal de saída cuja fase está relacionada à de seu sinal de entrada, através do uso de um gerador variável de frequência e um detector de fase. Existem muitos tipos de PLL, sendo que no caso do AD9912 é utilizado o PLL Linear, que utiliza um Oscilador Controlado por Voltagem (Voltage-controlled Oscillator, ou VCO) e um multiplicador analógico.
Utilizando a frequência do sinal de entrada (controlada pelo usuário) e um VCO de maior frequência, o PLL pode multiplicar a frequência de entrada por um valor determinado via software e assim tornar possível o funcionamento de um DDS com maior flexibilidade.

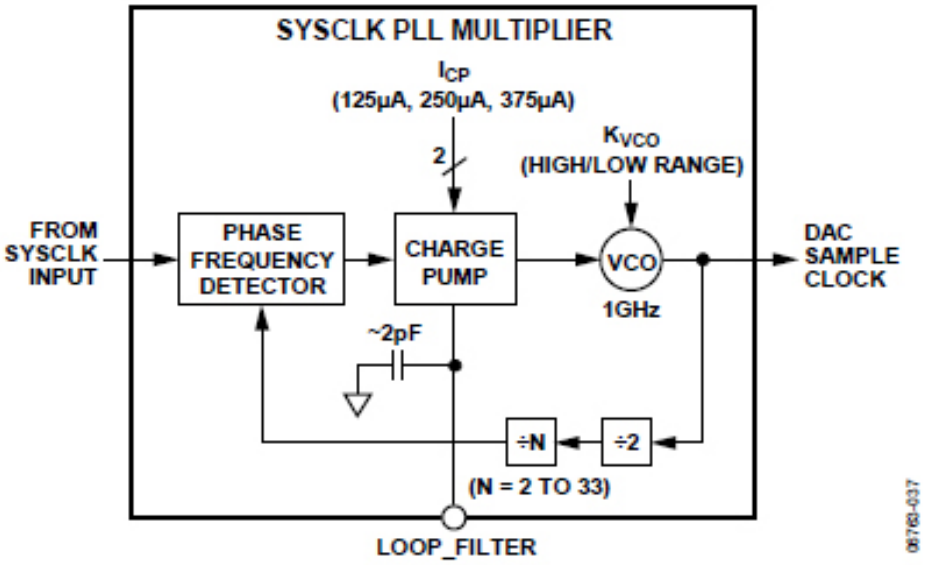


Figura 2.7: Diagrama de blocos do multiplicador PLL do AD9912.

O circuito integrado AD9912 possui dois multiplicadores em seu PLL – o primeiro pode duplicar a frequência de entrada, enquanto o segundo utiliza-se da saída do primeiro para multiplicá-la por um valor que vai de 2 a 33 (sendo que estes multiplicadores pode ser desligados, caso seja desejável uma saída de frequência igual à da entrada). Para multiplicar inicialmente a frequência por 2 o circuito detecta cada borda de subida e descida do sinal de entrada de frequência, gerando um pulso completo a cada detecção. É então feita uma realimentação do sinal de saída a um Detector de Fase/Frequência (Phase/Frequency Detector, ou PFD), e através de uma Charge Pump controlada via software a frequência do VCO é ajustada para multiplicar a frequência de saída novamente. Com isto, temos uma saída final de frequência multiplicada por valores que vão de 2 a 33 e acima disto, valores pares de 34 até 66.

CAPÍTULO 3: Materiais e métodos

Este capítulo se dedica a detalhar cada material e procedimento experimental utilizado neste trabalho. Os resultados dos procedimentos experimentais estão no capítulo 4.

3.1 Chip AD-9910

Este chip é um DDS fabricado pela Analog Devices. Entre suas características principais estão:

- 1 GSPS (sigla para *Giga-Samples Per Second*) de clock interno;
- Conversor Digital-Analógico (DAC) de 14 bits;
- Até 400 MHz de *output* analógico;
- Sua FTW possui 32 bits, o que permite resolução de frequência de no mínimo 0,23 Hz;
- Sua POW tem 14 bits e sua ASF tem 16 bits;
- Comunicação SPI.

Assim, das equações 2.1, 2.2 e 2.3 apresentadas na seção 2.1, temos:

$$f_{OUT} = \begin{cases} \frac{FTW}{2^{32}} \cdot f_{SYSCLK}, \text{ se } 0 \leq f_{OUT} < 2^{31} \\ (1 - \frac{FTW}{2^{32}}) \cdot f_{SYSCLK}, \text{ se } f_{OUT} \geq 2^{31} \end{cases} \tag{3.1}$$

$$\Delta\theta = \begin{cases} 2\pi\frac{POW}{2^{14}} \ (rad) \\ 360\frac{POW}{2^{14}} \ (graus) \end{cases} \tag{3.2}$$

$$Amplitude\ Scale = \begin{cases} \frac{ASF}{2^{16}} \text{ (fração)} \\ 20\log(\frac{ASF}{2^{16}}) \ (dB) \end{cases} \tag{3.3}$$

Neste trabalho utilizamos o AD 9910 em seu kit de desenvolvimento, chamado de AD 9910/PCBZ. Tais kits de desenvolvimento consistem de uma placa desenvolvida pela Analog Devices que permite controle do chip via USB através de um software dedicado. O uso deste kit de desenvolvimento se deu apenas para aprendizado básico das funções e do controle de um DDS a partir do trabalho de conclusão de curso *Diagnóstico da Placa AD9910 – Analog Devices*. [Carvalho e Pereira 2010]

3.1.1 Testes com a placa AD9910/PCBZ

Inicialmente repetimos os testes feitos no supracitado trabalho *Diagnóstico da Placa AD9910* com a finalidade de aprendizado, através do software dedicado da AD9910/PCBZ, que pode ser visto na imagem 3.1. Foram feitos os seguintes testes:

1. Verificação da consistência do sinal gerado pelo SMB 100A Signal Generator e da análise desse sinal pelo MXA Signal Analyzer, ligando diretamente o gerador de sinal no analisador de espectro;

2. Documentação do espelhamento do sinal que ocorre em $f_{CLOCK} \pm f_{OUT}$ e se repete em intervalo igual ao do clock. Para este teste são configurados no software:

   - *External Clock*, que é o sinal gerado pelo SMB 100A;
   - *System Clock*, que é o sinal utilizado pelo chip e pode ser igual ao External Clock ou metade dele;
   - *Output Frequency*, que é a frequência de saída desejada;
   - *DAC Gain Control*, que controla o ganho de corrente do DAC;
   - *I/O Sync Clock Output Pin*, que controla o uso do driver para a saída do clock síncrono.

3. Controle da amplitude através da ASF;

4. Utilização de sinal de saída filtrado. Para este teste comparamos o resultado da ligação do SMB 100A Signal Generator na entrada de clock de referência J1 e a saída não filtrada J3 da AD9910/PCBZ no MXA Signal Analyzer com o resultado de ligar a saída filtrada J4 da AD9910/PCBZ no MXA Signal Analyzer;

5. Teste do uso do multiplicador de frequência. Este teste exige configurações diferentes dos anteriores: Continuamos utilizando a saída filtrada J4 testa anteriormente, porém o clock externo é colado em um valor baixo; o divisor de frequência é desligado e o multiplicador é ligado;

6. Verificação do modo RAM, onde programa-se perfis de FTW, ASF e POW previamente em um arquivo txt que é lido pelo software para a RAM interna do chip. Testamos dois modos:

   - Modo Direct Switch: onde o chip sintetiza diretamente a frequência da FTW que está no começo do arquivo;
   - Modo Ramp Up: onde o chip sintetiza as frequências de cada uma das FTWs que está no arquivo e após isso se fixa na última.

Nos dois casos a FTW é ajustada tendo como base a razão entre a frequência que queremos como saída (Fdds) e a frequência que usamos na entrada do AD9912 (Fs), expressa em 2^{32} unidades de resolução. A equação que representa isto é a parte superior da equação 3.1, onde $0 \leq f_{OUT} < 2^{31}$

7. Amplitude, frequência e potência, onde configuramos o clock externo em uma determinada frequência com nível de 5,00 dBm, o system clock (utilizando ou não a opção */2 Divider*), a frequência de saída no *Profile 0*, *I/O Sync Clock Output Pin* habilitado e variamos o DAC Gain Control e o ASF. Isso com o objetivo de verificar a existência de uma relação entre o valor de corrente que controlamos através do software (I_{OUT}) e o valor de corrente medido na saída (I_{SAIDA}).

Os resultados de todos estes testes sao apresentados na seção 4.1.

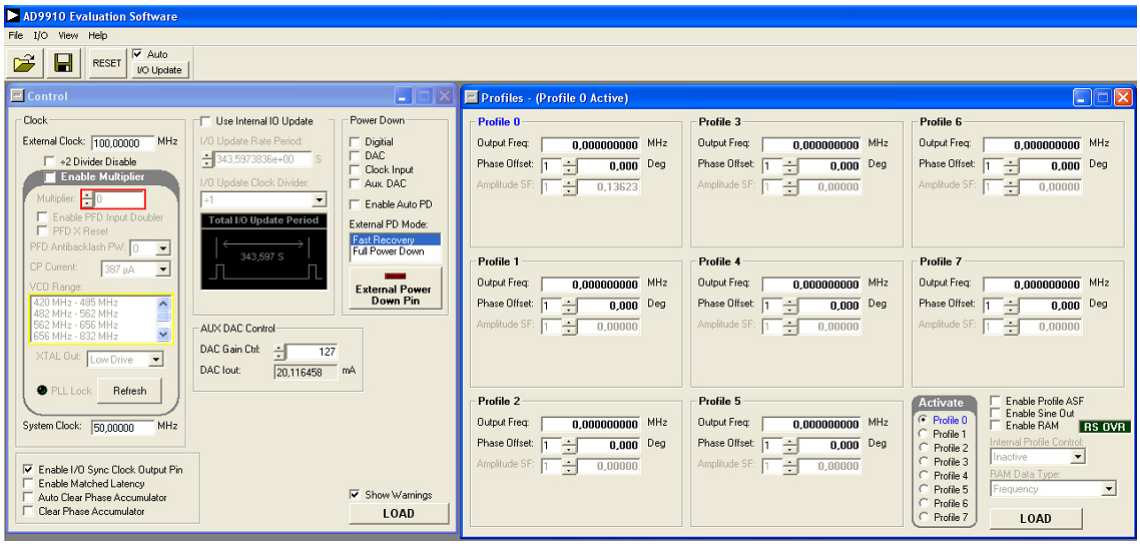


Figura 3.1: Tela inicial do software da placa AD9910/PCBZ.

3.1.2 Single Tone

Dados os resultados dos testes de amplitude, frequência e potência (seção 4.1.7), nos concentramos em controlar a frequência via porta serial sem o uso do programa específico da placa. Decidimos iniciar pelo modo de operação aparentemente mais simples: o Single Tone.

Neste modo os controles vem diretamente dos registradores de programação, tendo oito perfis diferentes de programação possíveis para tal (pinos 52 a 54 do chip). Uma mudança de estado nos pinos de controle de perfil com a próxima borda de subida do SYNC_CLK faz *update* no DDS com os parâmetros do perfil. Dados o entendimento de como a frequência de saída, o ângulo relativo de offset e a amplitude relativa funcionam

INPUTS	OUTPUTS	INPUT/OUTPUTs
2 – PLL_LOOP_FILTER	19 – PLL_LOCK	59 – I/O_UPDATE
14 – MASTER_RESET	55 – SYNC_CLK	67 – SDIO
18 – EXT_PWR_DWN	68 – SDO	
54:52 – PROFILESx	80 – $\overline{\text{IOUT}}$	
60 – OSK	81 – IOUT	
69 – SCLK	84 – DAC_RSET	
70 – $\overline{\text{CS}}$	94 – REFCLK_OUT	
71 – I/O_RESET		
90 – REF_CLK		
91 – $\overline{\text{REF\_CLK}}$		
95 – XTAL_SEL		

Tabela 3.1: Pinos referentes ao modo Single Tone.

(respectivamente equações 3.1, 3.2 e 3.3), selecionamos os pinos que possuem alguma relação inicial com o modo Single Tone. Estes pinos são mostrados na tabela 3.1.

Como utilizaremos inicialmente o LabVIEW®, encontramos em sua versão 8.5 um arquivo *.vi* pronto com comunicação de porta serial básica, permitindo um teste inicial de comunicação (figura 3.2). Iniciamos contruindo uma placa de conversão RS232 (protocolo de dados utilizado por portas seriais, com nível de tensão de 12V) para TTL (com nível de tensão 5V). Esquema da placa na figura 3.3 e resultados na seção 4.2.

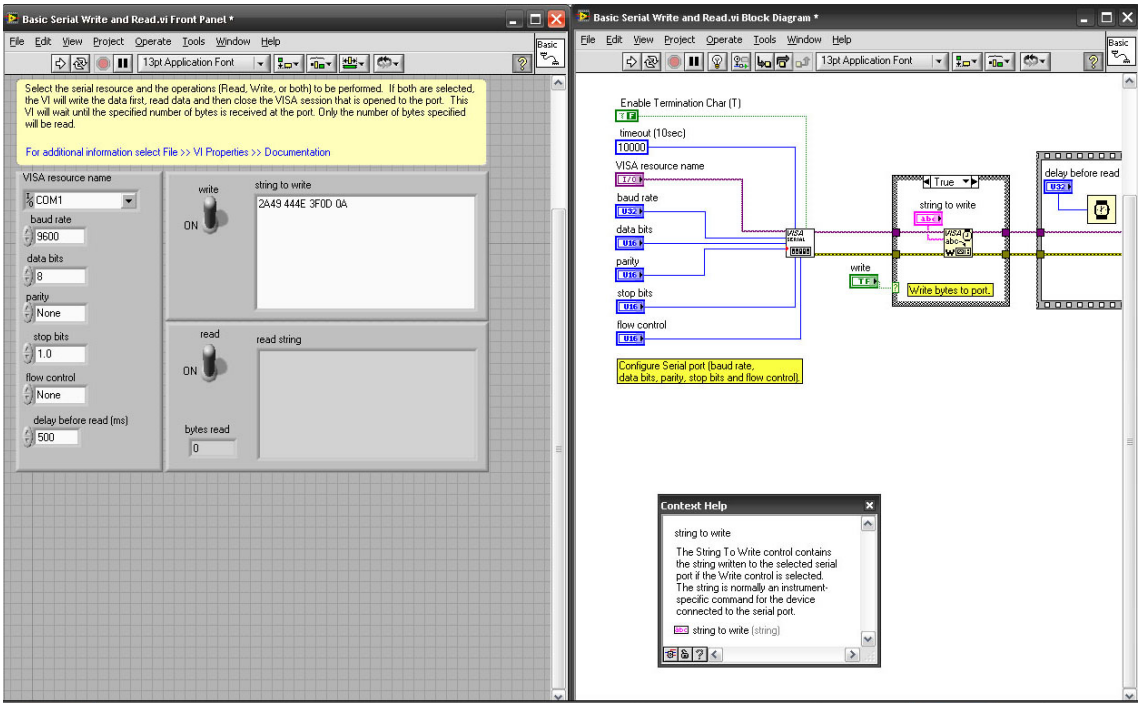


Figura 3.2: VI com comunicação serial básica.

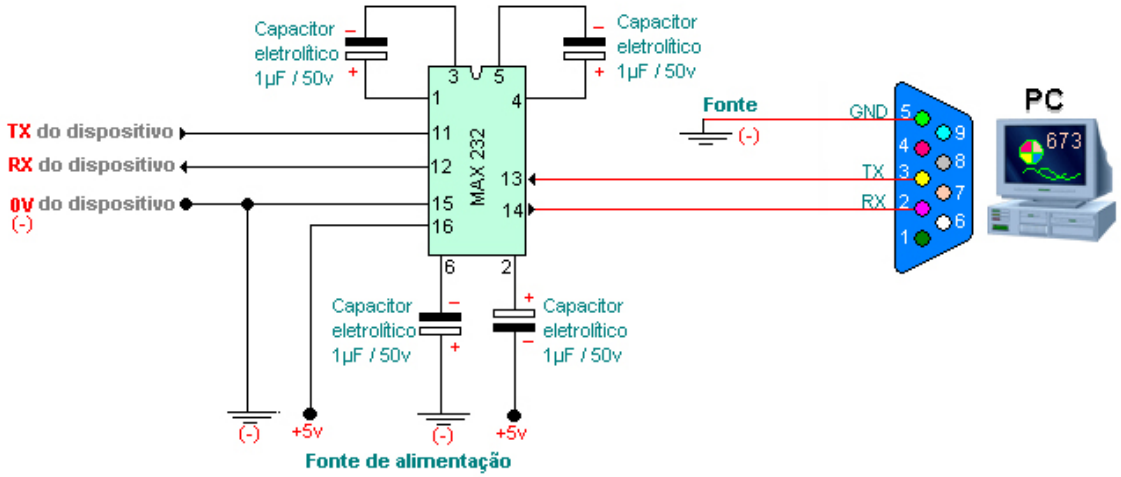


Figura 3.3: Esquema da placa de conversão DB9-RS232.

3.2 Chip AD-9912

Este chip também é um DDS fabricado pela Analog Devices. Entre suas características principais estão:

- 1 GSPS de clock interno;
- DAC de 14 bits;
- Até 400 MHz de *output* analógico;
- Sua FTW possui 48 bits, o que permite resolução de frequência de no mínimo 40μHz;
- Sua POW tem 14 bits;
- Não possui ASF, porém possui um controle de escala de corrente, a FSC, de 10 bits;
- Comunicação SPI.

Este chip é o objeto principal deste trabalho. Foi escolhido devido à sua principal vantagem em relação ao AD 9910, a resolução de frequência conseguida graças aos bits de sua FTW. Esta resolução é essencial quando se deve chegar o mais próximo possível dos 9.192.631.770 Hz, tendo em mente que como esta frequência é cerca de 200 vezes maior que o *output* do DDS é necessário que este *output* seja ainda multiplicado. Adaptando a equação para frequência 2.1 de saída apresentada anteriormente, temos: 2.2 e 2.3 apresentadas na seção 2.1, temos:

$$f_{OUT} = \frac{FTW}{2^{48}} \cdot f_{SYSCLK} \tag{3.4}$$

A equação para o caso $f_{OUT} \geq 2^{N-1}$ não se aplica no AD 9912. Para o offset $\Delta\theta$ temos:

$$\Delta\theta = \begin{cases} 2\pi\frac{POW}{2^{14}} \ (rad) \\ 360\frac{POW}{2^{14}} \ (graus) \end{cases} \tag{3.5}$$

Como toda vantagem acarreta uma desvantagem, o ganho em precisão da frequência acarreta que não há um fator de escala de amplitude tal qual o do AD 9910. Tal controle necessitaria ser feito externamente através de amplificadores, por exemplo, o que foge do escopo deste trabalho que é o de controlar a frequência de saída do AD 9912. O que o AD 9912 possui é uma palavra de 10 bits, a FSC (sigla para *Full-Scale Current*), que serve parcialmente como um fator de escala de corrente. A corrente de saída do DAC depende de dois fatores: a corrente de referência I_{DAC_REF} e do valor de FSC. Este valor de é definido ao conectar–se um resistor R_{DAC_REF} entre o pino DAC_RSET e o terra. O pino DAC_RSET é internamente conectado a uma tensão de referência de 1.2V, o que nos leva à equação 3.6.

$$I_{DAC\_REF} = \frac{1,2}{R_{DAC\_REF}} \tag{3.6}$$

De posse do valor de FSC e de I_{DAC_REF} calcula-se a corrente de saída I_{DAC_FS}:

$$I_{DAC\_FS} = I_{DAC\_REF}(72 + \frac{192 \cdot FSC}{1024}) \tag{3.7}$$

Assim, considerando o valor recomendado de 120μA para I_{DAC_REF}, a corrente de saída I_{DAC_FS} ficará entre 8,6mA e 31,7 mA. Para fazer o controle utilizamos a comunicação SPI natural do chip, fazendo um bypass da parte do kit responsável pela comunicação USB. A placa pode ser vista na figura 3.4, com uma região destacada em verde à esquerda mostrando a parte responsável pela comunicação USB. Embora esta região pareça pequena, um olhar mais atento nos mostra que o restante da placa é composto basicamente de pinos de acesso ao chip e entradas/saídas de sinal. Estas são explicitadas na tabelas 3.2 e 3.3.

Da análise das tabelas 3.2 e 3.3, concluímos que não seria necessário ligar nada em J4, J6, J12. No pino J10 ligamos o clock, destacado em vermelho na imagem 8. Em J16, J17, J31 e J32 ligamos 1.8 V, pinos destacados em laranja na mesma imagem. E por fim em J9 e J11, destacados em verde, ligamos 3.3V.

