ANDRÉ LUIZ ZIMMERMANN

SISTEMA EMBARCADO DE PROCESSAMENTO DIGITAL PARA INSPEÇÃO ULTRASÔNICA DE DUTOS

São Paulo
2009

ANDRÉ LUIZ ZIMMERMANN

SISTEMA EMBARCADO DE PROCESSAMENTO DIGITAL PARA INSPEÇÃO ULTRASÔNICA DE DUTOS

MONOGRAFIA APRESENTADA À ESCOLA POLITÉCNICA PARA OBTENÇÃO DO TÍTULO DE ENGENHEIRO

ÁREA DE CONCENTRAÇÃO: ENGENHARIA MECATRÔNICA

ORIENTADOR:
PROF. DR. CELSO MASSATOSHI FURUKAWA

São Paulo
2009

FICHA CATALOGRÁFICA

Zimmermann, André Luiz
Sistema embarcado de processamento digital para inspeção ultrasônica de dutos / A.L. Zimmermann. -- São Paulo, 2009.
79 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1. Processamento digital de sinais 2. Eletrônica embarcada I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II. t.

AGRADECIMENTOS

Agradeço à minha mãe, por todo o apoio que recebi em toda a minha vida, em especial durante o período de graduação. Sem seu apoio esta etapa teria sido muito mais difícil.

Agradeço à minha namorada, por toda a ajuda que recebi e por tornar este um período muito mais agradável.

Agradeço ao meu orientador, Prof. Dr. Celso M. Furukawa, pela firme orientação e por todo o aprendizado que me propiciou durante a execução deste trabalho.

Agradeço ao Eng. Me. Douglas D. S. Santana, por todo o apoio e conhecimento proporcionados ao longo deste trabalho.

Agradeço à ANP – Agência Nacional do Petróleo, Gás Natural e Biocombustíveis - e ao PRH-19 pelo apoio financeiro ao projeto.

“Nenhum homem realmente produtivo pensa como se estivesse escrevendo uma dissertação.”

- Albert Einstein.

RESUMO

O presente trabalho tem como objetivo o estudo e desenvolvimento de parte de um sistema eletrônico embarcado para dispositivos de inspeção de dutos metálicos, capazes de detectar pontos de corrosão. A técnica estudada baseia-se na emissão de sinais ultra-sônicos e recepção destes sinais após sua reflexão nas interfaces interna e externa das paredes dos dutos. A estimativa de profundidade das cavidades erodidas é efetuada analisando-se o *TOA* (*Time of Arrival*) ou em português, tempo de vôo do sinal emitido, de forma que com medições sucessivas, pode-se traçar os perfis de relevo interno e de espessura das paredes ao longo do duto. Neste trabalho são abordados parte da estrutura de hardware que será embarcada em PIGs e alguns dos algoritmos aplicados no tratamento de sinais. O algoritmo de detecção do *TOA* tem como base o modelo matemático conhecido como filtro casado, associado à modulação de fase do sinal enviado, visando o aumento de eficácia do método. A implemetação dos algoritmos é efetuada através de hardware dedicado para processamento em tempo real, conhecido como *DSP* (*Digital Signal Processing*) implementado em *FPGA* (*Field Programmable gate array*) de alta densidade, usando linguagem *VHDL* de descrição de circuitos.

Palavras-chave: *PIG* (*Pipeline Inspection Gauge*), Inspeção por Ultra-Som, Eletrônica Embarcada, DSP (*Digital Signal Processing*).

ABSTRACT

This work describes the development of an ultrasonic system for anti-corrosion inspection of metal pipelines. A maximum likelihood (ML) technique is used to improve the estimation of to the TOA (Time of Arrival) of multiple echoes, in order to evaluate the thickness of the wall directly from the difference between the first two echoes. This work describes the development of a complete system, including the entire hardware structure which will be loaded on PIGS, and the ML algorithm applied in signal processing. The TOA estimation algorithm is based on a mathematical model known as Matched Filter or North Filter, combined with phase modulation of the signal, known as BPSK (Binary Phase Shift Keying). The algorithm is fully implemented in dedicated hardware for real time processing using high density FPGA (Field Programmable gate array).

Keywords: PIG (Pipeline Inspection Gauge) Inspection by ultrasound, Embedded Systems, DSP (Digital Signal Processing).

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1. INTRODUÇÃO

A corrosão é a principal causa de deterioração de dutos usados no transporte de petróleo e gás natural. Muitos esforços têm sido despendidos no intuito de prolongar a vida útil destas instalações, e ao mesmo tempo evitar gastos excessivos com paradas de manutenção. Neste sentido, o conhecimento sobre a taxa de corrosão de dutos instalados pode permitir um planejamento eficiente de manutenção e ainda estender a vida útil das instalações, com segurança, até que as mesmas deixem de apresentar condições adequadas de funcionamento. Considerando o enorme investimento que os dutos modernos representam, o aumento na sua vida útil pode representar um ganho considerável nos rendimentos deste capital investido.

Outro aspecto importante a se destacar é a finalidade preventiva deste projeto. Considerando a importância da preservação do meio ambiente como fator indispensável ao desenvolvimento sustentável, este projeto ganha enorme relevância, dada a expansão do setor de petróleo e gás e o consequente crescimento da malha de oleodutos e gasodutos instalados em todo o território nacional. Com a aplicação de métodos mais precisos e confiáveis, possíveis vazamentos podem ser previstos evitando catástrofes ambientais como o vazamento do oleoduto OSPAR em 2000.