3.3 Chip FT2232D

Para a comunicação com os sistemas já existentes no equipamento do Relógio, pensamos em integrar uma solução em LabVIEW®que pudesse se comunicar com o AD9912. A idéia inicial foi utilizar uma conexão USB para estabelecer comunicação SPI, que segundo

Plug	Identificação na placa	Pino	Identificação no chip	Descrição no datasheet do chip
J1	OUT	35	OUT	Saída HSTL. Ver as seções “Specifications” e “Primary 1.8 V Differential HSTL Driver” para detalhes.
J2	OUTB	34	OUTB	Saída HSTL complementar.
J4	DUT_OUT / FILTER_IN	50, 51	DAC_OUT, DAC_OUTB	DAC Output e DAC Output complementar. Este sinal deve ser filtrado e enviado de volta através da entrada FDBK_IN. Estes pinos possuem um resistor interno de pull-down de 50 Ω cada.
J5	DUT_FILTER _OUT	J4, J6		
J6	FDBK_IN	40, 41	FDBK_INB, FDBK_IN	Entrada de feedback e entrada de feedback complementar. Quando usando as saídas HSTL e CMOS, FDBK_INB é conectado à saída filtrada DAC_OUTB e FDBK_IN é conectado à saída filtrada DAC_OUT. Esta entra é tipicamente usada com corrente alternada e, quando configurada como tal, aceita qualquer sinal diferencial de altura mínima de crista igual a 400mV.
J7	CMOS_OUT	38	OUT_CMOS	Saída CMOS de 3.3V. Ver as seções “Specifications”, “Output Clock Drivers” e “2x Frequency Multiplier”. Este pino torna-se CMOS de 1.8 V se o pino 37 é colocado em 1.8 V.
J9	VDDX_DRV	37	VDDX_DRV	Fonte analógica para o driver de saída CMOS. Este pino é normalmente 3.3 V mas pode ser 1.8 V. Este pino deve ser ligado mesmo que o driver CMOS não seja usado. Ver a seção “Power Supply Partitioning”

Tabela 3.2: Pinos da placa AD9912/PCBZ - parte 1

Plug	Identificação na placa	Pino	Identificação no chip	Descrição no datasheet do chip
J10	SYSCLK	27	SYSCLK	Entrada do clock de sistema. Deve ser sempre ligada em AC, exceto quando usando cristal. CMOS de 1,8 pode ser usado, mas pode introduzir spur caso o ciclo de trabalho não seja 50 %. QUando usando cristal, ligar o pino CLKMODESEL ao AVSS e conectar o cristal entre este pino e o 28.
J11	VDD_DAC3	46, 47	VDD_DAC3	Conectar a uma fonte nominal de 3.3 V.
J12	XO_PWR			
J16	VDD_DACCLK	44, 45	VDD_DACCLK	Conectar a uma fonte nominal de 1.8 V.
J17	VDD_DRV/VDD_FDBK	36	VDD_DRV	Conectar a uma fonte nominal de 1.8 V.
J31	VDD_SYSCLK	25, 26	VDD_SYSCLK	Conectar a uma fonte nominal de 1.8 V.
J32	VDD_DACDEC	53	VDD_DACDEC	Conectar a uma fonte nominal de 1.8 V.

Tabela 3.3: Pinos da placa AD9912/PCBZ - parte 2

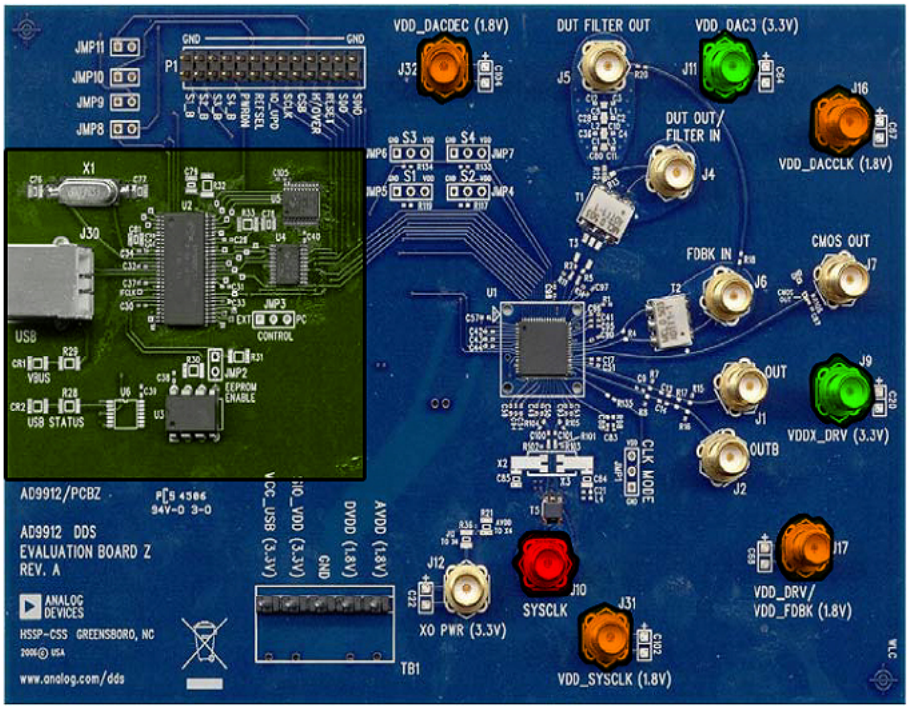


Figura 3.4: Placa do kit de desenvolvimento AD 9912/PCBZ.

o datasheet do AD9912 pode ser utilizada para programar seus registradores e assim estabelecer controle da forma desejada.

Inicialmente, utilizou-se um circuito integrado FT2232D da FTDI. Este circuito possibilita realizar conexão SPI via LabVIEW®com bibliotecas de funções já disponíveis no site do fabricante, gratuitamente e sem *royalties*. Um circuito-exemplo disponibilizado pelo fabricante foi então construído para testar a comunicação SPI na escrita e leitura de dados em uma EEPROM 93LC56B. Tal circuito pode ser visto na figura 3.5.

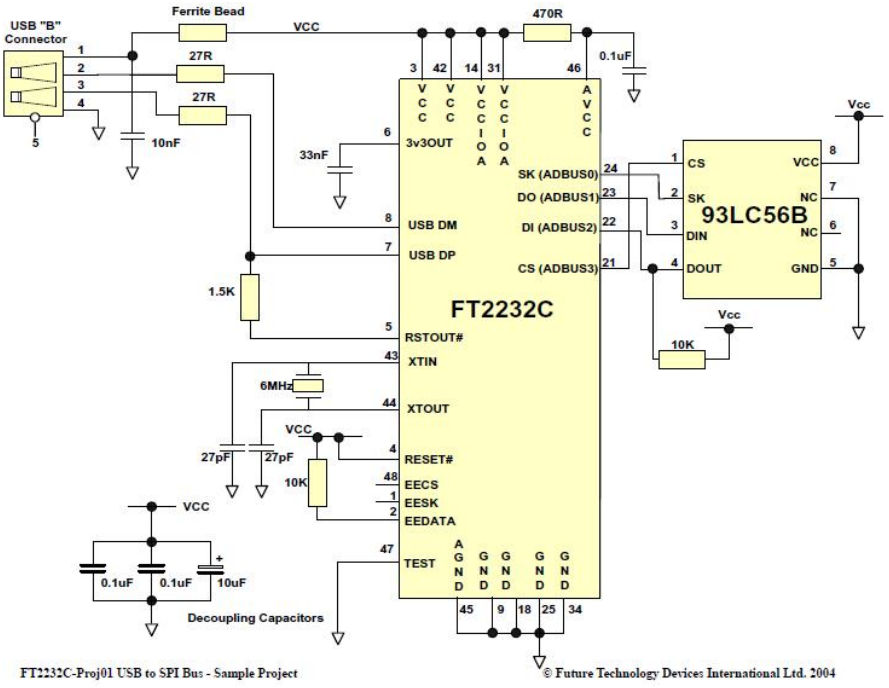


Figura 3.5: Circuito-exemplo disponibilizado pelo fabricante.

Nota-se que no circuito disponibilizado é ilustrado o FT2232C, que é a versão antiga (pré-RoHS[1]) do FT2232D, mas ambos funcionam de forma idêntica. O circuito é simples e pôde ser construído em uma *protoboard* para testes, juntamente com um programa-exemplo feito em linguagem de programação Delphi que realiza esta comunicação. Este circuito pode ser visto na figura 3.6. O programa-exemplo funcionou corretamente, gravando na EEPROM uma sequência de caracteres salva anteriormente em um arquivo de texto e lendo-a depois.

[1]Diretiva que proíbe uso de algumas substâncias perigosas em processos de fabricação de produtos.

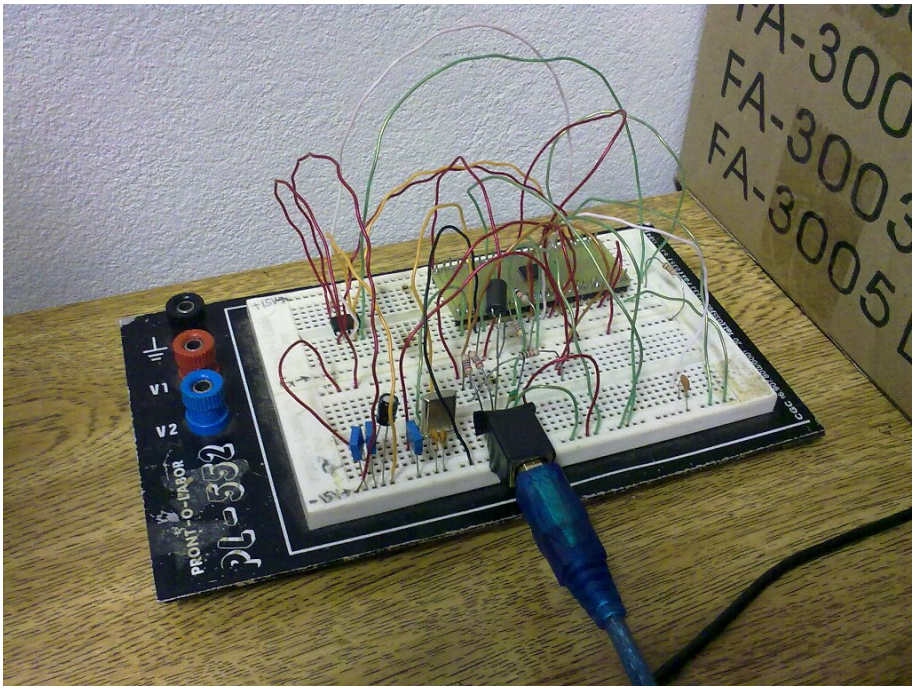


Figura 3.6: Circuito-exemplo disponibilizado pelo fabricante.

Com isto, foi possível concluir que o circuito funcionava e foi possível partir para testes em LabVIEW®. Utilizamos inicialmente seis funções básicas importadas do arquivo *FTCSPI.dll*, craado pela FTDI com o objetivo de permitir comunicação SPI através de seus chips. As funções seguem-se:

- **SPI_GetDeviceNameLocID** – Esta função retorna o nome e o identificador de localização do FT2232D. Suas entradas e saídas são:
  - *DeviceIndex* – Número do device, iniciando em 0. Normalmente a porta A do FT2232D é reconhecida no *index 0* e a porta B no *index 1*;
  - *DeviceNameBuffer* – Ponteiro para o buffer que recebe o nome do FT2232D. A porta A aparece identificada como “Dual RS232 A” e a porta B como “Dual RS232 B”;
  - *BufferSize* – Tamanho do buffer que recebe o nome do device. Definimos seu valor como 64 caracteres, respeitando o valor mínimo de 50 caracteres;
  - *LocID* – Ponteiro para uma variável tipo DWORD que recebe a localização do FT2232D conectado ao sistema;
  - *FTC_Status* – Retorna o status da função, um número inteiro entre 0 e 65 definindo se a função funcionou corretamente (código 0) ou obteve erro (códigos 1 a 64). Todas as funções importadas da dll citada anteriormente possuem uma saída FTC_Status, que funciona exatamente da mesma maneira. Logo, não serão citadas novamente.

- **SPI_GetErrorCodeString** – Esta função não é essencial, porém foi usada algumas vezes, já que recebe o código da saída FTC_Status de outra função e o transforma em uma mensagem de erro. Suas entradas e saídas são:
  - *Language* – Ponteiro para uma string que define a linguagem. Seu padrão é "EN" (inglês);
  - *StatusCode* – Recebe o código de status vindo de uma função anterior;
  - *ErrorMessageBuffer* – Ponteiro para o buffer que recebe a mensagem de erro. Retornará a mensagem de erro, que poderá ser vista através de um indicador;
  - *BufferSize* – Tamanho do buffer que recebe a string de erro. Definimos seu valor como 100 caracteres, respeitando o mínimo de 100 caracteres.
- **SPI_OpenEx** – Esta função verifica se o FT2232D está conectado e então verifica se nenhuma aplicação o está usando. Caso não exista aplicação utilizando-o no momento, a função faz uma tentativa de abrir o FT2232D.
  - *DeviceName* – Ponteiro para uma string que contém o nome do FT2232D, vindo da função SPI_GetDeviceNameLocID;
  - *LocID* – Especifica o identificador de localização do device a ser aberto, também vindo da função SPI_GetDeviceNameLocID;
  - *Handle* – Ponteiro para variável do tipo FTC_HANDLE onde a identificação para o dispositivo aberto será retornada. Esta variável deve ser usada em todas as operações subsequentes para acessar o dispositivo;
  - *FTC_Status* – Retorna o status da função.
- **SPI_InitDevice** – Esta função inicializa o FT2232D, permitindo-o ser usado daí pra frente.
  - *Handle* – Ponteiro para variável do tipo FTC_HANDLE onde a identificação para o dispositivo aberto é utilizada. Este valor vem da função SPI_OpenEx;
  - *ClockDivisor* – Divisor de frequência, sendo este um número inteiro. Calcula-se a frequência final como:

$$f = \frac{6}{1 + ClockDivisor}\ (MHz) \tag{3.8}$$

- **SPI_Read** – Esta função lê dados de um aparelho externo conectado ao FT2232D através de protocolo SPI.
  - *Handle* – ponteiro para variável do tipo FTC_HANDLE onde a identificação para o dispositivo aberto é utilizada. Este valor vem da função SPI_OpenEx.

  - *ReadStartCondition* – Ponteiro para a estrutura que contém os estados iniciais (low/high) do clock, *data out* e dos pinos *signal out*/*chip select*;
  - *ClockOutControlBitsMSBFirst* – Define a ordem em que os bits de controle são enviados: MSB primeiro (se a entrada está em estado verdadeiro) ou LSB primeiro (se a entrada está em estado falso);
  - *ClockOutControlBitsPosEdge* – Define se os bits de controle enviados são positivos (se a entrada está em verdadeiro) ou negativos (se a entrada está em falso);
  - *NumControlBitsToWrite* – Define o número de bits de controle a serem escritos para um aparelho externo. Seu valor mínimo é 2;
  - *WriteControlBuffer* – Ponteiro para o buffer que contém os dados a serem escritos;
  - *NumControlBytesToWrite* – Especifica o número de bytes de controle no buffer de controle, podendo ser de 1 a 255 bytes;
  - *ClockInControlBitsMSBFirst* – Define a ordem em que os bits de controle são lidos: MSB primeiro (se a entrada está em estado verdadeiro), LSB primeiro (se a entrada está em estado falso);
  - *ClockInControlBitsPostEdge* – Define se os bits de controle lidos são positivos (se a entrada está em verdadeiro) ou negativos (se a entrada está em falso);
  - *NumDataBitsToRead* – Número de bits a serem lidos (2 até 524280);
  - *ReadDataBuffer* – Ponteiro para o buffer que retorna os dados lidos do aparelho externo. O tamanho é fixo e deve ser 65535;
  - *NumDataBytesReturned* – Ponteiro para a variável DWORD que recebe o número de bytes realmente lidos;
  - *HighPinsReadActivateStates* – Ponteiro para GPIOH1-4;
  - *FTC_Status* – Retorna o status da função.

- **SPI_Write** – Esta função escreve dados em um aparelho externo conectado ao FT2232D através de protocolo SPI.
  - *Handle* – Ponteiro para variável do tipo FTC_HANDLE onde a identificação para o dispositivo aberto é utilizada. Este valor vem da função SPI_OpenEx;
  - *WriteStartCondition* – Ponteiro para a estrutura que contém os estados iniciais (low/high) do clock, data out e dos pinos signal out/chip select;
  - *ClockOutDataBitsMSBFirst* – Define a ordem em que os bits de dados são enviados: MSB primeiro se a entrada está em estado verdadeiro e LSB primeiro se a entrada está em estado falso;

- *ClockOutDataBitsPosEdge* – Define se os bits de dados enviados são positivos (se a entrada está em verdadeiro) ou negativos (se a entrada está em falso);
- *NumControlBitsToWrite* – Define o número de bits de controle a serem escritos para um aparelho externo. Seu valor mínimo é 2;
- *WriteControlBuffer* – Ponteiro para o buffer que contém os dados de controle a serem escritos;
- *NumControlBytesToWrite* – Especifica o número de bytes de controle no buffer de controle, podendo ser de 1 a 255 bytes;
- *WriteDataBits* – Escreve dados em um aparelho externo se o bit está em estado verdadeiro;
- *NumDataBitsToWrite* - Número de bits a serem escritos (2 até 524280 – equivalente a 64 kbytes);
- *WriteDataBuffer* – Ponteiro para o buffer que contém os dados a serem escritos;
- *NumDataBytesToWrite* – Especifica o número de bytes no buffer de gravação de dados que contém os dados que devem ser escritos no aparelho externo. Seu tamanho pode variar de 1 até 65535 bytes (64 kbytes);
- *WaitWriteDataComplete* – Ponteiro para a estrutura que determina se o FT2232D deve aguardar até todos os bytes de dados serem escritos;
- *HighPinsWriteActiveStates* – Ponteiro para a estrutura que contém qual dos quatro pinos de uso geral (GPIOH1- GPIOH4) deve ser usado durante a gravação.

- **SPI_Close** – Esta função fecha um FT2232D previamente aberto.
  - *Handle* – ponteiro para variável do tipo FTC_HANDLE onde a identificação para o dispositivo a ser fechado é utilizada. Este valor vem da função SPI_OpenEx;
  - *FTC_Status* – Retorna o status da função.

3.4 Microcontrolador PIC 16f876a

Abandonamos a idéia de comunicação USB e nos voltamos para RS232, através da porta serial. Para isso selecionamos o PIC 16f876a, que possui módulo MSSP (*Master Synchronous Serial Port*). Este módulo é uma interface serial que pode ser usada em dois modos: SPI (*Serial Peripheral Interface*) e I^2C (*Inter-Integrated Circuit*). O modo SPI permite tanto acesso em modo Slave, onde o 16f876a é controlado externamente, como Master, onde o 16f876a comanda um periférico. Como a idéia do trabalho é controlar o

AD9912, necessitamos do modo Master. Para a programação do PIC, utilizamos o software mikroC, um compilador de C criado visando programação em PICs das famílias 12, 16 e 18. Para verificar a programação sem a necessidade de reprogramar o PIC a cada tentativa ou pequena mudança na programação, utilizamos a ferramenta ISIS Schematic Capture do software Proteus, que permite importar um arquivo hex e simular o funcionamento do microprocessador. Um arquivo *hex* é um arquivo contendo código de máquina para utilização em microprocessadores, EPROMs e outros chips, geralmente gerado por um compilador de C ou C++ ou linguagem Assembly. Inicialmente testamos a comunicação RS232 do PIC, além da inicialização do modo SPI. Abaixo seque-se a programação:

```
void spi_inicio(void)
{
        SSPCON = 0b00100010;      //colisão, detecta overflow na recepção,
        //habilita sincronismo, modo master FOSC/64.
        SSPSTAT = 0b11000000;     //modo MASTER, transmite na mudança de
        //clock, bit de indicador de recepção (1 -> completo, 0 -> incompleto).
}

void main()
{

        char bytein, byteout, buffer;

        UART1_Init(9600);        // Inicializa UART a 9600 bps
        spi_inicio();        // Inicializa SPI
        Delay_ms(100);        // espera estabilizar

         while (1) {
           if (UART1_Data_Ready())        // se dados estão prontos pro envio
           {
            bytein = UART1_Read();         // ler o UART
            buffer=bytein;                //armazena o UART
            UART1_Write(buffer);          // enviar pro PC por UART
           }
        }
}
```

Este código usa a função UART1_Init para inicializar o módulo USART (*Universal Synchronous Asynchronous Receiver Transmiter*), módulo responsável pela comunicação RS232, com *baud rate* 9600. Inicializa o modo SPI através da função spi_inicio, mas não o usa. Após um período esperando estabilizar a comunicação, entra em loop infinito onde a função UART1_Data_Ready verifica se existem dados prontos para envio. Caso existam, a

função UART1_Read lê a entrada, a salva na variável *bytein* de tipo *char*. Esta variável é salva na variável *buffer*, também de tipo *char* para depois ser usada pela função UART1_Write. A função UART1_Write serve para enviar para um periférico externo os dados que estão no buffer, e por isso exige que sua variável se chame *buffer*.

Na figura 3.7 vemos o circuito simulado na ferramenta ISIS para verificação do código apresentado acima. Deve-se observar que a ferramenta já considera o PIC como alimentado e que o console que gera sinal RS232 (onde, para porta serial, o sinal é +12V para bit 1 e -12V para bit 0) já está convertido para TTL (onde o sinal é 5V para bit 1 e 0V para bit 0). Este console pode ser visto na já citada figura marcado como TX. Há também um osciloscópio com a entrada de sinal A ligada à saída TXD do console e à entrada RX do PIC e com a entrada de sinal B ligada à entrada RXD do console e à saida TX do PIC. Com essa ligação, na entrada A pudemos ver na figura 3.8 o sinal RS232 saindo do console e na entrada B o sinal saindo do PIC. O console SPI, apesar de já ligado, não tem função neste programa.

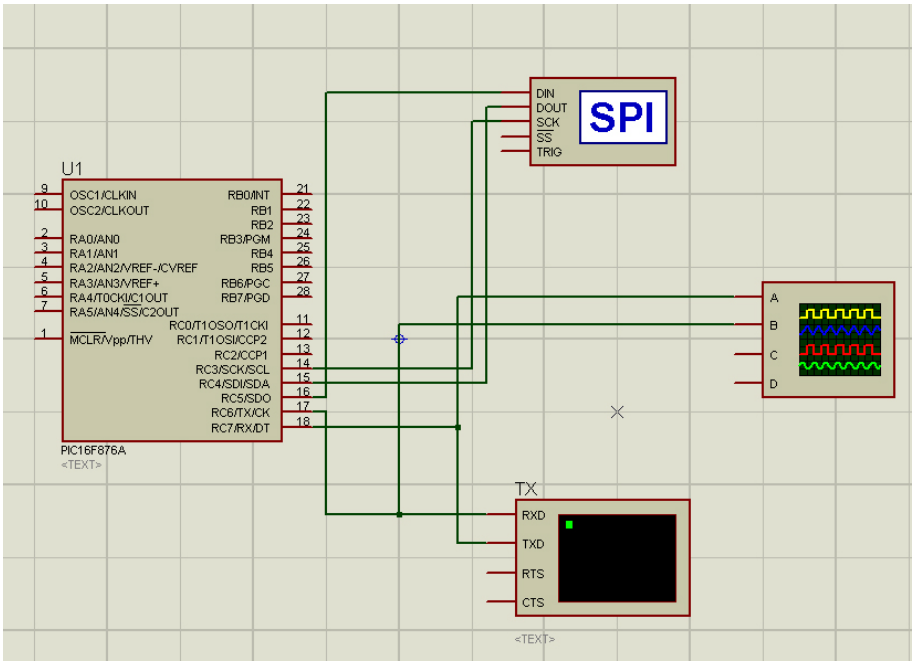


Figura 3.7: Circuito simulado na ferramenta ISIS.

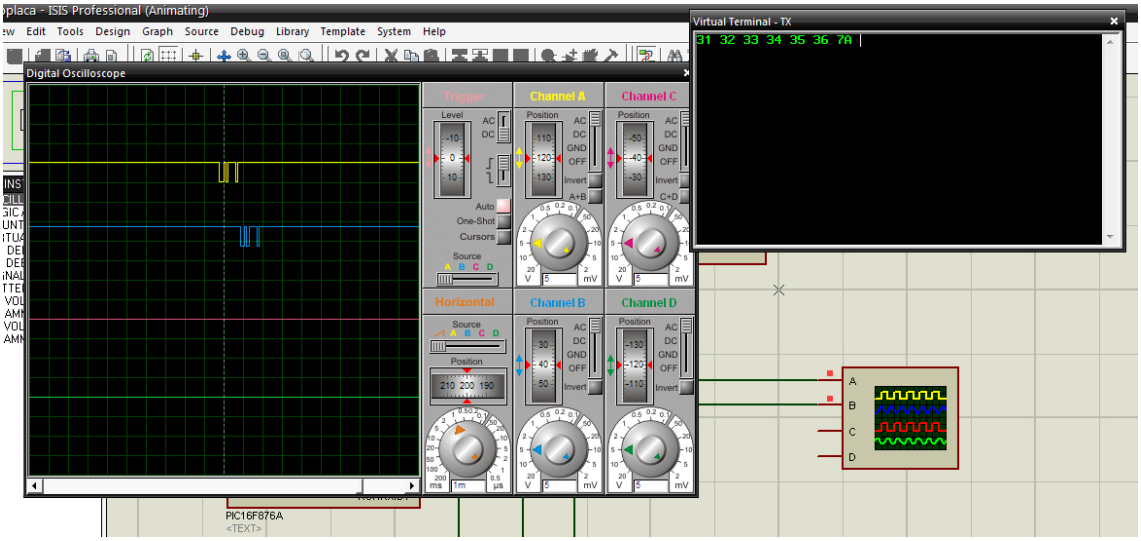


Figura 3.8: Circuito simulado na ferramenta ISIS.

Podemos ver também na figura 3.8 a tela de simulação do console enviando os valores hexadecimais 31, 32, 33, 34, 35, 36 e 7A, sinal que pode ser visto no osciloscópio em amarelo sendo enviado pelo console e em azul sendo recebido pelo console logo em seguida. Constatando o funcionamento da simulação, gravamos o programa no PIC e testamos na prática, confirmando o funcionamento prático.

O próximo passo é configurar a comunicação SPI. Embora o mesmo circuito mostrado anteriormente pudesse ser usado para este teste, adicionamos mais componentes para que ficasse mais próximo ao circuito real. Este pode ser visto na figura 3.9. Há um MAX232, responsável pela conversão de níveis de tensão RS232 pra TTL, cuja saída R1OUT está ligada a uma porta inversora *not* para obter a mesma saída que o MAX3232 utilizado no circuito real. Adicionamos ainda uma EEPROM 25LC080B para mostrar como ficaria a ligação neste caso.

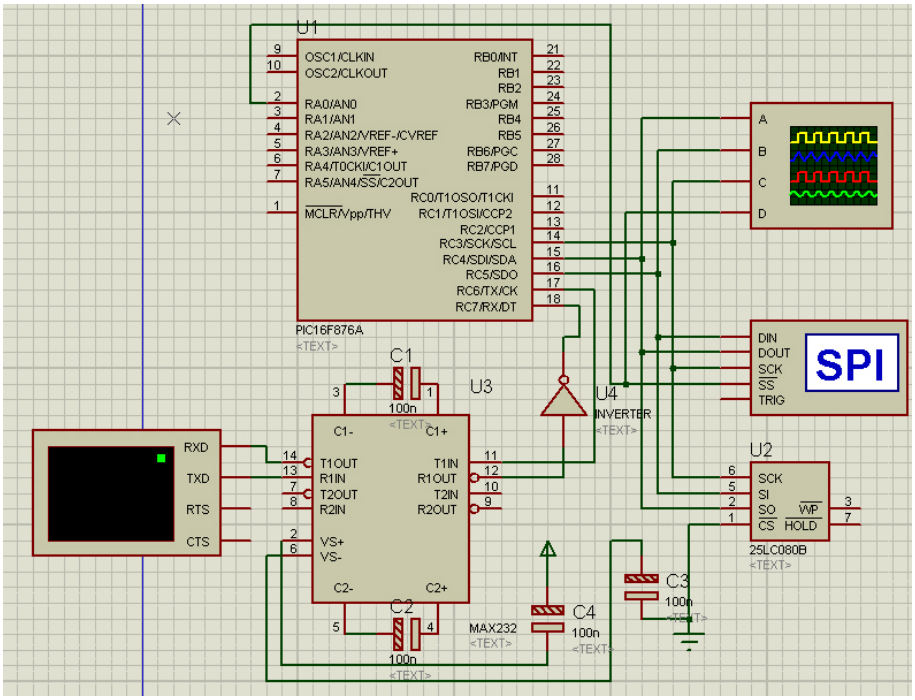


Figura 3.9: Circuito simulado na ferramenta ISIS.

A entrada A do osciloscópio está ligada na entrada SDI do PIC, que é ligada na saída SO da EEPROM. Para este teste, esta ligação não é usada. A entrada B está ligada na saída SDO do PIC, que é ligada nas entradas DIN do console SPI e SI da EEPROM. A entrada C está ligada no SCK, que é o clock da aplicação. A entrada D, por fim, está ligada na saída RA0. Nosso objetivo inicial nesse teste é gravar a Frequency Tuning Word, palavra de 48 bits responsável pela definição da frequência a ser sintetizada pelo AD9912. Por default, o pino SDIO do AD9912 atua em modo bidirecional, funcionando tanto como input quanto como output, mas preferimos utilizar o pino SDIO apenas como input. Assim, é necessário colocar o bit 0 do registrador 0x0000 em valor 1, fazendo com que o valor a ser gravado neste registrador seja 10011001. Para gravar um valor em um registrador, é necessária uma *instruction word* de 16 bits, onde o bit 15 é R/$\overline{W}$, os bits 14:13 são [W1:W0], indicando quantos bytes serão gravados e os bits 12:0 são [A12:A0], indicando o endereço do registrador que receberá o byte a ser gravado ou do registrador a receber o primeiro byte caso sejam mais do que um byte. O funcionamento dos bits [W1:W0] se dá da seguinte maneira:

W1	W2	Bytes a transmitir
0	0	1
0	1	2
1	0	3
1	1	Modo *streaming*

Tabela 3.4: Funcionamento dos bits [W1:W0].

Já a definição de qual endereço deve ser o do registrador a receber o primeiro byte depende do modo de transmissão:

- caso seja LSB, o registrador definido em [A12:A0] deve ser também o menos significativo. Por exemplo, se quisermos gravar dos registradores 0x01A6 até 0x01AB, responsáveis pela FTW, [A12:A0] deve conter o endereço 0x01A6;
- caso seja MSB, o registrador definido em [A12:A0] deve ser também o maiss significativo. Repetindo o exemplo anterior, se quisermos gravar dos registradores 0x01A6 até 0x01AB, [A12:A0] deve conter o endereço 0x01AB.

É necessário também que o pino CSB (*Chip Select Bit*) do AD9912 esteja em nível 0 para habilitar a leitura. No modo *streaming*, este pino irá definir também quando o último byte é enviado, o que exigirá que o CSB volte para nível 1. Após o envio da instruction word de 16 bits, envia-se os dados propriamente ditos. Configurado o modo unidirecional, enviamos então a FTW. Novamente coloca-se CSB em nível 0, envia-se uma instruction word definindo o registrador inicial como 0x01AB para modo de transmissão MSB, os bits [W1:W0] como 11 para modo streaming. Logo em seguida são enviados os 6 bytes que compõem a FTW e coloca-se CSB em nível 1 novamente. Segue-se abaixo o código para este teste:

```
void csb(void)
{
        PORTA=0x00;
        TRISA=0;
}

void main()
{
        int k;
        char dados[6], config[2], uni;

        uni = 0x99;
        config[0] = 0x00;
```

```
config[1] = 0x00;

UART1_Init(9600);              // Inicializa UART a 9600 bps

SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV64,
 _SPI_DATA_SAMPLE_END, _SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);
/*
Inicialização SPI
Master Clock: 11.0592MHz/64
Leitura ao fim do envio
Estado "idle" em nível lógico baixo
Transmissão na subida do nível lógico
*/

csb();

Delay\_ms(100);  // Espera 100 ms para estabilizar

PORTA.f0 = 0;
SPI1_Write(config[0]);
SPI1_Write(config[1]);
SPI1_Write(uni);
PORTA.f0 = 1;

config[0]=0b01100001;
config[1]=0b10101011;

while(1)
{
if(UART1_Data_Ready())
{
      for (k=0; k<6; k++)
      {
      while(!UART1_Data_Ready()); // Lê 48 bits de FTW
      //até encher o buffer
      dados[k]=UART1_Read();
      }


      PORTA.f0 = 0;
      SPI1_Write(config[0]); //Escreve por SPI a config. em 16 bits
```

```
                SPI1_Write(config[1]);

                for(k=0;k<6;k++)
                {
                SPI1_Write(dados[k]); // Escreve os dados do FTW
                }
                PORTA.f0 = 1;
                Delay_ms(10);

                PORTA.f0 = 0;
                SPI1_Write(0x00);           // I/O Update, pulso em 0b00000001
                SPI1_Write(0x05);
                SPI1_Write(0x01);
                PORTA.f0 = 1;

        }
        }
}
```

A função *csb* configura a inicialização da porta Port A do PIC como output, para que possamos usar o pino RA0 como gerador de sinal para o CSB do AD9912. A variável *config* de tipo *char* guarda a *instruction word*, a variável *dados* de tipo *char* guarda a FTW e a variável *uni* de tipo *char* guarda o byte de configuração do registrador 0x0000. O comando PORTA.f0 define a saída RA0. Um comando *for* lê os 48 bits da FTW usando a função UART1_read. E a função SPI1_write envia via SPI. Esta função deve ser repetida várias vezes, pois manda apenas um byte por vez que é acessada. Usando estas funções, o registrador 0x0000 é configurado, os registradores 0x01A6 até 0x01AB, responsáveis pela FTW, também e por fim é necessário um *I/O Update*, feito com um pulso no bit 0 do registrador 0x0005. As imagens 3.10 e 3.11 mostram o funcionamento do teste.

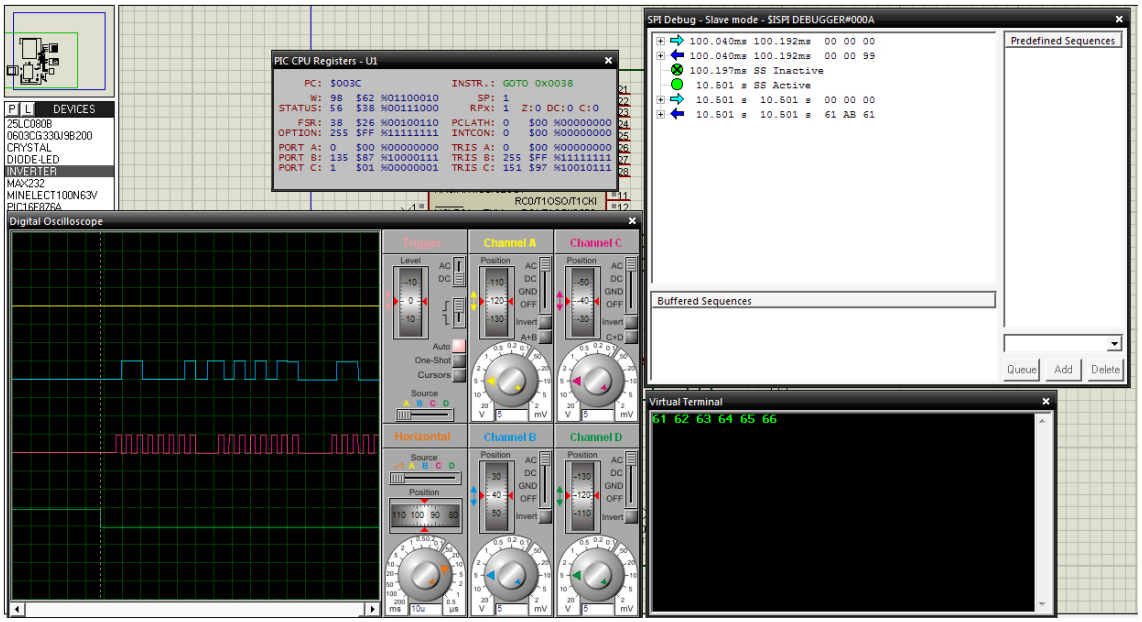


Figura 3.10: Circuito simulado na ferramenta ISIS.

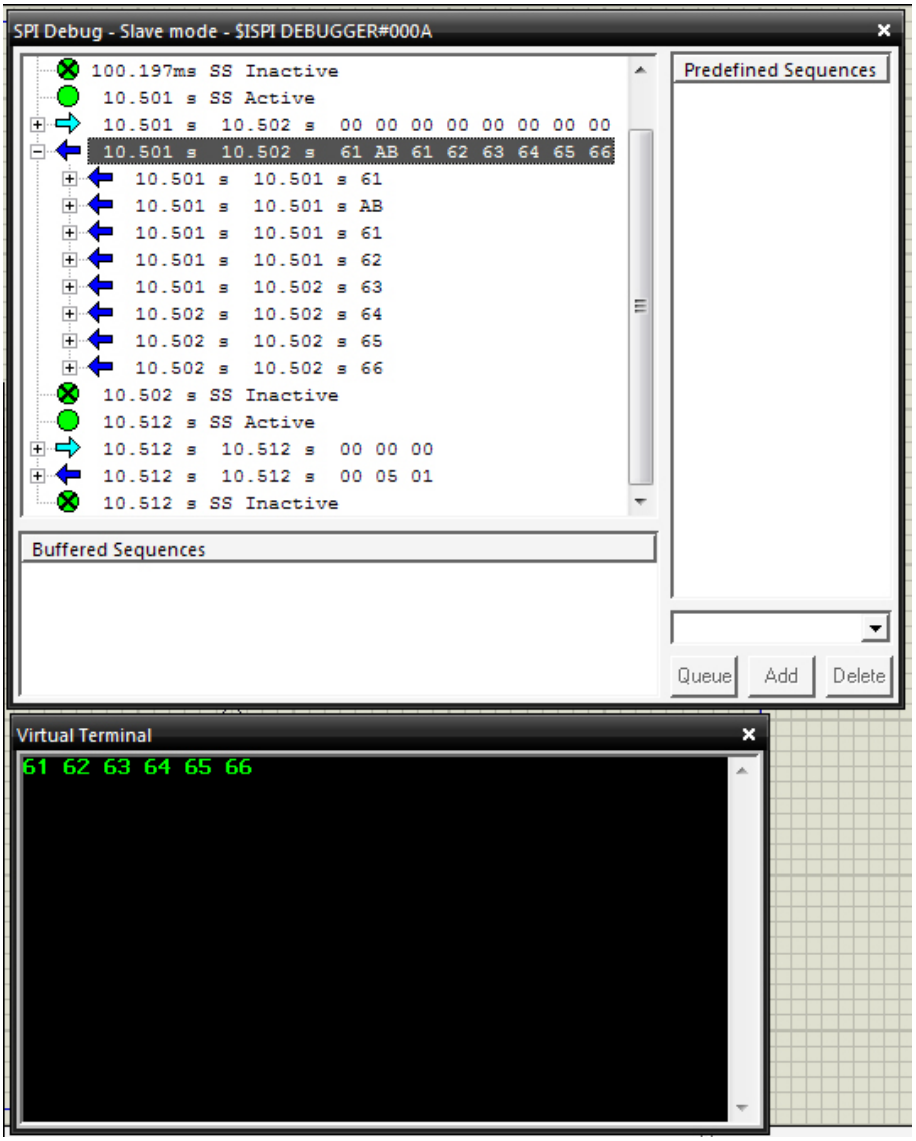


Figura 3.11: Terminal virtual e SPI Debug.