O *PIG* (*Pipeline Inspection Gauge*) é o principal equipamento utilizado na inspeção de dutos, sendo voltado para a detecção de falhas como trincas e pontos de perda de material por processos corrosivos. As configurações possíveis para o uso de *PIGs* em inspeções e análises é diversa, e depende diretamente da instrumentação instalada no mesmo. Os sensores mais comuns embarcados em *PIGs* são os sensores táteis, sensores de vazamento de fluxo magnético (*MFL-Magnetic Flux Leakage*) e sensores ultra-sônicos. Estas técnicas vêm sendo estudadas e aplicadas por mais de 40 anos, dentre as quais se destaca aquela com o uso de transdutores de ultra-som por seus bons resultados, principalmente devido a avanços recentes nesta área (Canales, et al, 2007).

2. OBJETIVOS

Este trabalho destina-se ao desenvolvimento de parte do aparato eletrônico embarcado de dispositivos aplicados na inspeção de dutos metálicos por ondas acústicas de ultra-som, trabalhando com a técnica de pulso-eco.

Dentro deste escopo, objetiva-se atingir um refinamento da inspeção via ultra-som, em relação à abordagem analógica tradicional. Estes possíveis refinamentos serão estudados e testados, a princípio, com a aplicação de novas tecnologias aliadas a ferramentas matemáticas/estatísticas conhecidas, as quais são descritas a seguir:

- Substituição dos circuitos analógicos por circuitos digitais para a emissão, recepção e tratamento dos sinais. Objetiva-se a aplicação de *DSP (Digital Signal Processing)* com o uso de *FPGA* (*Field Programmable gate array*) para o processamento de sinais em tempo real, de forma a reduzir o volume de memória necessária ao armazenamento de dados, sem queda na taxa de amostragem.
- Aplicação de Filtro Casado no cálculo do tempo de vôo do sinal, visando diminuir a susceptibilidade de falhas de leitura devido a ruídos, ecos laterais e atenuação.

O foco deste projeto será concentrado nos seguintes tópicos:

a) Estudo, desenvolvimento e realização de testes do módulo eletrônico Pulsador/Receptor, incluindo *hardware* e *firmware*.
   Este módulo é responsável pela geração de sinais elétricos de controle dos transdutores de ultra-som e pela recepção, pré-processamento e amplificação dos sinais de retorno proveniente dos ecos.
b) Implementação de Filtro Casado via *DSP*.

3. TÉCNICAS DE INSPEÇÃO: PIGs

Devido aos diâmetros relativamente pequenos dos dutos padrão atualmente utilizados (200 mm – 500 mm), e ao fato de oleodutos e gasodutos normalmente se estenderem por vários quilômetros, estando normalmente, submersos ou enterrados, a inspeção *in situ* é muito difícil e, por vezes, impossível. Por esta razão, inspeções em dutos são usualmente realizadas com equipamentos conhecidos como *PIGs* [1].

Um *PIG (Pipeline Inspection Gauge)* é um aparelho utilizado na inspeção interna de dutos de transporte de material fluido, sendo voltado para a detecção de falhas, como trincas, e pontos de perda de material por processos corrosivos ao longo das paredes de tubos como aqueles usados no transporte de óleos fósseis e gases. Um PIG de inspeção padrão pode ser observado na figura 1.

Figura 1: Fotografia de um PIG instrumentado típico.

Existem ainda outras aplicações para PIGs como limpeza de dutos, reparos, reaplicação de revestimentos internos, etc. Estas outras aplicações não serão alvo de estudo do presente trabalho.

Os *PIGs* de inspeção podem apresentar duas formas básicas:

- *PIGs* “livres”: São os *PIGs* que contém todo o equipamento necessário à execução da inspeção, incluindo baterias, sistema de aquisição de dados, etc. Estes *PIGs* são, normalmente, impulsionados pelo próprio fluido que percorre a tubulação.
- *PIGs* com “cordão umbilical”: São *PIGs* que apresentam apenas a estrutura mecânica mínima para a inspeção, na maior parte das vezes incluindo pouco mais do que os sensores utilizados. Este tipo de *PIG* é conectado a um cabo que fornece a energia para seu funcionamento, além da conexão com o sistema de aquisição de dados que fica instalado em terra. Os *PIGs* com cordão umbilical podem ser impulsionados tanto pelo fluido do duto quanto por motores próprios, dependendo do projeto.

Ambas as configurações apresentam vantagens e desvantagens:

Os *PIGs* livres têm a vantagem de não necessitar dos cabos, o que agrega mais facilidade em seu manuseio, além de não apresentarem o risco de falha na inspeção por rompimento do cabo ou falhas de conexão.

Já os *PIGs* com cordão umbilical podem apresentar formatos menores, o que permite seu uso em instalações nas quais um *PIG* livre não poderia ser utilizado devido ao seu tamanho, como tubulações que apresentam curvas muito acentuadas ou válvulas ao longo do caminho.

As configurações possíveis para o uso de *PIGs* em inspeções é diversa, e depende diretamente da instrumentação instalada no mesmo.