É possível ver na figura 3.10 o osciloscópio simulado em azul os dados, em rosa o clock e em verde o pino CSB. Na imagem 3.11 é possível verificar o *debugger* do SPI, mostrando toda a transmissão de dados programa e o console RS232 onde os números hexadecimais 61, 62, 63, 64, 65 e 66 representando a FTW. Ainda em 3.11 vemos o detalhe do terminal virtual com os números hexadecimais representando a FTW e o *debugger* com a transmissão toda: *instruction word*, byte de configuração do registrador 0x000, nova *instruction word*, seis bytes da FTW, terceira *instruction word* e *I/O Update*. O próximo teste usa o mesmo circuito, desta vez com o objetivo de gravar dados na memória EEPROM. Segue-se o programa:

```
void csb(void)
{
        PORTA=0x00;
```

```
        TRISA=0;
}

void main()
{
        int k;
        char dados[6], escreve, wrenable;

        UART1_Init(9600);
        SPI1_Init_Advanced(_SPI_MASTER_OSC_DIV64,
        _SPI_DATA_SAMPLE_END, _SPI_CLK_IDLE_LOW, _SPI_LOW_2_HIGH);
        csb();
        PORTA.f0 = 1;
        Delay_ms(100);            //Espera estabilizar
        wrenable=0b00000110;      //Ativa o latch que habilita escrita
        escreve=0b00000010;       //Byte que ativa modo de escrita


        while(1)
        {

           if(UART1_Data_Ready())
           {
              for (k=0; k<6; k++)
                {
                while(!UART1_Data_Ready()); // Lê 6 bytes até encher o buffer
                dados[k]=UART1_Read();
                }
              PORTA.f0 = 0;
              SPI1_Write(wrenable);       //Envia o byte que ativa o latch
              //WRITE ENABLE, precisa ser feito a cada escrita
              PORTA.f0 = 1;
              Delay_ms(100);
              PORTA.f0 = 0;               //Inicia a transmissão dos dados
              SPI1_Write(escreve);          //Ordem de escrita
              for(k=0;k<6;k++)
                {
                SPI1_Write(dados[k]); // Escreve o endereço (1 byte)
                // e os dados (5 bytes)
                }
              PORTA.f0 = 1; //Termina a escrita
```

```
                }

}
}
```

O programa acima é uma variação do anterior. Na figura 3.12 vemos novamente a simulação em ambiente ISIS e na figura 3.13 vemos a confirmação de que a programação funciona.

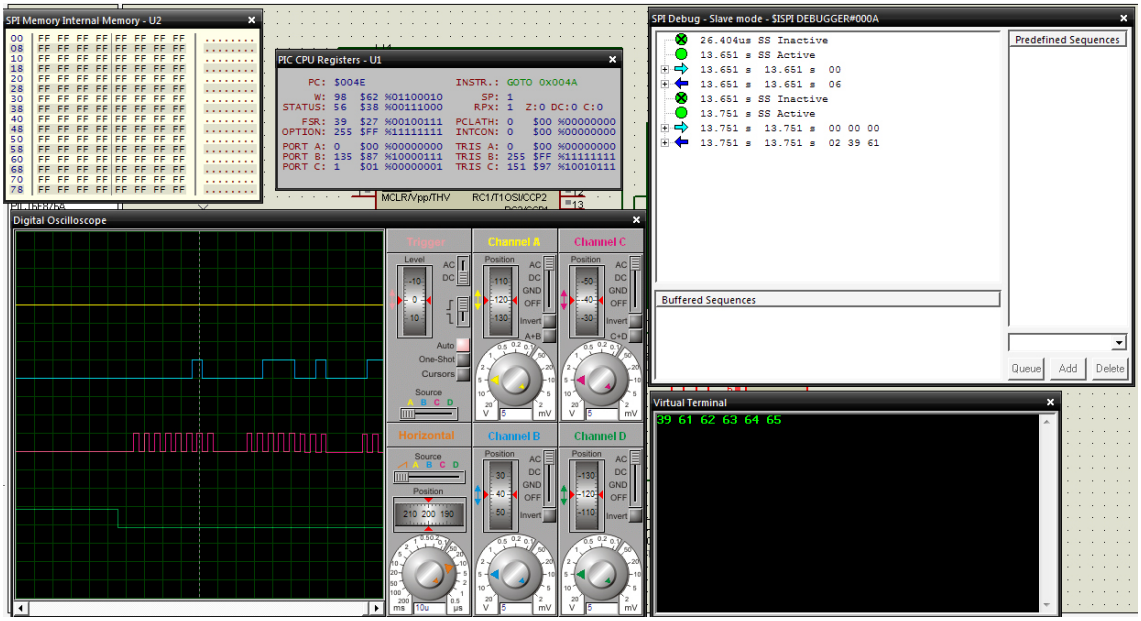


Figura 3.12: Circuito simulado na ferramenta ISIS.

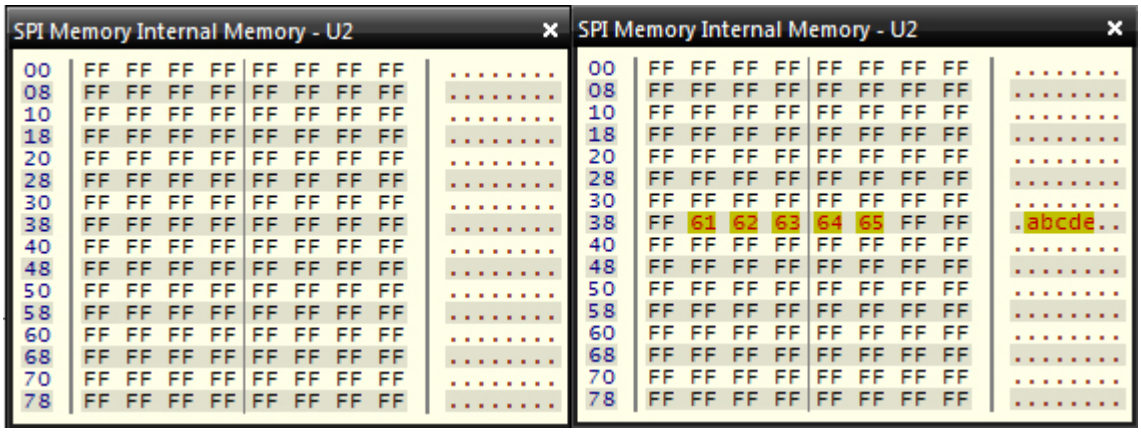


Figura 3.13: Verificação da memória antes e depois.

3.5 LabVIEW®

Para facilitar a integração dos sistemas de controle da frequência gerada pelo DDS aos sistemas já utilizados no relógio atômico tipo Fountain do CePOF (Centro de Pesquisa

em Óptica e Fotônica) do IFSC-USP, foi proposto criar uma interface através do software LabVIEW, que já é utilizado para controle de outros sistemas. Isto facilitaria o processo de ajuste de referências de frequência e reduziria o número de softwares utilizados nos computadores do laboratório, já que anteriormente era utilizado para ajuste do DDS o software da Analog Devices. Além disto, futuramente isto possibilita o controle de vários DDS através de uma única interface, como exemplificado na (figura 2.5), permitindo um controle mais eficiente por parte do operador em laboratórios grandes que dispõem de vários DDS como o AD9912.

O software LabVIEW, da National Instruments, permite a programação de softwares em uma linguagem de blocos que pode ser utilizada com relativa facilidade para aplicações como esta. Na interface criada o usuário insere a frequência de saída desejada e a frequência de entrada do gerador de sinal (no caso, 1 GHz), obtendo assim a FTW de 6 bytes e um botão para enviá-la via comunicação serial RS-232 para o microcontrolador. Existem também algumas opções para configurar a comunicação serial (como controle dos bits de parada), mas estes precisam ser iguais ao valor usado no microcontrolador e foram deixados apenas para manter a versatilidade da interface.

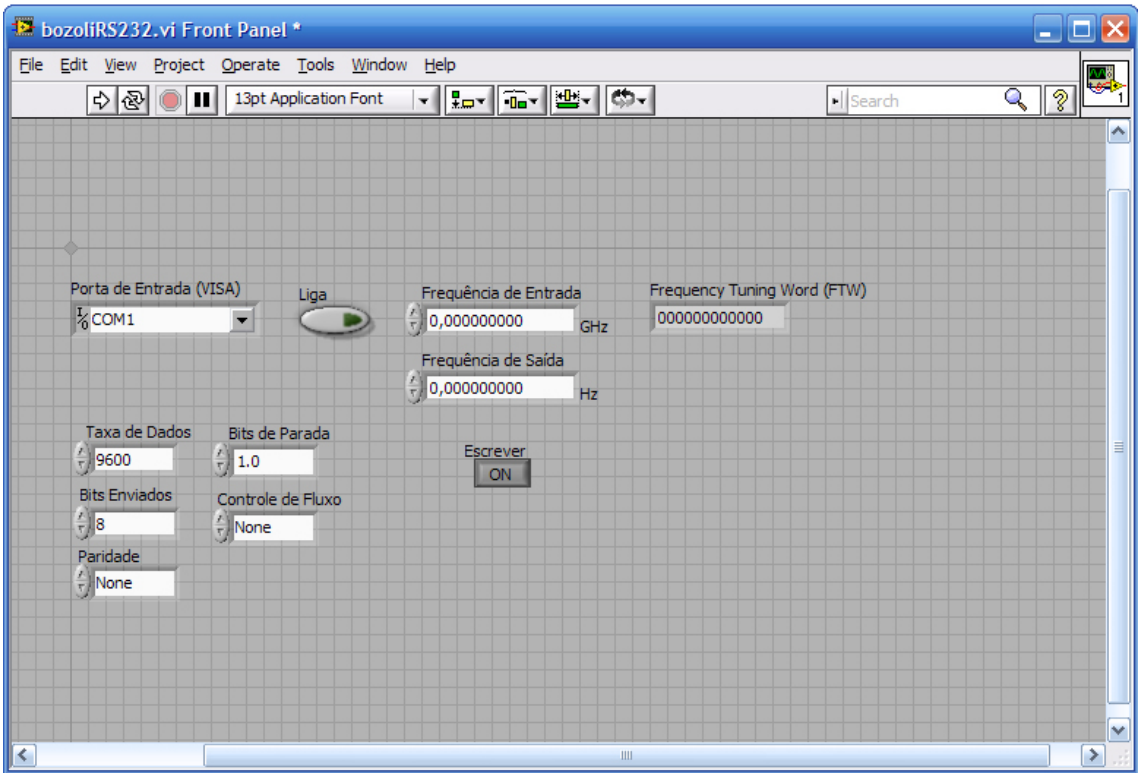


Figura 3.14: Interface do controle do DDS em LabVIEW®.

A sequência de blocos para a programação da interface é simples, executando apenas a equação 2.1 para determinar a FTW a ser inserida no microcontrolador. Um botão "Escreve" serve para ativar a comunicação RS-232 e inserir os dados.

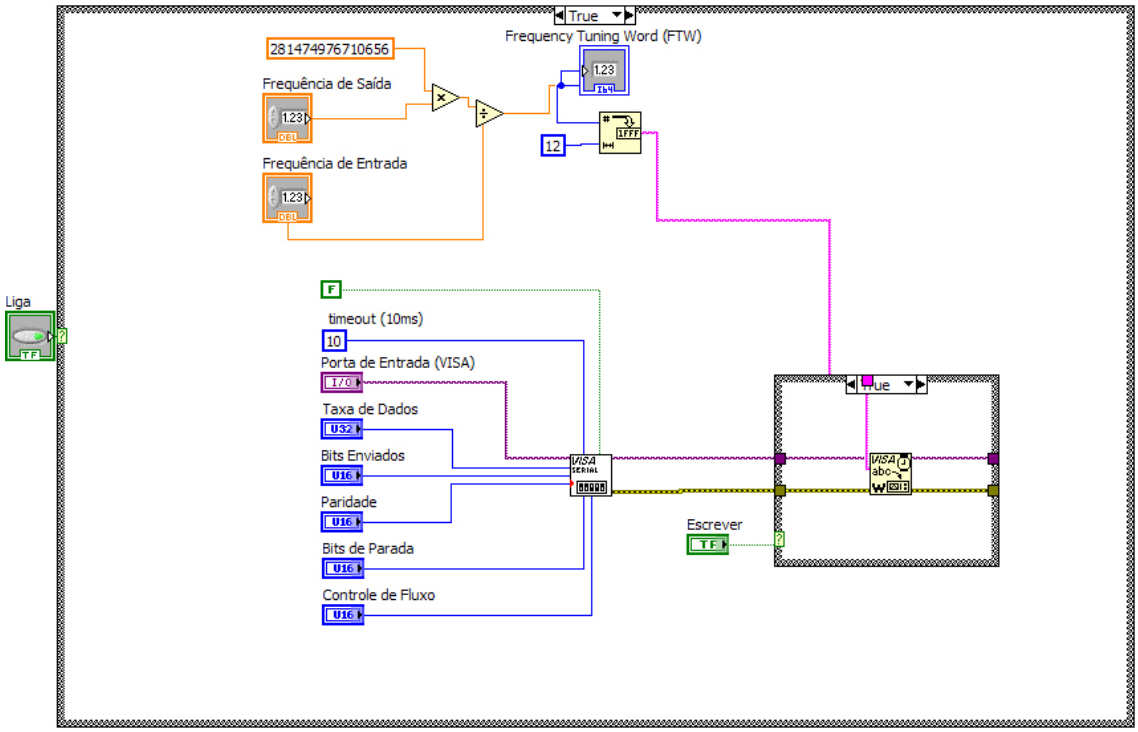


Figura 3.15: Código em blocos do controle do DDS.

CAPÍTULO 4: Resultados e Conclusões

Este capítulo se dedica a detalhar os resultados dos procedimentos experimentais citados no capítulo 3.

4.1 Testes com a placa AD9910/PCBZ

Conforme citado na seção 3.1.1, repetimos os testes do trabalho *Diagnóstico da Placa AD9910*.

4.1.1 Consistência do sinal gerado e da análise

Para a verificação da consistência do sinal gerado pelo SMB 100A Signal Generator e da análise desse sinal pelo MXA Signal Analyzer, ligamos diretamente o gerador de sinal com uma frequência de 250 MHz no analisador de espectro. A imagem 4.1 mostra resultado condizente com o esperado.

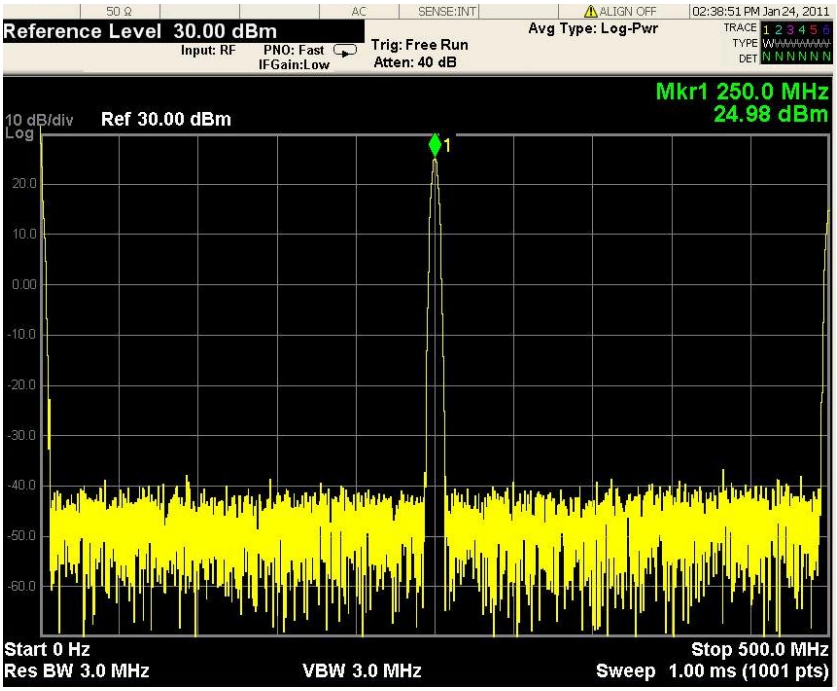


Figura 4.1: Resultado da verificação de consistência.

4.1.2 Documentação do espelhamento do sinal

Para a documentação do espelhamento do sinal que ocorre em $f_{CLOCK} \pm f_{OUT}$ e se repete em intervalo igual ao do clock, ligamos o SMB 100A Signal Generator na entrada de

clock de referência J1 e a saída não filtrada J3 da AD9910/PCBZ no MXA Signal Analyzer. Configuramos da seguinte maneira no software:

External clock	200 MHz
System clock	100 MHz
Profile 0 – Output Frequency	45 MHz
DAC Gain Control	0
I/O Sync Clock Output Pin	Enabled

Tabela 4.1: Configuração da documentação de espelhamento do sinal.

A figura 4.2 mostra resultado condizente com o esperado.

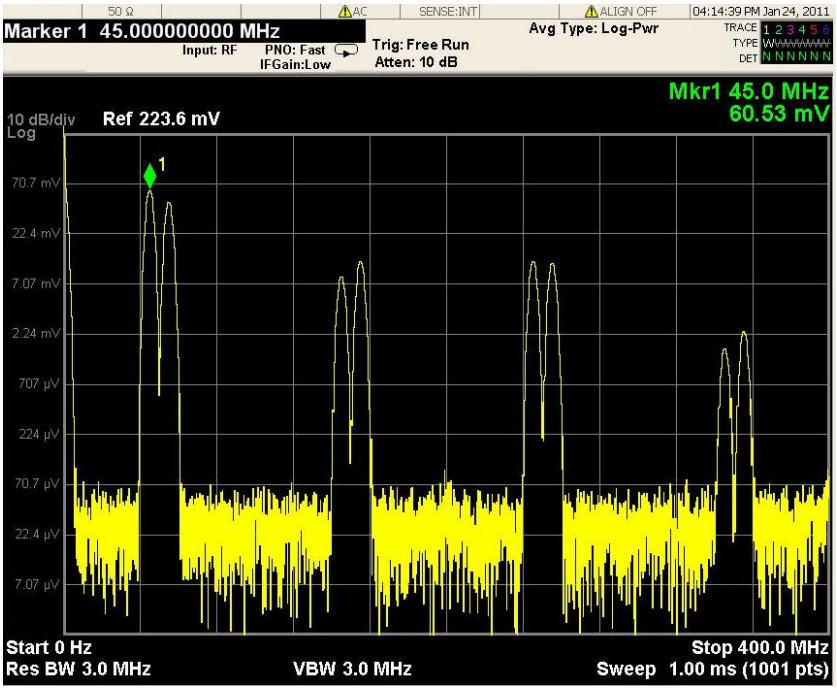


Figura 4.2: Resultado da documentação de espelhamento de sinal.

4.1.3 Controle de amplitude via ASF

No caso do controle da amplitude através da ASF mantivemos a mesma ligação física entre SMB 100A Signal Generator, AD9910/PCBZ e MXA Signal Analyzer presente no teste anterior. A tabela 4.2 apresenta os resultados da variação da ASF com clock externo de 100MHz, 10mV e saída de 10 MHz enquanto a tabela 4.3 apresenta os resultados variando ASF com clock externo de 200MHz, 10mV e saída de 20 MHz.

A figura 4.3 mostra a mudança de amplitude alterando ASF de 0,5 para 1 para a configuração de software exibida na tabela 4.4, novamente como esperado.

Iout(mA)	ASF	Iout X ASF(mA)	Corrente analisador(mA)
10	0,25	2,5	0,504
10	0,5	5	1,010
10	0,7	7	1,414
30	0,01	0,3	0,061
30	0,5	15	3,055

Tabela 4.2: Resultados com clock externo de 100MHz, 10mV e saída de 10 MHz.

Iout(mA)	ASF	Iout X ASF(mA)	Corrente analisador(mA)
20	0,4	8	1,688
20	0,5	10	2,110
20	0,1	2	0,422
30	0,1	3	0,636
30	0,5	15	3,182

Tabela 4.3: Resultados com clock externo de 200MHz, 10mV e saída de 20 MHz.

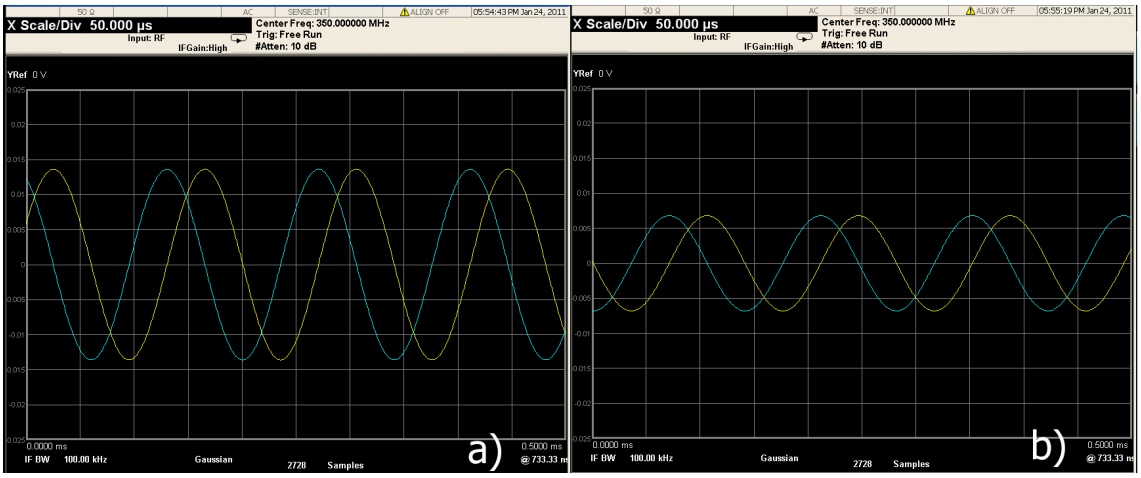


Figura 4.3: Resultado da a mudança de amplitude alterando ASF de 1 (a) para 0,5 (b) para a mesma configuração.