Os sensores mais comuns embarcados em *PIGs* são os sensores táteis, sensores de vazamento de fluxo magnético (*MFL-Magnetic Flux Leakage*) e sensores por transdutores de ultra-som. Estas técnicas vêm sendo estudadas e aplicadas por mais de 40 anos [1], dentre as quais se destaca aquela com o uso de transdutores de ultra-som por seus bons resultados, principalmente devido a avanços recentes nesta área [1, 2]. Por esta razão, a aplicação de sensores por transdutores de ultra-som será empregada neste trabalho, em detrimento das demais.

4. INSPEÇÃO POR SINAIS ULTRA-SÔNICOS

Atualmente existem técnicas diversas que utilizam ondas acústicas, em especial o ultra-som, em aplicações industriais de inspeção. Entre estas técnicas, merecem destaque o *ToFD (Time of Flight Difraction), PA (Phased Array)* e Pulso-Eco.

Em qualquer destas técnicas, o esquema básico de aplicações de inspeção por ultra-som é composto pelo corpo a ser inspecionado, fluido de acoplamento, transdutor, cabos de conexão e, por fim, o sistema eletrônico de processamento de sinais.

O método de inspeção por ultra-som aplicado neste trabalho é conhecido como método **pulso-eco**, cujas características serão elucidadas neste capítulo com mais detalhes.

O método de inspeção não-destrutiva por ultra-som pulso-eco consiste em emitir sinais acústicos conhecidos nas faixas de freqüência do ultra-som, através do fluido (petróleo, gás, etc.) e efetuar as leituras dos ecos do sinal refletido nas paredes do duto, como ilustra a figura 2.

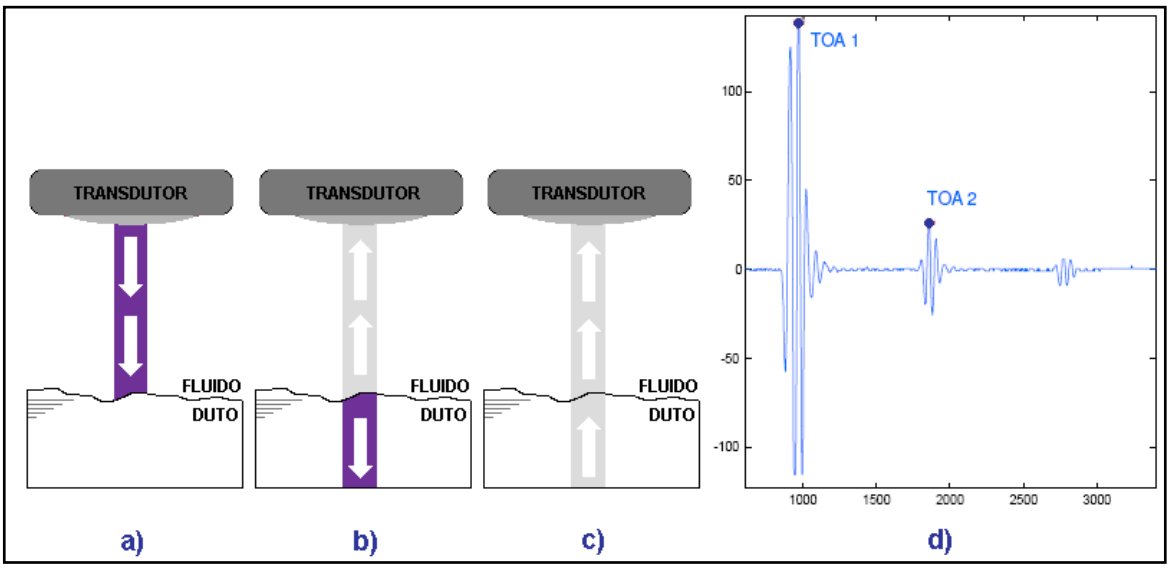


Figura 2: Inspeção por ultra-som através da técnica pulso-eco. a) Emissão do sinal de ultra-som; b) reflexão e transmissão do sinal na interface interna do duto; c) Reflexão do sinal na interface externa do duto; d) Gráfico indicando os dois primeiros ecos, provenientes das reflexões nas interfaces interna e externa do duto, respectivamente.

Note que, ao atingir o limite entre o fluido e a superfície interna da parede do duto, parte do sinal é refletida, gerando o 1º eco, que fornece informação sobre o perfil interno do duto, e parte do sinal é transmitida para o interior da parede metálica, onde ocorrem novos ciclos de transmissão e reflexão, gerando os ecos subseqüentes. A primeira reflexão na parede externa do duto gera o segundo eco, que fornece informação sobre a espessura do duto no ponto de inspeção. O instante de chegada de cada um dos sinais de eco ao transdutor é conhecido como *TOA*, do inglês *Time of Arrival*.

Uma vez conhecidas as velocidades de propagação da onda acústica no fluido e no metal, pode-se estimar o perfil interno e os valores de espessura das paredes dos dutos a partir dos dois primeiros ecos captados. O perfil interno é obtido a partir do *TOA* 1 e a espessura é obtida tomando a diferença entre os *TOAs* 1 e 2.