External clock	900 MHz
System clock	450 MHz
Profile 0 – Output Frequency	45 MHz
DAC Gain Control	0
I/O Sync Clock Output Pin	Enabled

Tabela 4.4: Configuração do controle de amplitude via ASF.

4.1.4 Consistência do sinal gerado e da análise

O teste utilização de sinal de saída filtrado foi feito com a configuração de software apresentada na tabela 4.5. O lado (a) da figura 4.4 mostra o resultado da ligação do SMB 100A Signal Generator na entrada de clock de referência J1 e a saída não filtrada J3 da AD9910/PCBZ no MXA Signal Analyzer, enquanto o lado (b) mostra o resultado de ligar a saída filtrada J4 da AD9910/PCBZ no MXA Signal Analyzer. Novamente resultado condizente com o esperado.

External clock	200 MHz
System clock	100 MHz
Profile 0 – Output Frequency	45 MHz
DAC Gain Control	127
DAC I_{OUT}	20,116458 mA
I/O Sync Clock Output Pin	Enabled

Tabela 4.5: Configuração do teste de utilização de saída filtrada.

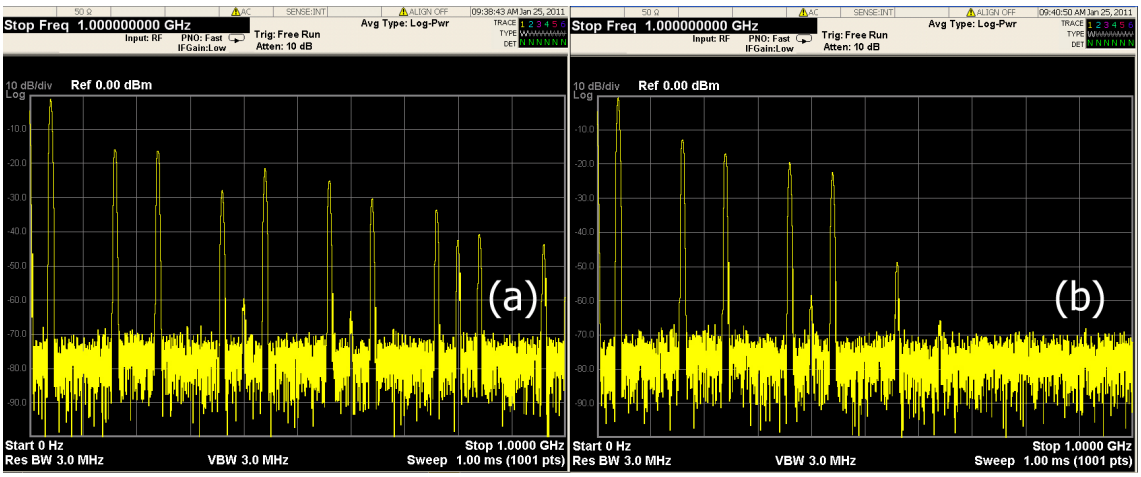


Figura 4.4: Resultado do teste de uso da saída filtrada (b) em comparação com uso da saída não-filtrada (a).

4.1.5 Uso do multiplicador de frequência

O teste do uso do multiplicador de frequência requer configurações completamente diferentes das anteriores, utilizando um clock externo de frequência baixa em relação às utilizadas anteriormente. Este perfil de configuração é registrado na tabela 4.6 e o resultado, ligando a saída filtrada J4 da AD9910/PCBZ no MXA Signal Analyzer, pode ser visto na imagem 4.5. Novamente, tivemos o resultado esperado.

External clock	10 MHz
/2 Divider	Disabled
Multiplier	Enabled - 100x
VCO Range	920 - 1080 MHz
System clock	1000 MHz
Profile 0 – Output Frequency	200 MHz
DAC Gain Control	127
DAC I_{OUT}	20,116458 mA
I/O Sync Clock Output Pin	Enabled

Tabela 4.6: Configuração para teste de uso do multiplicador de frequencia.

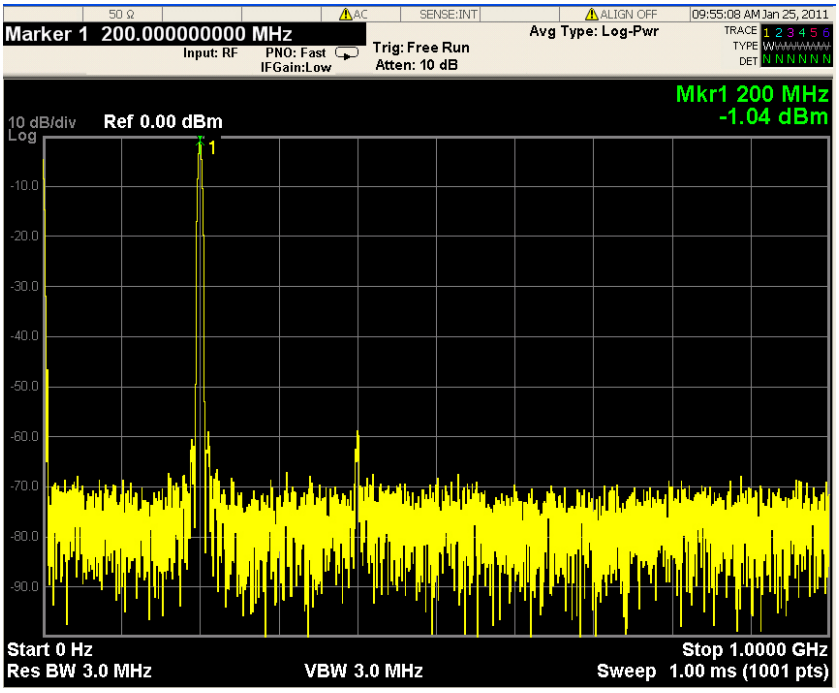


Figura 4.5: Resultado do teste de uso do multiplicador de frequência.

4.1.6 Teste do modo RAM

Foram feitos três testes de uso de memória RAM, sendo dois no modo Direct Switch e um no modo Ramp Up. Mantivemos o uso da saída filtrada J4 e os dois testes no modo Direct Switch seguem a configuração de software apresentada na tabela 4.7.

External clock	200 MHz
/2 Divider	Disabled
System clock	200 MHz
Enable RAM	
Ram Segment	0
Beginning Adress	0
Final Adress	409
Mode Control	Direct Switch

Tabela 4.7: Configuração dos testes de modo RAM.

Para o primeiro teste com esta configuração desejamos uma saída de 60 MHz e para o segundo teste 50 MHz. Sendo $FTW = 2^{32} \cdot \frac{f_{OUT}}{f_{SYSCLK}}$ temos então $FTW = 0,3 \cdot 2^{32}$ e $FTW = 0,25 \cdot 2^{32}$ para f_{OUT}=60MHz e f_{OUT}=50MHz, respectivamente. Tais valores devem ser convertidos para binário para serem colocados na primeira linha do arquivo *RAMDump.txt*. Os resultados destes testes estão na figura 4.6, sendo o lado (a) para f_{OUT}=60 MHz e o lado (b) para f_{OUT}=50 MHz.

Para o teste com modo Ramp Up, a configuração é a mesma da tabela 4.7, apenas mudando o *Mode Control* para *Ramp Up*. Desejamos uma saída de 43 MHz o que nos leva a $FTW = 0,215 \cdot 2^{32}$, valor que também deve ser convertido para binário e então colocado na última linha do arquivo *RAMPDump.txt*. O resultado deste testes está na figura 4.7.

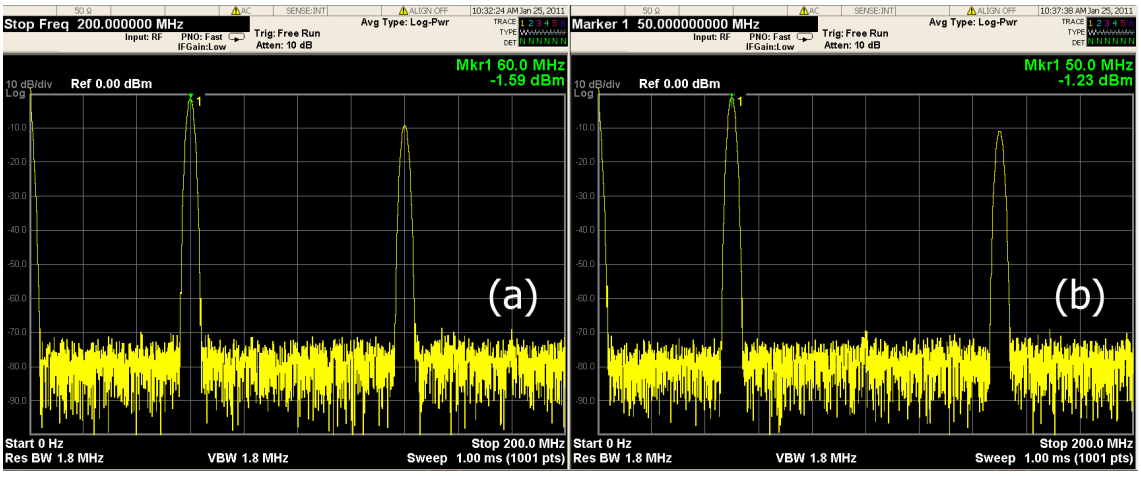


Figura 4.6: Resultado dos testes de modo Direct Switch, sendo o lado (a) para f_{OUT}=60 MHz e o lado (b) para f_{OUT}=50 MHz.

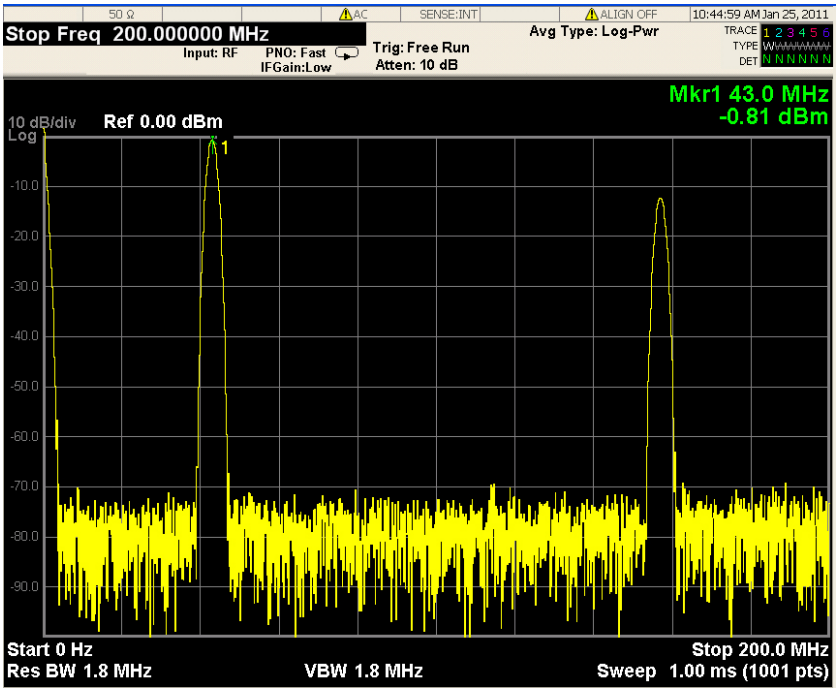


Figura 4.7: Resultado do teste de modo Ramp Up.

4.1.7 Testes de amplitude, frequência e potência

Foram feitos sete testes de amplitude, frequência e potência para verificar a existência de uma relação entre o valor de corrente que controlamos através do software (I_{OUT}) e o valor de corrente medido na saída(I_{SAIDA}). Utilizando os resultados onde não ocorreram corte de sinal, calculamos:

$$I_{SAIDA} = \frac{P}{V_{RMS}} \tag{4.1}$$

$$I_{OUT} = DACI_{OUT} \cdot ASF \tag{4.2}$$

E por fim, encontramos uma razão:

$$K = \frac{I_{OUT}}{I_{SAIDA}} \tag{4.3}$$

Utilizamos para esses testes a saída não-filtrada J3. Para o primeiro teste, configuramos o clock externo para 200 MHz com nível de 5,00 dBm, system clock para 100 MHz, frequência de saída para 50 MHz no Profile 0, I/O Sync Clock Output Pin habilitado e variamos o DAC Gain Control e o ASF. Os resultados e as configurações de DAC Gain Control e ASF do primeiro teste estão na tabela 4.8 .

De posse dos resultados da tabela supracitada, calculamos os valores de I_{OUT}, I_{SAIDA} e

DAC Gain Ctrl	DAC IOUT(mA)	ASF	Amplitude (V)	V_{RMS}	RF Envelope (dBm)	dBm hh to W
255	31,663125	0,25	0,138	0,097581	-7,289	0,000186681
255	31,663125	0,5	0,276	0,195161	-1,266	0,000747137
255	31,663125	1	0,41[1]		Max 4,763 Mín 2,301 Med 3,532[2]	Max 0,002994332 Mín 0,001698635 Med 0,0023464835[3]
96	17,320000	0,25	0,08	0,056569	-12,59	$5{,}50808 \cdot 10^{-5}$
96	17,320000	0,5	0,15	0,106066	-6,571	0,000220242
96	17,320000	1	0,3	0,212132	-0,549	0,000881252

Tabela 4.8: Resultados do primeiro teste de amplitude, frequência e potência.

K com as equações 4.1, 4.2 e 4.3. Resultados presentes na tabela 4.9.

I_{OUT} (A)	I_{SAIDA} (A)	K
0,007915781	0,001913092	4,137689135
0,015831563	0,0038283	4,135403374
-	-	-
0,00433	0,0009737	4,446956544
0,00866	0,002076461	4,170557884
0,01732	0,004154261	4,169213588

Tabela 4.9: Cálculos do primeiro teste de amplitude, frequência e potência

Cada linha da tabela 4.9 corresponde aos cálculos para a mesma linha da tabela 4.8. Devemos desconsiderar a terceira linha da tabela 4.8, pois se trata da linha com sinal cortado. A figura 4.8 mostra o resultado desta terceira linha. Assumimos K=4,15 pois é o meio termo entre 4,135 e 4,170. Observamos inconsistência com o K encontrado na página 28 do trabalho *Diagnóstico da Placa AD9910*, K = 5, e devido a este resultado conflitante decidimos continuar com os testes na tentativa de encontrar um padrão.

Fizemos mais seis testes verificando o valor K. Os testes 2 e 3 permitiram concluir que o valor K depende diretamente da frequência de saída ou do clock do sistema. O testes 4 nos permitiu delimitar que esta relação com a frequência de saída. Fixamos nestes testes o system clock em 200 MHz e a frequência de saída foi ajustada para 150 MHz nos testes 2 e 3, onde variamos o clock externo e 100 MHz no teste 4 Estas configurações podem ser vistas na tabela 4.10. Apresentamos também na mesma tabela os valores de K encontrados em cada teste. Os resultados destes testes estão no anexo A.

[1] Valor cortado.

[2] Equivale a 0,002255278 W.

[3] Equivale a 3,704 dBm.

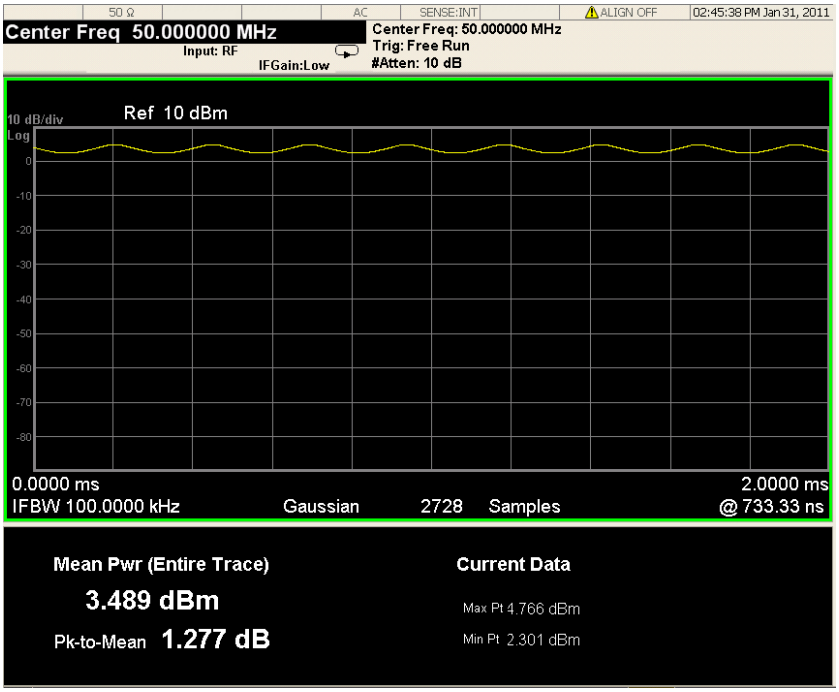


Figura 4.8: Resultado desconsiderado do primeiro teste de amplitude, frequência e potência.

Teste	2	3	4
External clock (MHz)	200	400	400
/2 Divider	Disabled	Enabled	Enabled
System clock (MHz)	200	200	200
Frequência de Saída (MHz)	150	150	100
K	30,6	30,5	8,5

Tabela 4.10: Configurações dos testes 2, 3 e 4 de amplitude, frequência e potência.

De posse desta conclusão, decidimos fazer mais testes para tentar encontrar uma relação matemática. Fizemos mais três testes onde fixamos o clock externo em 800 MHz com nível de 5,00 dBm, system clock em 400 MHz e I/O Sync Clock Output Pin habilitado. Mudamos a frequência de saída para 200, 250 e 350 MHz no Profile 0 nos testes 5, 6 e 7 respectivamente. Os resultados destes testes estão no anexo A. Os valores de K resultantes dos testes 5 e 6 foram respectivamente 3,6 e 11. Porém o sétimo teste apresentou K aproximadamente 31,4 para DAC Gain Ctrl definido em 255 e aproximadamente 9,4 para DAC Gain Ctrl 96, resultado este que sugere que embora K não seja um valor aleatório, depende de mais do que apenas uma variável.

Como o escopo deste trabalho não é confirmar ou contestar dados anteriores, deixamos de lado a tentativa de encontrar uma fórmula para K. Decidimos então nos concentrar na tentativa de controlar a frequência via porta serial sem o uso do programa específico da placa. Após leitura dos datasheets do chip AD9910 e da placa AD9910/PCBZ, decidimos iniciar pelo modo de operação aparentemente mais simples: o Single Tone.

4.2 Modo Single Tone da placa AD9910/PCBZ

Conforme citado na seção 3.1.1, construímos uma placa de conversão RS232 para DB9 para utilização com o arquivo *.vi* com comunicação de porta serial básica (mostrado na figura 3.2). Neste arquivo programamos cada byte que deve ser enviado. Selecionamos então os registradores necessários para programação e definimos os bytes necessários para os resultados que gostaríamos:

- **0x00 - CFR1:** 00 00 00 00 02, que resulta em SDIO Input Only, sendo os dois primeiros algarismos hexadeicimais indicativos do registrador;
- **0x01 - CFR2:** 01 01 C0 08 20, que resulta em ASF Enabled, Internal I/O Update Active e SYNC_CLK enabled;
- **0x02 - CFR3:** 02 1F BF 40 00, que resulta em Divider disabled ou 02 1F BF C0 00, que resulta em Divider enabled;
- **0x04 - I/O Update Rate:** 04 00 00 61 A8, que define em 1ms;
- **0x0E - Profile 0:** 0E 3F FF 00 00 *XX XX XX XX*, onde os valores de X mudam dependendo da frequência desejada.

Levando em consideração o datasheet do chip[4] denotamos a ordem necessária de programação dos registradores: 04, 00, 01, 02 e 0E. Programamos então a seguinte sequencia: 04 00 00 61 A8 00 00 00 00 02 01 01 C0 08 20 02 1F BF 40 00 0E 3F FF 00 00 80 00 00 00. Com esta sequência e clock externo de 200 MHz obtivemos um system clock de 200 MHz com /2 Divider desativado, frequência de saída de 100 MHz, phase offset de 0,000 graus, profile ASF habilitado e com valor 0,9994. Assim como nos testes da seção 4.1, ligamos o SMB 100A Signal Generator na entrada de clock de referência J1 e a saída não filtrada J3 da AD9910/PCBZ no MXA Signal Analyzer.