A propagação de ondas acústicas pode ser prevista com precisão em meios homogêneos e bem conhecidos. Nesses meios, a reflexão das ondas em superfícies metálicas planas e polidas é praticamente especular, gerando sinais de eco intensos e muito próximos aos previstos pela teoria. Porém, quando se trabalha fora das condições controladas de laboratório com sinais de eco gerados por superfícies corroídas e rugosas, em meios que apresentam alta atenuação, a relação sinal/ruído do eco detectado é fortemente prejudicada.Com estes sinais recebidos de regiões corroídas e em meios com alta atenuação , como o petróleo, a estimativa do *TOA* se torna mais imprecisa, e desta maneira o resultado da inspeção torna-se menos confiável. Daí vem a necessidade do desenvolvimento de técnicas que melhorem a precisão da estimativa do *TOA.*

5. TÉCNICAS DE GERAÇÃO E TRATAMENTO DIGITAL DE SINAIS

Fenômenos como atenuação e ruido comprometem a qualidade da inspeção, diminuindo a precisão e a confiabilidade dos resultados. Estes problemas são especialmente relevantes no caso dos dutos de transporte de petróleo, onde ocorre acúmulo significativo de resíduos nas paredes, além do alto índice de atenuação do petróleo, dificultando a captação dos ecos decorrentes do sinal. Neste contexto evidenciam-se as vantagens do emprego de uma abordagem digital para se obter resultados melhores e mais confiáveis em contraposição à abordagem analógica do problema.

A técnica analógica tradicional envolve a captação dos ecos dos pulsos gerados através de transdutores de ultra-som, que transformam estas ondas em sinais elétricos, e por sua vez são comparados num **detector de limiar**, ou *Threshold Detector*, em inglês. O *TOA* é então computado quando o nível do sinal gerado atinge o limiar de comparação. Acontece, porém, que este método de estimação é extremamente suscetível a erros de leitura devidos à incidência de ruído e à atenuação da onda no meio de propagação, como se pode observar na figura 3.

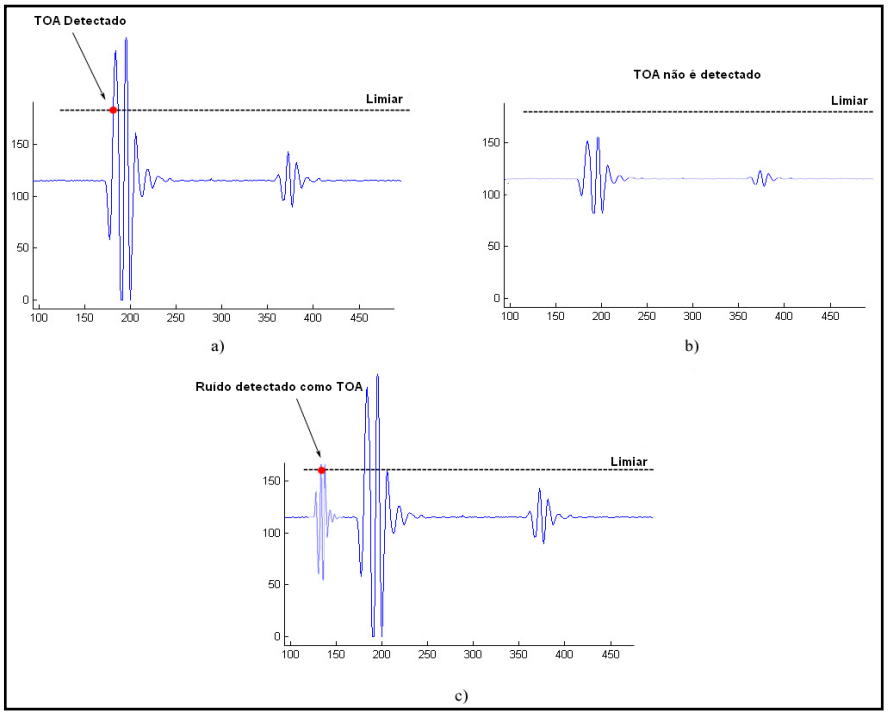


Figura 3: Detecção do TOA em (a) e falha em (b) e (c) por atenuação e ruído; Fonte: [1].

Em contraposição à abordagem analógica, atualmente a aplicação de técnicas digitais apresenta grandes vantagens do ponto de vista técnico associadas a baixo custo de implementação. Dentre estas vantagens destaca-se a possibilidade de refinamento dos resultados estimando os *TOAs* através de Modulação de Fase do sinal com **Códigos de Barker** associados ao uso de **Filtros Casados** para a detecção do tempo de vôo dos sinais. A aplicação destas técnicas viabiliza leituras precisas mesmo em condições de alto ruído e elevada atenuação [2].

Para o problema básico de recepção do eco, pode-se considerar o seguinte modelo matemático:

$$r(t) = C.s(t-\tau) + w(t), \qquad (1)$$

para $0 \leq t \leq T$, onde

r(t) – Sinal recebido

s(t) – Sinal enviado

C – Constante de escalamento devido à atenuação (C<1)

τ – Atraso do sinal devido ao tempo de vôo (*TOA*)

w(t) – Ruído branco gaussiano

T – Período total de observação.

Com este modelo o problema de estimação do *TOA* passa a ser estimar o valor de τ com o menor erro possível a partir da leitura *r(t)*. A estimação de um parâmetro determinístico através da observação de variáveis aleatórias pode ser feita aplicando-se o critério de Máxima Verossimilhança (*ML-Maximum Likelihood*), que se baseia na maximização de uma função densidade de probabilidade que neste caso relaciona os valores lidos de *r(t)* aos valores prováveis de τ. Pode-se demonstrar [3,4] que a máxima verossimilhança ocorre para o valor de τ que maximiza a expressão

$$\Lambda(r(t),\tau) = \int_0^T r(u)s(u-\tau)du\,. \qquad (2)$$

A expressão (2) é equivalente a uma integral de convolução entre o sinal *r(t)* e o sinal enviado, *s(t),* invertido no tempo, ou seja, *s(-t).* Como esta função representaria fisicamente um sistema não-causal, é necessária a introdução de uma constante t_f, de forma que *s(-t)* passe a ser $s(t_f - t)$, onde t_f é a duração do sinal. Desta forma o resultado da convolução passa a ser a estimativa do *TOA* somada a um atraso fixo conhecido. Pode-se ainda interpretar que o problema é dado por um sistema linear cuja entrada é o sinal *r(t)*, a função de transferência é $s(t_f - t)$ e a saída é uma função, cujo maior pico fornece a estimativa do *TOA*. A função de transferência deste sistema linear é conhecida como **Filtro Casado**:

$$h(t) = k.s(t_f - t) \qquad (3)$$

Ou seja, o filtro casado é uma função que corresponde ao sinal que se deseja captar invertido no tempo, atrasado e multiplicado por uma constante, como se pode observar na figura 4.

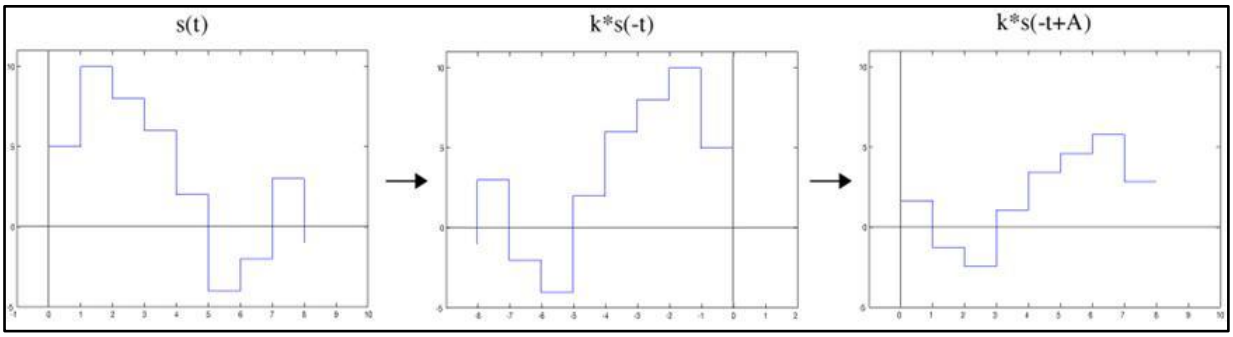


Figura 4: Sinal original (esq.), Invertido no tempo (centro), Invertido, atrasado e escalado (dir.).

Em resumo, segundo o modelo adotado, o uso do filtro casado permite que se obtenha uma estimativa estatística do tempo de vôo do sinal enviado através do ponto de máximo da sua função de autocorrelação.

A figura 5 apresenta um exemplo de sinal imerso em ruído e a função de saída da convolução com seu filtro casado, indicando o pico que fornece o *TOA* do sinal recebido. No gráfico da esquerda a linha cheia representa o sinal imerso em ruído e a linha tracejada representa o sinal original.

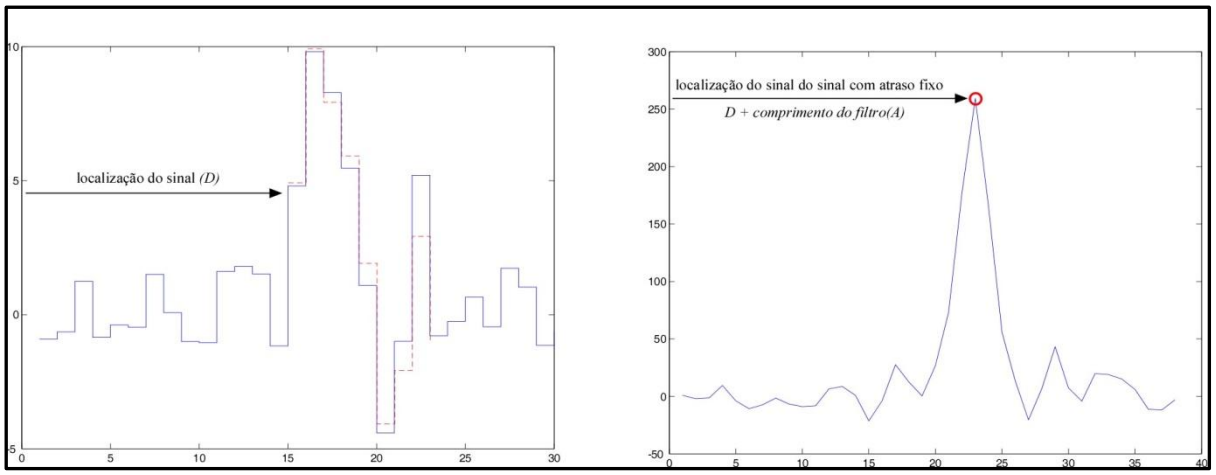


Figura 5: Sinal enviado imerso em ruído (esq.), e convolução entre o filtro casado e o sinal recebido (dir.).