4.3 Testes com a placa AD9912/PCBZ

O primeiro teste com a AD9912/PCBZ teve a finalidade de testar o modo PLL. Com o system clock em 25 MHZ e os bits S4, S3, S2 e S1 em 0, 0, 0 e 1, respectivamente, indicam SYSCLK Input Mode para Xtal/PLL representado pelo valor do bit S4 e enquanto os outros três bits indicariam uma frequência de saída de 38,87 MHz considerando multiplicador PLL com valor default de 40. O registrador 0020h inicia em valor 12h, que implica em multiplicador PLL de 40x. Com o sinal de 54 MHz registrado na figura 4.9 verificamos que o filtro PLL instalado na placa não segue o valor default, e sim o valor de 56x. Atualizando

[4] [Analog Devices, Inc. 2007-2010]

o registrador 0020h para 1Ah, com o valor calculado pelo programa de avaliação da placa de 54,4311523437500 MHz, temos o sinal de aproximadamente 54,43 MHz visto na figura 4.10.

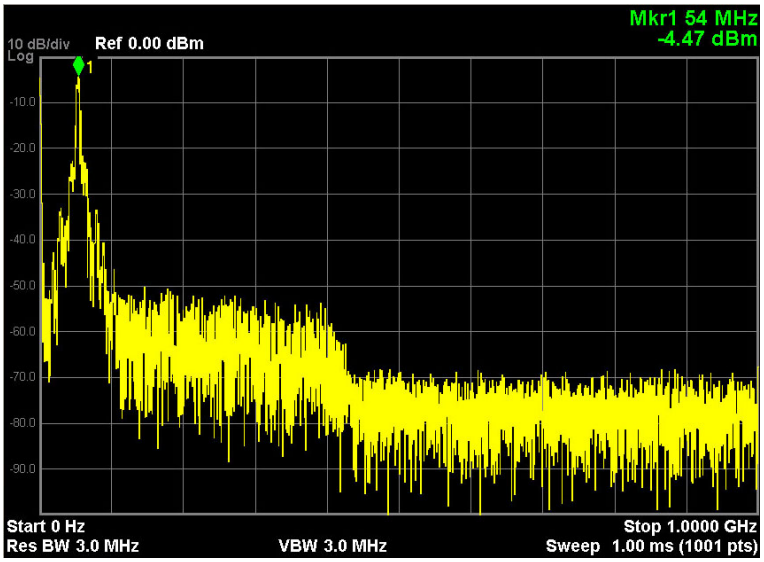


Figura 4.9: Frequência de saída de 54 MHz implicando em PLL 56x.

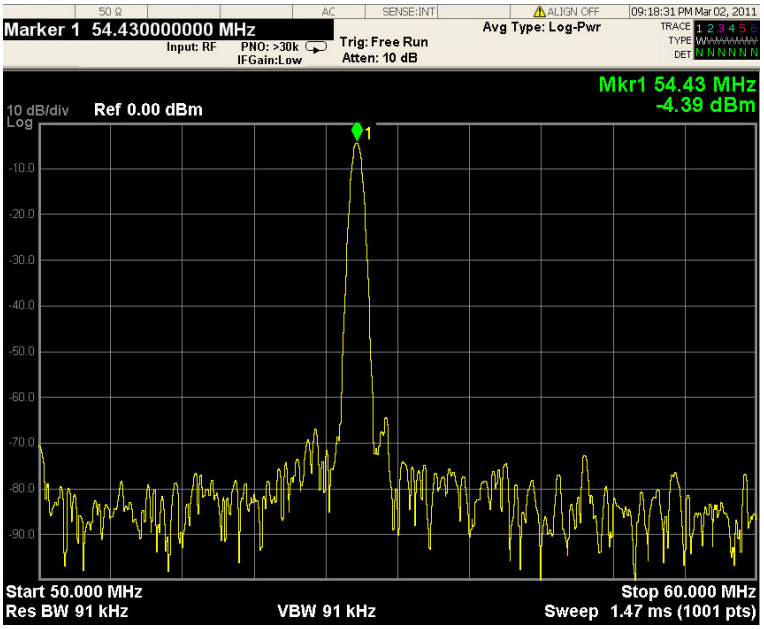


Figura 4.10: Frequência de saída de 54,43 MHz após atualização do registrador 0020h.

Colocando o valor 249249249249h nos registradores 01AB até 01A6 temos 200 MHz, 36DB6DB6DB6Eh nos registradores 01AB até 01A6 temos 300 MHz e colocando o valor 492492492492h nos registradores 01AB até 01A6 temos 400 MHz na imagem 4.11. Podemos notar uma atenuação em 425 e 450 MHz, fazendo com que essas frequências não espelhem 375 e 350 MHz respectivamente.

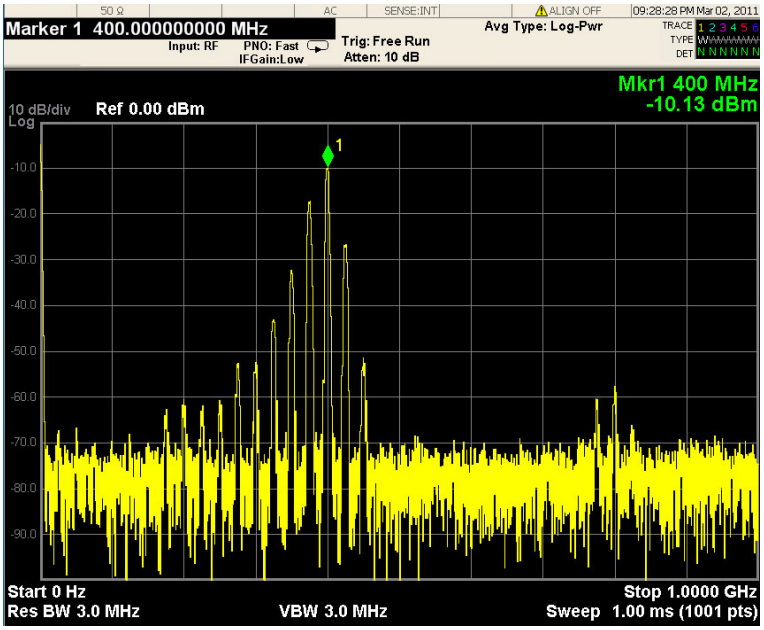


Figura 4.11: Frequência de saída de 400 MHz.

Colocando o valor 5B6DB6DB6DB7h nos registradores 01AB até 01A6 teríamos 500 MHz em teoria, porém a saída filtrada nos dá o resultado visto na figura 4.12. Vimos resultado similar em 600 MHz, 700, 800 e 900 MHz. Colocando o valor B6DB6DB6DB70h nos registradores 01AB até 01A6 teríamos 1 GHz em teoria, porém a saída filtrada nos dá o resultado visto na imagem 4.13, resultado similar ao de 400 MHz. Colocando 1.1 GHz temos espelhamento do resultado de 300 MHz, 1.2 GHz com 200 MHz e 1.3 GHz com 100 MHz. Colocando o valor D2A6C405D9F7h nos registradores 01AB até 01A6 teríamos 1152 MHz em teoria, porém a saída filtrada nos dá o resultado visto em 4.14, resultado similar ao de 248 MHz. Colocando mais alguns valores aleatórios próximos a ou maiores que 1 GHz concluímos que o resultado espelha a diferença da frequência colocada com relação à 1.4 GHz (equação 4.4, em MHz, se a frequência programada é acima de aproximadamente 950 MHz).

$$F_{SAIDA} = 1400 - F_{PROGRAMADA} \tag{4.4}$$

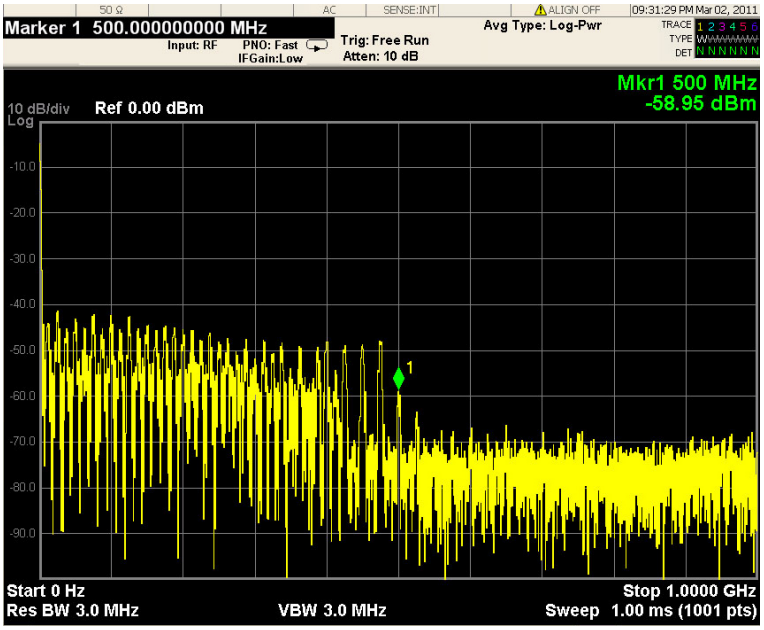


Figura 4.12: Atenuamento em 500MHz

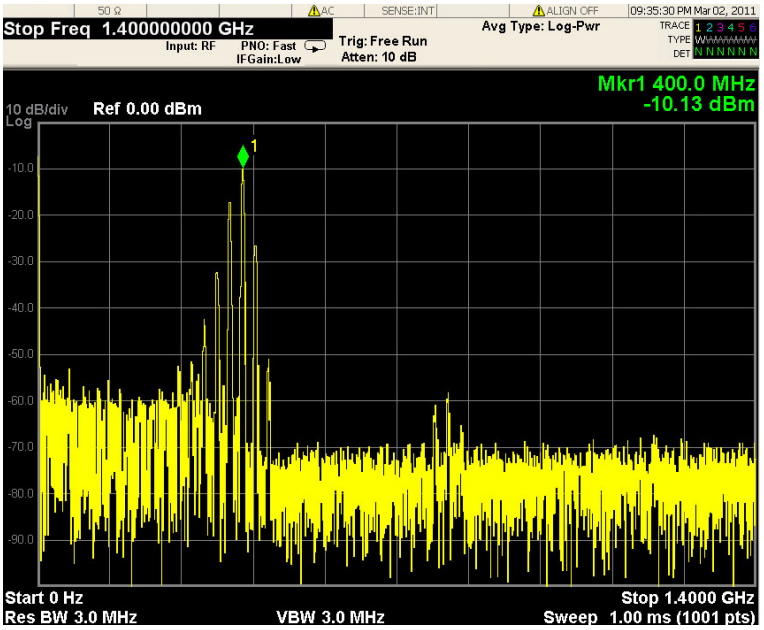


Figura 4.13: Frequência de saída de 400 MHz com programação em 1GHz

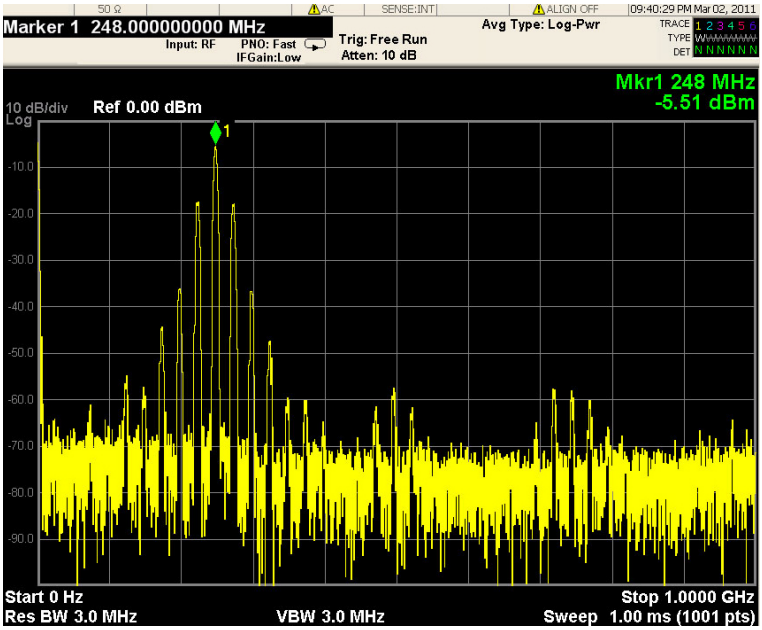


Figura 4.14: Frequência de saída de 248 MHz com programação em 1152 MHz

Mudando a saída de DUT_FILTER_OUT utilizada nos testes anteriores para CMOS_OUT, refizemos os passos. Reiniciando a placa, o registrador 0020h inicia em valor 12h, que implica em multiplicador PLL de 40 vezes. Com isso e os valores de S1, S2, S3 e S4, conseguimos o sinal de 54 MHz. Atualizando o registrador 0020h para 1Ah (56 vezes), temos o sinal de 54 MHz visto na figura 4.15. Colocando o valor 124924924925h nos registradores 01AB até 01A6 temos 100 MHz na figura 4.16. Conseguimos também 250 MHz e 400 MHz. Configurando a saída para 450 MHz podemos ver um pico em 426 MHz e menor dBm para frequências mais altas na figura 4.17. Entre 500 e 900 MHz os resultados são similares aos que encontramos na mesma faixa de frequências utilizando a saída filtrada. E novamente, para valores próximos a ou maiores que 1 GHz concluímos que o resultado espelha a diferença da frequência colocada com relação à 1.4 GHz.

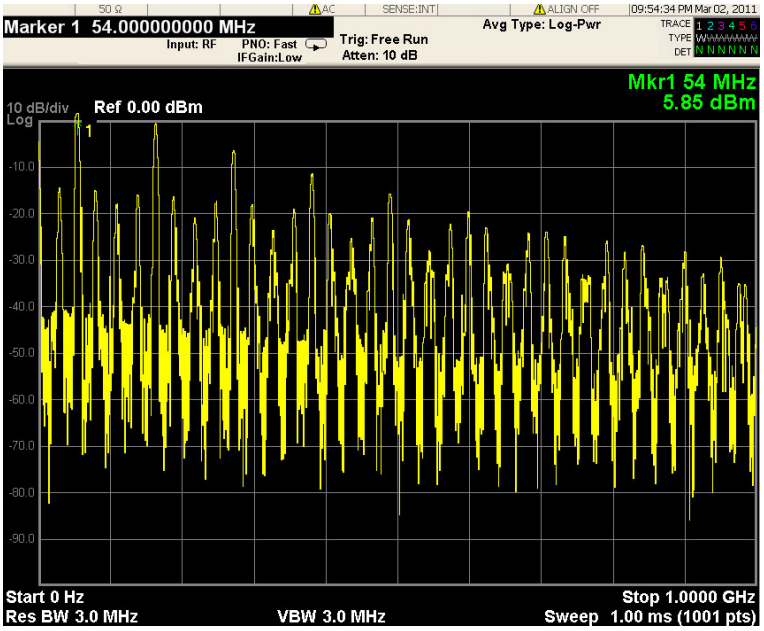


Figura 4.15: Frequência de saída de 54 MHz na saída CMOS_OUT.

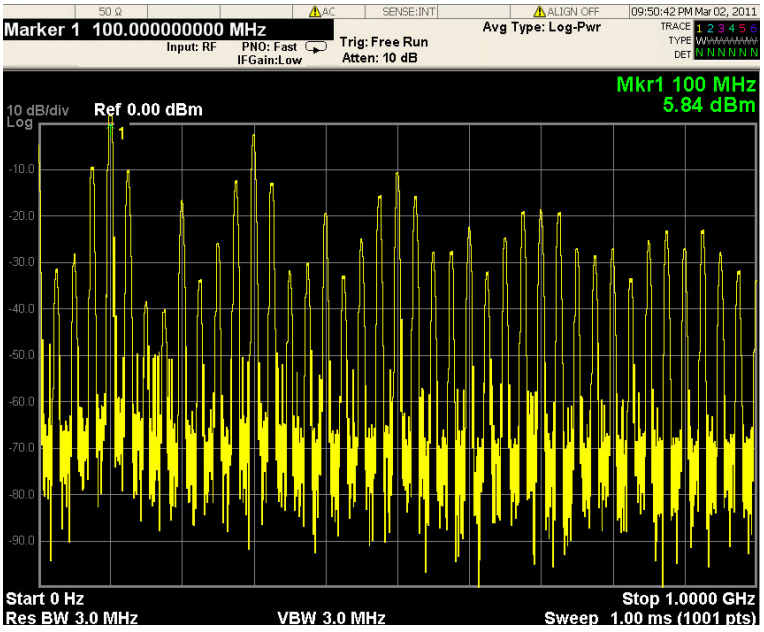


Figura 4.16: Frequência de saída de 100 MHz na saída CMOS_OUT.

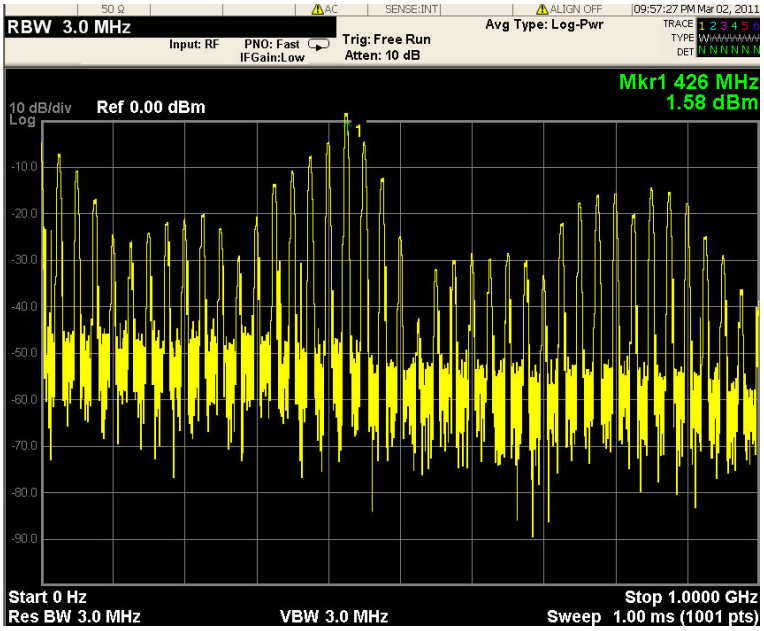


Figura 4.17: Pico de frequência de saída em 426 MHz com programação de 450 MHz na saída CMOS_OUT.

Como a aplicação usará 10 MHz ou 40 MHz, com PLL, mudamos o clock de sistema para 40 MHz, ainda mantendo a saída como CMOS_OUT. Configuramos os bits (S4, S3, S2, S1) como (0, 1, 1, 0), onde S4 em um indica SYSCLK Input mode para direto e (S3, S2, S1) em (1,1,0) indicaria uma frequência de saída de 196.6 MHz considerando *sysclk* de 40 MHz e multiplicador de 40 vezes, porém obtivemos o resultado de 183 MHz.
Colocamos 071C71C71C72h nos registradores de FTW e multiplicador de 36 vezes (que indica system clock interno de 1440 MHz), indicando os 40 MHz vistos na figura 4.18. Mantendo o multiplicador de 36 vemos os valores de 100 MHz, 200 MHz (figura 4.19), 300 MHz e 400 MHz (figura 4.20). Mantendo a característica de atenuamento a partir de aproximadamente 450 MHz registrado nos testes anteriores, vemos as saídas para os valores hexadecimais correspondentes a 500 MHz, 700 MHz e 900 MHz na figura 4.21 A, B e C respectivamente. Colocando valor hexadecimal para 1GHz temos o pico de 400 MHz em 4.22.
Imaginamos que o pico deveria ser em 440 MHz dada que esta é a diferença entre 1GHz e o clock interno de 1.44 GHz. Colocando valor hexadecimal para 440MHz temos o pico de 400 MHz em 4.23. Colocando valores de 1040 MHz, 1140 MHz, 1240 MHz, 1340 MHz e 1400 MHz tivemos resultados espelhados de 400 MHz, 300 MHz, 200 MHz, 100 MHz e 40 MHz respectivamente. Resultados semelhantes são obtidos com multiplicador de 34 vezes (que indica system clock interno de 1360 MHz).