É neste contexto que se encontra a importância da modulação do sinal enviado, pois com esta técnica é possível obter sinais cuja convolução com seu filtro casado apresenta picos mais estreitos e elevados do que os picos apresentados por sinais de ultra-som típicos. Matematicamente, isso significa dizer que com o uso de modulação pode-se obter sinais cuja função de autocorrelação apresenta picos distintos, apropriados a esta aplicação.

Neste caso, a modulação ótima seria aquela que proporcionasse ao sinal enviado valor máximo de autocorrelação no instante do *TOA* e valor mínimo em todo o restante da aquisição.

Os métodos de modulação de sinais são numerosos e bastante diversos, sendo os mais conhecidos a modulação de amplitude (*AM*), a modulação de frequência (*FM*) e a modulação de fase do sinal (*PM*).

Neste trabalho a técnica de modulação aplicada é conhecida como *Binary Phase Shift Keying (BPSK)* sendo, portanto, um tipo de modulação de fase, e consiste em inverter a fase da onda portadora durante certos períodos de acordo com o código de modulação utilizado. A figura 6 mostra um exemplo de onda portadora cuja fase foi modulada por um código binário. Note que quando o valor do código é 1 a fase da onda portadora é somada de zero, e quando o valor do código é -1 a fase da onda portadora é somada de π radianos.

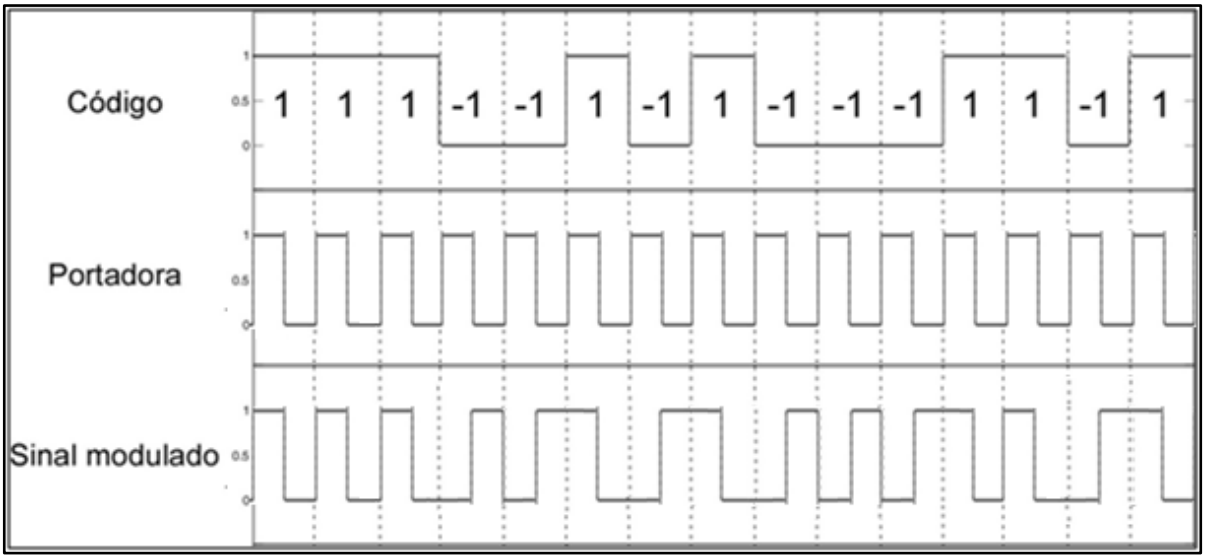


Figura 6: BPSK com um código qualquer sendo aplicado à onda portadora.

Nesta aplicação serão utilizados Códigos de Barker para a modulação dos sinais. Os Códigos de Barker são seqüências binárias cuja função de autocorrelação apresenta um pico bastante acentuado num ponto e valores mínimos no restante do domínio. Atualmente são conhecidos nove códigos com estas características, os quais são apresentados na tabela 1.

Número de Bits	Código
2	+1 -1 /+1+1
3	+1 +1 -1
4	+1 -1 +1 +1 /+1-1-1-1
5	+1 +1 +1 -1 +1
7	+1 +1 +1 -1 -1 +1 -1
11	+1 +1 +1 -1 -1 -1 +1 -1 -1 +1 -1
13	+1 +1 +1 +1 +1 -1 -1 +1 +1 -1 +1 -1 +1

Tabela 1: Os Códigos de Barker atualmente conhecidos.

Outra vantagem da aplicação de circuitos digitais nesse contexto é a possibilidade de se conseguir uma redução drástica no volume de dados a serem armazenados. Esta redução é possível graças à possibilidade de se implementar circuitos capazes de processar as informações colhidas durante os intervalos de aquisição, ou seja, os dados são processados, e armazenados em tempo real, otimizando o volume de memória utilizada.

A redução no volume de dados é efetuada utilizando-se circuitos digitais detectores de picos, de forma que penas os picos detectados são armazenados em memória,

ou seja, com a aplicação de circuitos detectores de picos não há a necessidade de se armazenar em memória todos os dados amostrados. A figura 7 mostra um sinal submetido ao detector de picos por duas vezes seguidas. Na primeira detecção, o detector reconhece os picos locais, e na segunda detecção são encontrados os picos da envoltória do sinal.