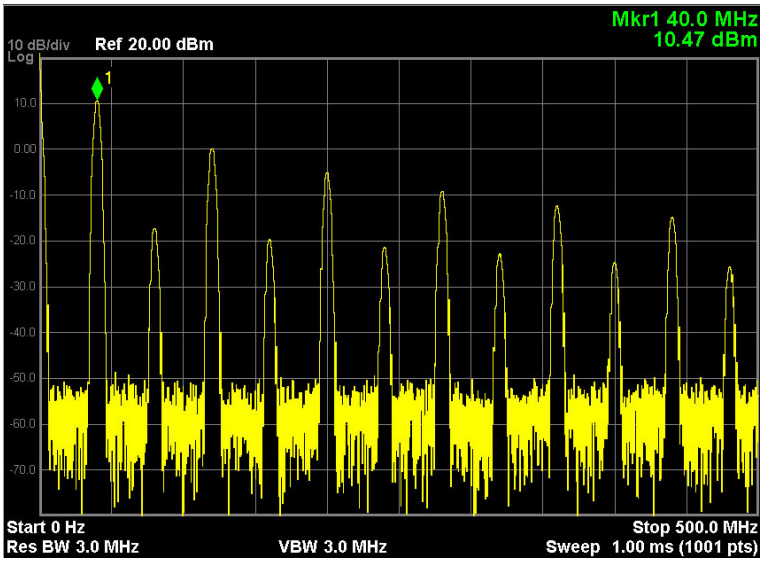


Figura 4.18: Frequência de saída em 40 MHz com multiplicador PLL de 36 vezes.

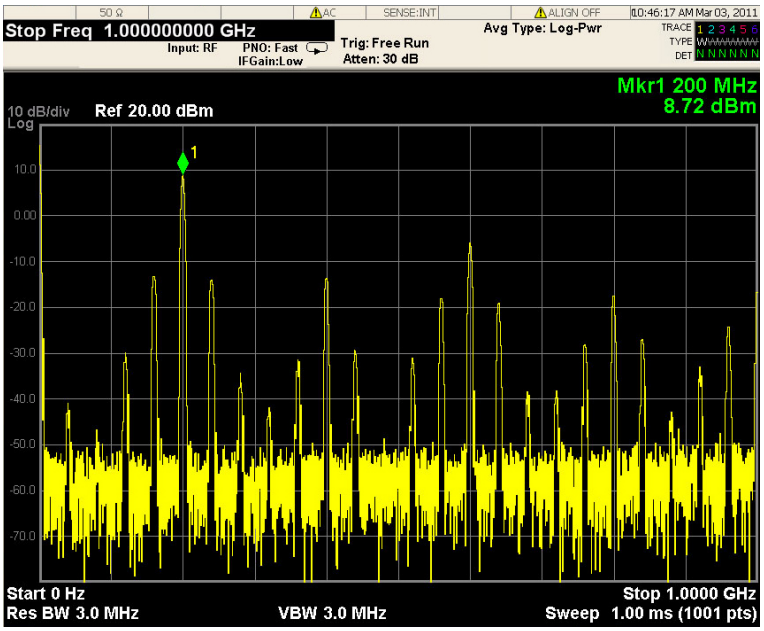


Figura 4.19: Frequência de saída em 200 MHz com multiplicador PLL de 36 vezes.

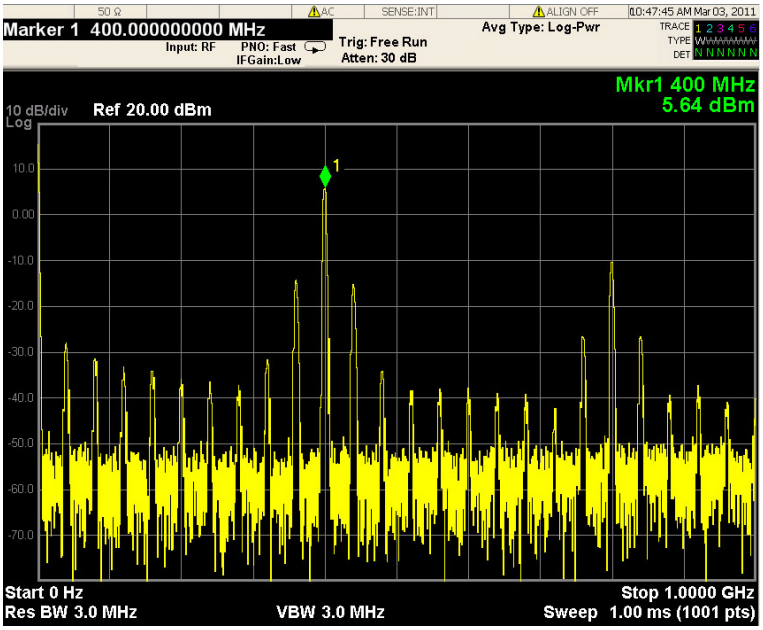


Figura 4.20: Frequência de saída em 400 MHz com multiplicador PLL de 36 vezes.

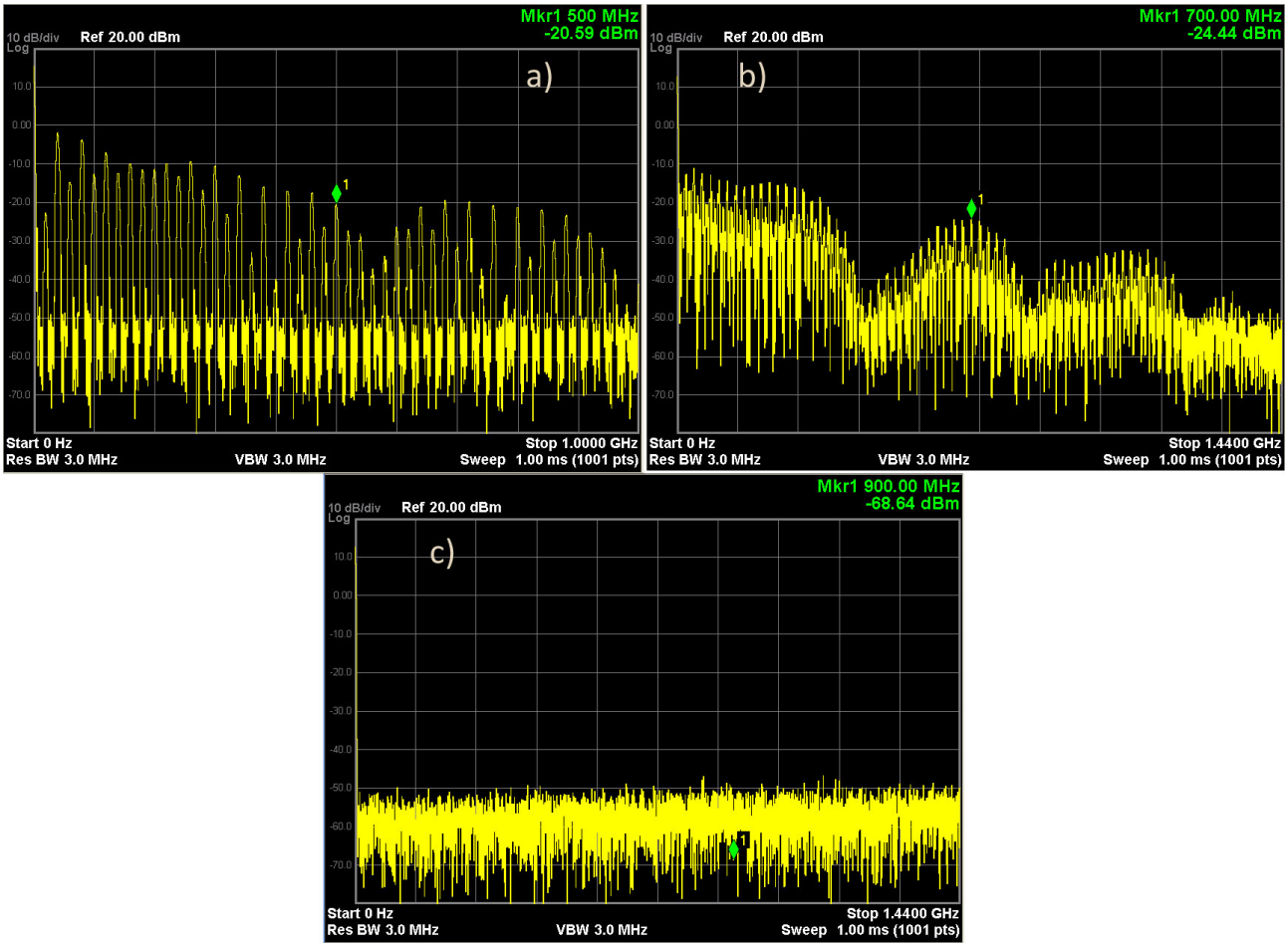


Figura 4.21: Frequências de saída para valores hexadecimais de 500 MHz, 700 MHz e 900 MHz em A, B e C respectivamente.

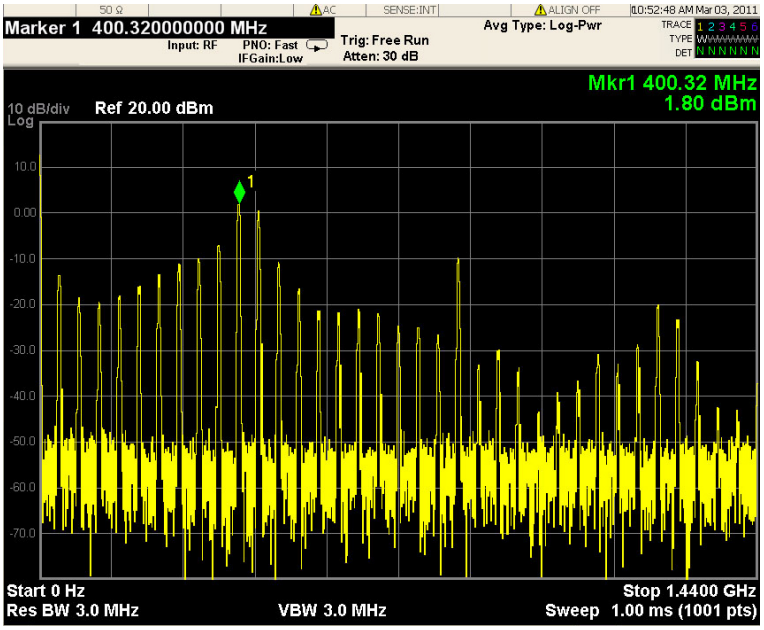


Figura 4.22: Pico de 400 MHz para valor hexadecimal de 1 GHz.

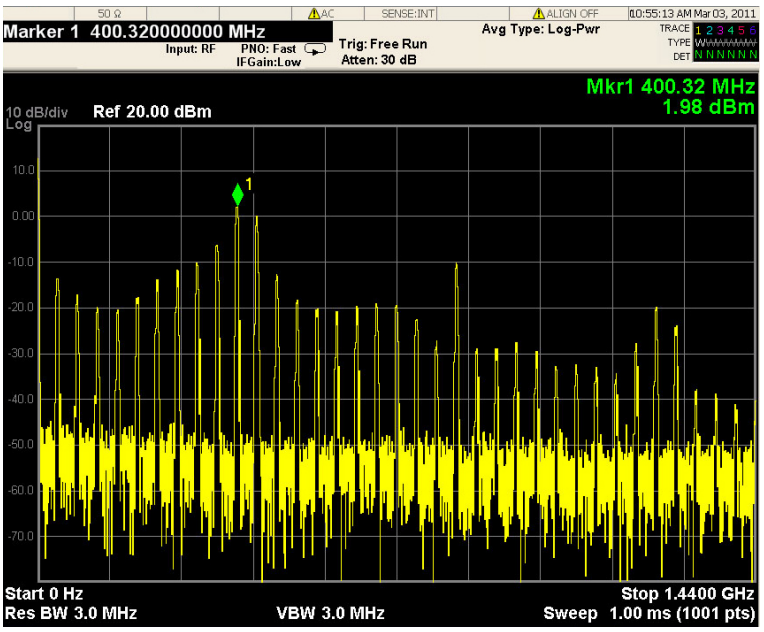


Figura 4.23: Pico de 400 MHz para valor hexadecimal de 440 MHz

Verificamos que os melhores resultados surgem com o sytem clock interno em cerca de 1400 MHz. De acordo com o datasheet do AD9912 [Analog Devices, Inc. 2007–2010, p. 6] o *range* de frequência de referência sem o PLL Doubler é de 11 até 200 MHz e com o PLL Doubler é de 6 MHz até 100 MHz. Para utilizar 10 MHz como entrada, precisamos ativar o PLL Doubler. Tentamos inicialmente usar *double* e multiplicador de 66 vezes (o que implicaria em fator de multiplicação 132 vezes e clock interno de 1.32 GHz) e valor hexadecimal correspondente à 400 MHz e obtivemos o resultado visto em **??**. Usando doubler

e multiplicador de 34 vezes, conseguimos um fator de multiplicação de 68 vezes, gerando system clock interno de 680 MHz. Colocando o valor hexadecimal correspondente à 200 MHz, obtivemos o resultado de 400 MHz (como se o clock interno fosse de 1360 MHz) encontrado em 4.24. Continuamos tendo resultado dobrado em valores hexadecimais para 50 MHz, 100 MHz e 125 MHz: 100 MHz, 200 MHz e 250 MHz, respectivamente.

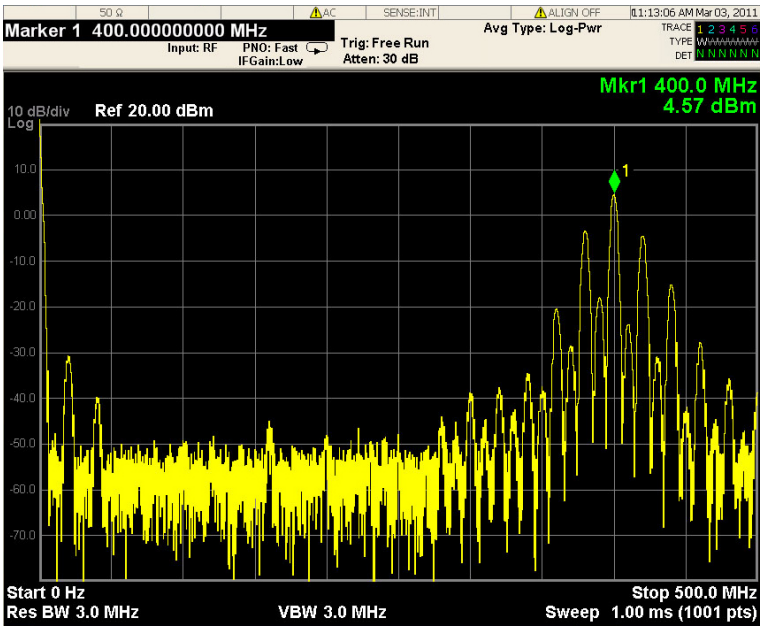


Figura 4.24: Pico de 400 MHz para valor hexadecimal de 200 MHz com PLL Doubler.

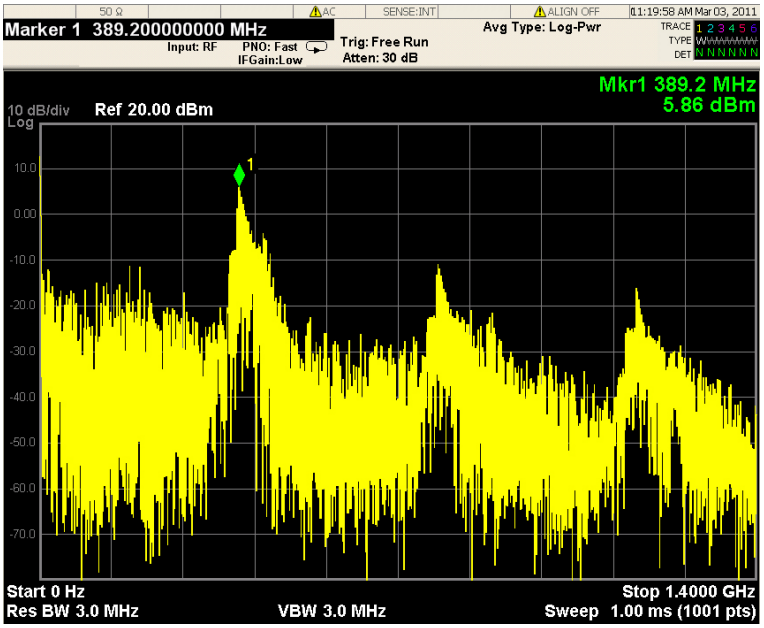


Figura 4.25: Pico de 389.2 MHz para valor hexadecimal de 400 MHz com PLL Doubler.

4.4 Testes com a placa do FT2232D

Inicialmente testamos a função SPI_Read, sem ter a função SPI_Write no programa. No entanto, o funcionamento previsto não foi observado. Independente de qual configuração fosse usada, o buffer de leitura de dados apresentou apenas bits em nível alto, enquanto sua saída FTC_Status retornou valor zero, sem indicação de erros (figura 4.26).

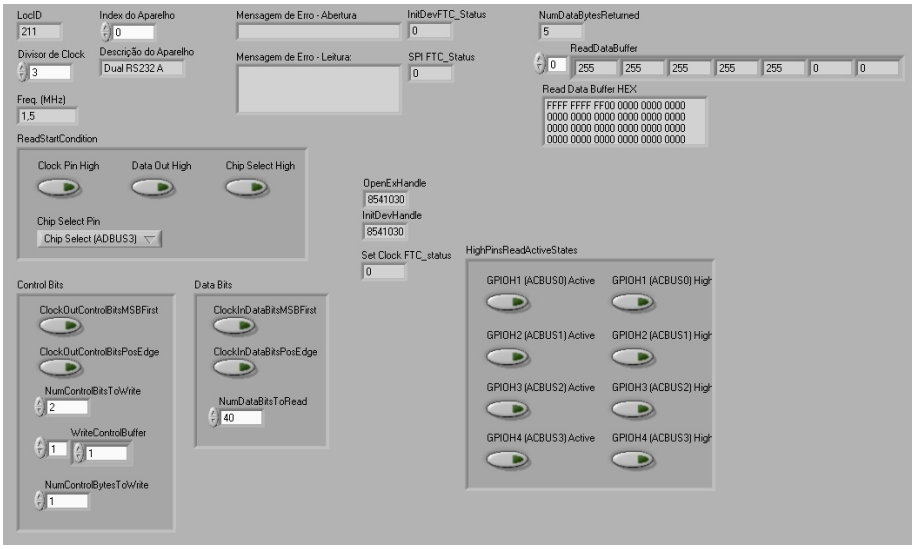


Figura 4.26: Tela do programa em LabVIEW®com os níveis altos em Read Data Buffer HEX.

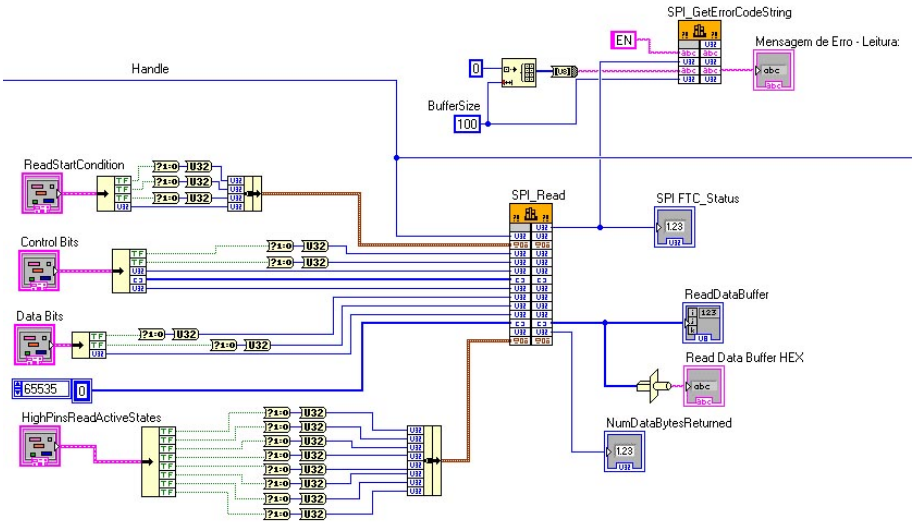


Figura 4.27: Programação em LabVIEW®do teste da função SPI_Read.

Posteriormente, foi também testada a função SPI_Write. Neste caso, tentamos fazer uma gravação via LabView e, como não obtivemos resultado satisfatório com a função SPI_Read, depois utilizamos o programa em Delphi para verificar se algo havia sido gra-

vado na EEPROM. Novamente os resultados não foram satisfatórios, visto que não houve qualquer alteração nos dados gravados anteriormente na memória.