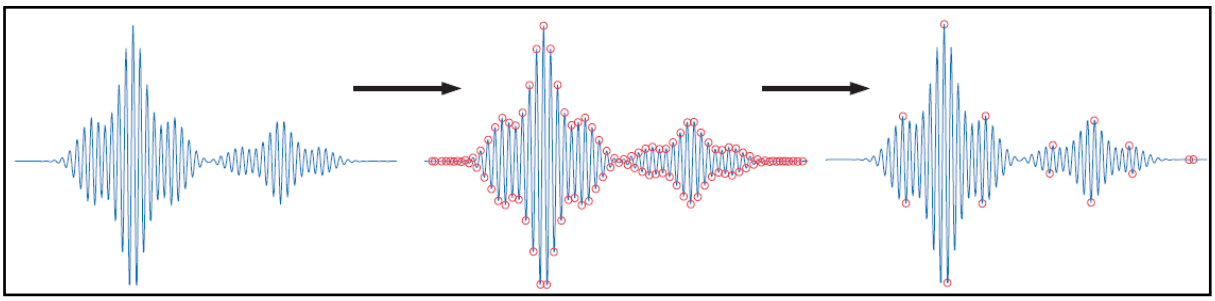


Figura 7: Filtragem dupla pelo detector de picos reduzindo um conjunto de 600 pontos para um conjunto de 14 pontos.

É importante observar que a implementação dos algoritmos até aqui discutidos é possível devido à aplicação de *DSP (Digital Signal Processing)*, e também a características particulares do sistema digital, que são apresentadas no capítulo a seguir.

6. A ESTRUTURA DO SISTEMA ELETRÔNICO EMBARCADO

O projeto do sistema de inspeção foi desenvolvido para operar sobre uma plataforma totalmente baseada em *DSP* (*Digital Signal Processing*). A escolha pela implementação de circuitos dedicados decorre da necessidade de se processar em tempo real toda a informação proveniente da inspeção, já que o volume de dados é consideravelmente grande (lembrando que o aparato deve operar por longos períodos dentro da tubulação), o que inviabiliza a estratégia de armazenamento em memória e posterior processamento. Da mesma forma, o processamento em tempo real por *software* não é viável, uma vez que com os microprocessadores atualmente disponíveis não é possível e nem mesmo prático desenvolver um arranjo pequeno e eficiente o suficiente para que o conjunto eletrônico mais as baterias necessárias caibam no limitado espaço físico disponível no *PIG* e ao mesmo tempo sejam capazes de processar sinais adquiridos a taxas de 50 a 100MS/s, taxas estas necessárias para a reconstrução de sinais gerados por transdutores ultra-sônicos com frequência central de 5 a 10MHz.

O sistema eletrônico completo desenvolvido para aplicação em *PIGs* ultra-sônicos segue uma arquitetura modular dividida em três blocos principais:

- **Módulo Pulsador/Receptor (MPR)**: Tem a função de gerar os pulsos elétricos de controle dos transdutores e condicionar os sinais de eco dos pulsos enviados. Tem ainda a função de amplificar o sinal recebido e transmiti-lo ao MAPD.
- **Módulo de Aquisição e Processamento Digital (MAPD)**: Utilizado para digitalizar o sinal de eco e processá-lo em tempo real para a detecção dos *TOAs*.
- **Módulo CPU(MCPU)**: Utilizado para gerenciar e configurar os MAPD e MPR. Este módulo permite ainda a comunicação externa do sistema e recuperação de dados armazenados.

O projeto dos módulos foi baseado no padrão de dimensões *PC/104* para a placa e conectores *PC/104 ISA Bus* de 16 bits de forma que estes módulos podem ser conectados diretamente a computadores *PC/104* disponíveis no mercado.

6.1. O PC/104

A arquitetura *PC/104* teve sua primeira especificação publicada em 1992. Suas principais características são: sistemas compactos, modulares, e compatíveis com PC, oferecendo uma variedade de funções com suporte de diversos fornecedores. Além disso, existe um Consórcio para manter e aprimorar os padrões técnicos, além de promover a visibilidade do *PC/104* na indústria ao redor do mundo.

O *PC/104* é simplesmente uma reformulação em versão modular da arquitetura *PC* destinada a aplicações embarcadas onde o espaço, consumo de energia e confiabilidade são críticos. Estes módulos podem servir como expansão para um *SBC (Single Board Computer)* ou podem ser aplicados como computadores inteiros que não requerem uma *backplane board*, como mostra a figura 8.