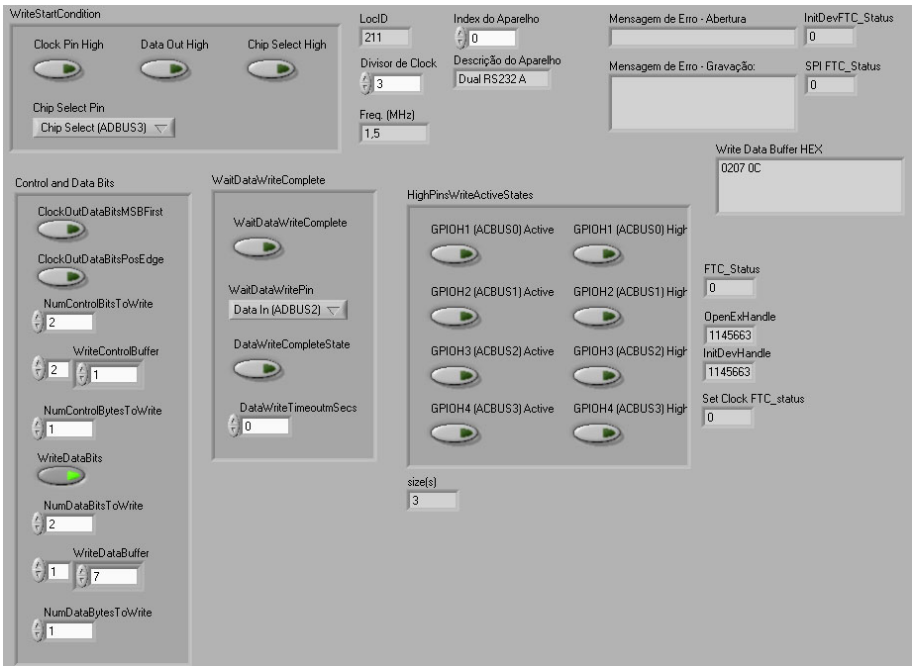


Figura 4.28: Tela do programa em LabVIEW®do teste da função SPI_Write.

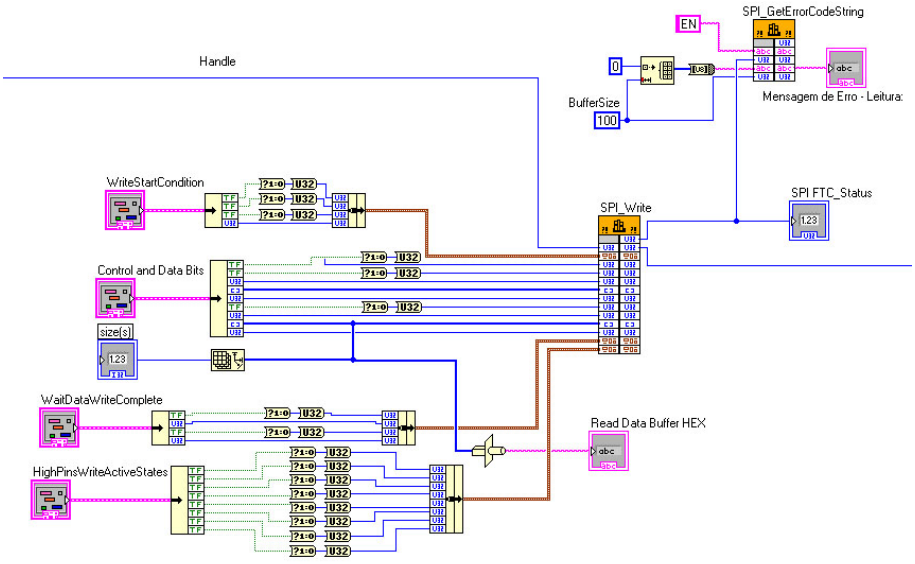


Figura 4.29: Programação em LabVIEW®do teste da função SPI_Write.

Vários testes foram realizados em LabVIEW®com a junção das funções mencionadas. Atulizamos um osciloscópio digital para verificar a comunicação, mas apesar de dados estarem sendo enviados para o FT2232D, não foi possível detectar transferência de dados entre o FT2232D e a EEPROM. A figura 4.30 possui 5V/divisão e 1ms/divisão.

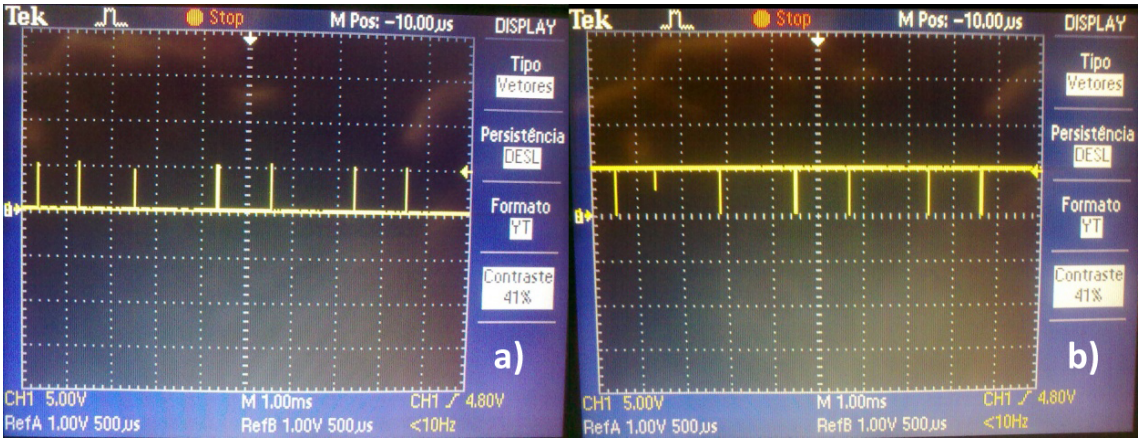


Figura 4.30: Em A) a visualização do sinal do pino In e em B) a visualização do sinal do pino Out.

Como resolver este problema (ou até criar um programa em Delphi) demandaria muito tempo e fugiria ao escopo do projeto, abandonamos o uso do FT2232D para programar e ler os registradores do AD9912.

4.5 Microcontrolador PIC 16f876a

Durante os testes com a versão final da placa com o microcontrolado PIC 16f876a observou-se que o circuito apenas realizava a sua função de programar o DDS no momento em que a fonte de alimentação do PIC era desligada. Após investigação do problema realizada por técnicos do LIEPO, foi constatado que era necessário colocar resistores de *pull-down* nos pinos da placa de avaliação AD9912/PCBZ que não estavam sendo utilizados - em especial o pino Master Reset. Após resolvido este problema, o circuito passou a executar sua função como o planejado.

CAPÍTULO 5: Conclusões e Observações

Durante o desenvolvimento deste projeto encontramos várias barreiras e verificamos a existência de novas possibilidades. Dentre as barreiras, a primeira foi a não existência de funções confiáveis para utilização do FT2332D em LabVIEW®por parte do fabricante. É possível inferir que dado um certo tempo para pesquisa e desenvolvimento de tais funções pudesse permitir o controle via LabVIEW®através da porta USB.

Além deste problema, que nos fez voltar para o controle via porta serial com protocolo RS-232, o maior problema observado foi a instabilidade do circuito utilizando PIC por razões até o momento desconhecidas. Tendo funcionado apenas em algumas ocasiões, acreditamos ser necessário que o circuito seja desenvolvido em uma placa de testes ao invés da junção entre protoboard e PBCZ antes que possa ser utilizado nos sistemas do relógio atômico.

Visto que o sistema tem grandes vantagens (principalmente no que diz respeito à compatibilidade com sistemas pré-existentes e ao custo baixo) em relação ao sistema utilizado atualmente, pode-se concluir que ele é uma escolha viável uma vez que a instabilidade for resolvida.

Com este projeto foi possível observar a grande liberdade na criação de dispositivos eletrônicos para interfaces. Através do uso de microcontroladores PIC da Microchip foi possível criar uma interface de baixo custo através de métodos simples de programação em linguagem C. Assim, para trabalhos futuros com o objetivo de utilização do modo PLL, acreditamos que possa ser útil refinar o controle da FTW no modo Single Tone para depois partir para o uso do modo PLL, que possui controle mais complexo.

Bibliografia

[Analog Devices, Inc. 2007-2010]ANALOG DEVICES, INC. *AD9910: 1 GSPS 14-Bit, 3.3 V CMOS Direct Digital Synthesizer Data Sheet (Rev. B)*. 2007–2010. Disponível em: <http://www.analog.com/static/imported-files/data_sheets/AD9910.pdf>.

[Analog Devices, Inc. 2007–2010]ANALOG DEVICES, INC. *AD9912: 1 GSPS Direct Digital Synthesizer with 14-Bit DAC (Rev. F)*. 2007–2010. Disponível em: <http://www.analog.com/static/imported-files/data_sheets/AD9912.pdf>.

[Carvalho e Pereira 2010]CARVALHO, R. A.; PEREIRA, T. R. *Diagnóstico da Placa AD9910 – Analog Devices*. nov 2010.

[Engineer Zone: AD9912 over SPI with PIC - IO Update Not Working 2011]ENGINEER Zone: AD9912 over SPI with PIC - IO Update Not Working. 2011. Disponível em: <http://ez.analog.com/message/43818>.

[Future Technology Devices International Ltd. 2009]FUTURE TECHNOLOGY DEVICES INTERNATIONAL LTD. *Application Note AN_111: Programmers Guide for High Speed FTCSPI DLL*. 2009. Disponível em: <http://www.ftdichip.com/Support/Documents/AppNotes/AN_111_Programmers_Guide_for_High_Sp

[Future Technology Devices International Ltd. 2011]FUTURE TECHNOLOGY DEVICES INTERNATIONAL LTD. *FT2232D: Dual USB to Serial UART/FIFO IC*. 2011. Disponível em: <http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS $_{F}T2232D.pdf$>.

[INMETRO 2001]INMETRO. *Sistema Internacional de Unidades*. 8. ed. [S.l.], 2001.

[Microchip Technology Inc. 2003]MICROCHIP TECHNOLOGY INC. *PIC16F87XA Data Sheet*. 2003. Disponível em: <http://ww1.microchip.com/downloads/en/devicedoc/39582b.pdf>.

[mikroElektronika 2003]MIKROELEKTRONIKA. *mikroC Users Manual*. 2003. Disponível em: <http://www.mikroe.com/pdf/mikroc/mikroc_manual.pdf>.

ANEXOS

ANEXO A: Resultados dos testes de amplitude, frequência e potência.

Este anexo se dedica a apresentar os resultados não mostrados no texto. Utilizamos para estes testes a saída não-filtrada J3.

A.1 Teste 2

Para o segundo teste, configuramos o clock externo para 200 MHz com nível de 5,00 dBm, system clock para 200 MHz, frequência de saída para 150 MHz no Profile 0, I/O Sync Clock Output Pin habilitado e variamos o DAC Gain Control e o ASF. Os resultados e as configurações de DAC Gain Control e ASF do segundo teste estão na tabela A.1 enquanto os cálculos de I_{OUT}, I_{SAIDA} e K com as equações 4.1, 4.2 e 4.3 estão presentes na tabela A.2.

DAC Gain Ctrl	DAC IOUT(mA)	ASF	Amplitude (V)	V_{RMS}	RF Envelope (dBm)	dBm para W
255	31,663125	0,25	0,0185	0,013081	-24,70	$3{,}38844 \cdot 10^{-6}$
255	31,663125	0,5	0,037	0,026163	-18,68	$1{,}35519 \cdot 10^{-5}$
255	31,663125	1	0,0742	0,052467	-12,67	$5{,}40754 \cdot 10^{-5}$
96	17,320000	0,25	0,01	0,007071	-30,01	$9{,}977 \cdot 10^{-7}$
96	17,320000	0,5	0,0202	0,014284	-24,00	$3{,}98107 \cdot 10^{-6}$
96	17,320000	1	0,0401	0,028355	-17,97	$1{,}59588 \cdot 10^{-5}$

Tabela A.1: Resultados do segundo teste de amplitude, frequência e potência.

I_{OUT} (A)	I_{SAIDA} (A)	K
0,007915781	0,000259026	30,55979953
0,015831563	0,00051798	30,56402313
0,031663125	0,00103065	30,72151884
0,00433	0,000141096	30,68830477
0,00866	0,000278717	31,07093079
0,01732	0,000562821	30,77352617

Tabela A.2: Cálculos do segundo teste de amplitude, frequência e potência

Assim, assumimos para este teste K=30,6.

A.2 Teste 3

Para o terceiro teste configuramos o clock externo para 400 MHz com nível de 5,00 dBm, mantivemos o system clock em 200 MHz, além frequência de saída para 150 MHz no Profile 0 e I/O Sync Clock Output Pin habilitado. Os resultados e as configurações de DAC Gain Control e ASF do terceiro teste estão na tabela A.3 e os cálculos de I_{OUT}, I_{SAIDA} e K estão presentes na tabela A.4.

DAC Gain Ctrl	DAC IOUT(mA)	ASF	Amplitude (V)	V_{RMS}	RF Envelope (dBm)	dBm para W
255	31,663125	0,25	0,0185	0,013081	-24,69	$3{,}39625 \cdot 10^{-6}$
255	31,663125	0,5	0,0368	0,026022	-18,67	$1{,}35831 \cdot 10^{-5}$
255	31,663125	1	0,0738	0,052184	-12,64	$5{,}44503 \cdot 10^{-5}$
96	17,320000	0,25	0,0100	0,007071	-29,99	$1{,}00231 \cdot 10^{-6}$
96	17,320000	0,5	0,0200	0,014142	-23,96	$4{,}01791 \cdot 10^{-6}$
96	17,320000	1	0,0402	0,028426	-17,93	$1{,}61065 \cdot 10^{-5}$

Tabela A.3: Resultados do terceiro teste de amplitude, frequência e potência.

I_{OUT} (A)	I_{SAIDA} (A)	K
0,007915781	0,000259623	30,48951394
0,015831563	0,000521996	30,32889687
0,031663125	0,001043419	30,34555881
0,00433	0,000141747	30,54730482
0,00866	0,000284109	30,48125821
0,01732	0,000566616	30,56743117

Tabela A.4: Cálculos do terceiro teste de amplitude, frequência e potência

Assim, assumimos para este teste K=30,5.

A.3 Teste 4

Para o quarto testemantivemos o clock externo em 400 MHz com nível de 5,00 dBm, o system clock em 200 MHz, além do I/O Sync Clock Output Pin habilitado. Alteramos

apenas a frequência de saída para 100 MHz no Profile 0. Os resultados e as configurações de DAC Gain Control e ASF do quarto teste estão na tabela A.5 e os cálculos de I_{OUT}, I_{SAIDA} e K estão presentes na tabela A.6.

DAC Gain Ctrl	DAC IOUT(mA)	ASF	Amplitude (V)	V_{RMS}	RF Envelope (dBm)	dBm para W
255	31,663125	0,25	0,066	0,046669	-13,60	$4{,}36516 \cdot 10^{-5}$
255	31,663125	0,5	0,132	0,093338	-7,58	0,000174582
255	31,663125	1	0,264	0,186676	-1,56	0,000698232
96	17,320000	0,25	0,036	0,025456	-18,90	$1{,}28825 \cdot 10^{-5}$
96	17,320000	0,5	0,072	0,050912	-12,88	$5{,}15229 \cdot 10^{-5}$
96	17,320000	1	0,144	0,101823	-6,85	0,000206538

Tabela A.5: Resultados do quarto teste de amplitude, frequência e potência.

I_{OUT} (A)	I_{SAIDA} (A)	K
0,007915781	0,000935343	8,462968183
0,015831563	0,001870428	8,464137831
0,031663125	0,00374034	8,46530684
0,00433	0,000506072	8,556091085
0,00866	0,001012005	8,557273064
0,01732	0,002028395	8,538771305

Tabela A.6: Cálculos do quarto teste de amplitude, frequência e potência

Assumimos K=8,5 para este teste. Como K mudou de 30,5 para 8,5 concluímos que K depende da frequência de saída e não do clock do sistema.

A.4 Teste 5

Para o quinto teste, configuramos o clock externo para 800 MHz com nível de 5,00 dBm, system clock para 400 MHz, frequência de saída para 200 MHz no Profile 0, I/O Sync Clock Output Pin habilitado. Os resultados e as configurações de DAC Gain Control e ASF do quinto teste estão na tabela A.7 enquanto os cálculos de I_{OUT}, I_{SAIDA} e K estão presentes na tabela A.8.

DAC Gain Ctrl	DAC IOUT(mA)	ASF	Amplitude (V)	V_{RMS}	RF Envelope (dBm)	dBm para W
255	31,663125	0,25	0,156	0,110309	-6,14	0,00024322
255	31,663125	0,5	0,312	0,220617	-0,12	0,000972747
255	31,663125	1	0,402 [1]	-	Max 5,07 Mín 2,08	Max 0,003213661 Mín 0,001614359
96	17,320000	0,25	0,081	0,057276	-12,82	$5{,}22396 \cdot 10^{-5}$
96	17,320000	0,5	0,162	0,114551	-5,80	0,000263027
96	17,320000	1	0,325	0,22981	0,23	0,001054387

Tabela A.7: Resultados do quinto teste de amplitude, frequência e potência.

I_{OUT} (A)	I_{SAIDA} (A)	K
0,007915781	0,002204908	3,590073755
0,015831563	0,004409206	3,590569932
0,031663125	-	-
0,00433	0,000912074	4,747422868
0,00866	0,002296149	3,771532971
0,01732	0,004588087	3,774993871

Tabela A.8: Cálculos do quinto teste de amplitude, frequência e potência

Assumimos K=3,6.

A.5 Teste 6

Com o teste 5, verificamos o valor de K voltar a se aproximar de 5, valor encontrado em 50 MHz. Verificaremos se o K passa a se manter próximo de 5 para valores superiores a 200 MHz para depois diminuirmos os intervalos de medição entre 50 MHz e 200 MHz. Mantivemos o clock externo em 800 MHz com nível de 5,00 dBm, system clock em 400 MHz e I/O Sync Clock Output Pin habilitado. Mudamos a frequência de saída para 250 MHz no Profile 0. Os resultados e as configurações de DAC Gain Control e ASF do sexto teste estão na tabela A.9 enquanto os cálculos de I_{OUT}, I_{SAIDA} e K estão presentes na tabela A.10.

[1] Valor cortado.

DAC Gain Ctrl	DAC IOUT(mA)	ASF	Amplitude (V)	V_{RMS}	RF Envelope (dBm)	dBm para W
255	31,663125	0,25	0,051	0,036062	-15,82	$2{,}61818 \cdot 10^{-5}$
255	31,663125	0,5	0,102	0,072125	-9,81	0,000104472
255	31,663125	1	0,205	0,144957	-3,79	0,00041783
96	17,320000	0,25	0,028	0,019799	-21,09	$7{,}78037 \cdot 10^{-6}$
96	17,320000	0,5	0,056	0,039598	-15,07	$3{,}11172 \cdot 10^{-5}$
96	17,320000	1	0,112	0,079196	-9,05	0,000124451

Tabela A.9: Resultados do sexto teste de amplitude, frequência e potência.

I_{OUT} (A)	I_{SAIDA} (A)	K
0,007915781	0,000726014	10,90307372
0,015831563	0,001448488	10,92971827
0,031663125	0,002882446	10,98481226
0,00433	0,000392968	11,01871449
0,00866	0,000785827	11,02023666
0,01732	0,001571437	11,02175905

Tabela A.10: Cálculos do sexto teste de amplitude, frequência e potência

Assumimos K=11.

A.6 Teste 7

Mantivemos o clock externo em 800 MHz com nível de 5,00 dBm, system clock em 400 MHz e I/O Sync Clock Output Pin habilitado. Mudamos a frequência de saída para 300 MHz no Profile 0. Os resultados e as configurações de DAC Gain Control e ASF do sétimo teste estão na tabela A.11 enquanto os cálculos de I_{OUT}, I_{SAIDA} e K estão presentes na tabela A.12.

DAC Gain Ctrl	DAC IOUT(mA)	ASF	Amplitude (V)	V_{RMS}	RF Envelope (dBm)	dBm para W
255	31,663125	0,25	0,018	0,012728	-24,94	$3{,}20627 \cdot 10^{-6}$
255	31,663125	0,5	0,036	0,025456	-18,92	$1{,}28233 \cdot 10^{-5}$
255	31,663125	1	0,072	0,050912	-12,89	$5{,}14044 \cdot 10^{-5}$
96	17,320000	0,25	0,033	0,023335	-19,70	$1{,}07152 \cdot 10^{-5}$
96	17,320000	0,5	0,066	0,046669	-13,68	$4{,}28549 \cdot 10^{-5}$
96	17,320000	1	0,131	0,092631	-7,64	0,000172187

Tabela A.11: Resultados do sétimo teste de amplitude, frequência e potência.

I_{OUT} (A)	I_{SAIDA} (A)	K
0,007915781	0,000251908	31,42326281
0,015831563	0,000503747	31,42760575
0,031663125	0,001009677	31,35965484
0,00433	0,000459199	9,429460344
0,00866	0,000918271	9,430762974
0,01732	0,001858847	9,31760264

Tabela A.12: Cálculos do sétimo teste de amplitude, frequência e potência

Pela primeira vez, K assumiu dois valores distintos.