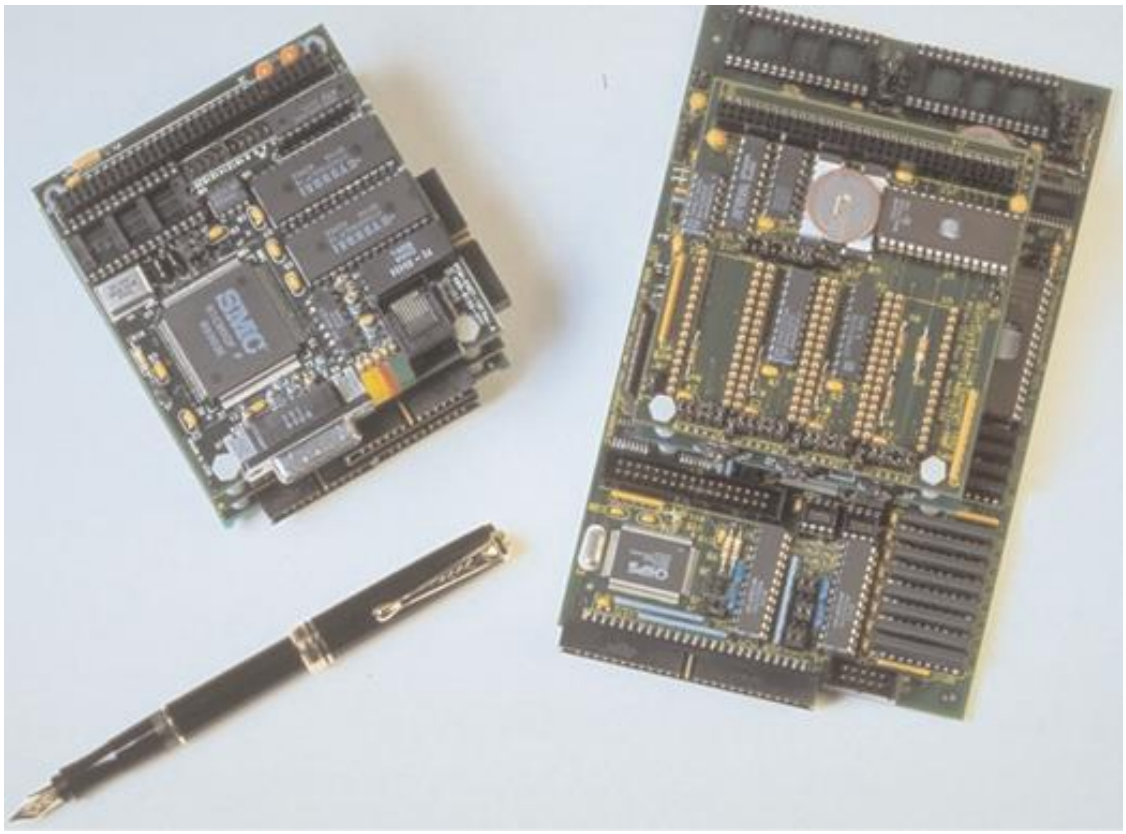


Figura 8: Módulo PC/104 usado como computador (esq.); e Módulo PC/104 usado como expansão para um SBC (dir.).

Um módulo *PC/104* é uma placa baseada em *Industry Standard Architecture (ISA) bus.* As definições de sinais e *Timing* são os mesmos do *PC* família *x86*. O conector P1 do *PC/104* tem 64 pinos assim como o *PC-XT* e é combinado com o conector P2

de 40 pinos para a plena compatibilidade com *PC/AT (IBM Personal Computer Advanced Technology)*.A soma dos pinos (64+40 = 104) é a origem do nome *PC/104*.

Uma das características mais inteligentes e vantajosas do *PC/104* é a seu confiável conector tipo pino-e-soquete. Cada conector é concebidos de forma a permitir que vários módulos sejam empilhados simultaneamente uns sobre os outros e conectados ao mesmo bus. Esta é uma característica única entre as arquiteturas de mezanino. Múltiplos módulos permitem uma maior flexibilidade no design, bem como uma maior capacidade de expansão, o que permite que mais funções sejam retiradas da *CPU (host)*, o que dá ao designer mais opções de seleção. A conclusão lógica deste esquema seria começar com um *PC/104* como o anfitrião da *SBC*. O bus tipo "*Stack*" então, torna-se todo o sistema computacional e de *I/O*, formando um robusto *PC* embarcado. Além da natureza "*Self-Stacking*" do bus, quatro buracos de canto estão incluídos para anexar fixadores rosqueáveis de metal ou plástico. Eles formam uma robusta construção mecânica que acrescenta resistência à choques e vibrações. Segundo a norma *PC/104,* até quatro módulos podem ser empilhados simultaneamente. O número total de módulos é uma função da corrente disponível no bus que é especificada em 4 mA.

Além disso, a força de retenção dos conectores torna difícil a separação dos módulos depois que uma unidade é montada. Para fazer a desmontagem mais fácil e rápida, a *Parvus Corporation* criou uma ferramenta de extração de placas *PC/104* para acelerar o trabalho, mostrada na figura 9.

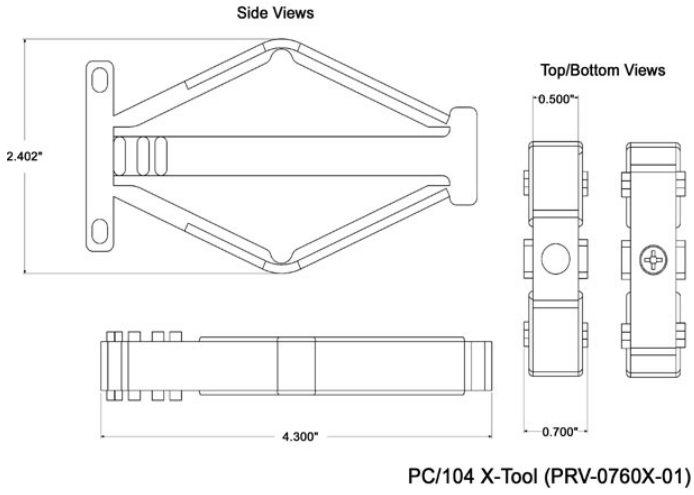


Figura 9: Ferramenta de extração criada pela Parvus.

O consórcio *PC/104* especifica duas versões de bus: 8 e 16 bits, que correspondem ao *PC* e *PC/AT*, respectivamente. As dimensões e o modelo padrão *PC/104* podem ser observados nas figuras 10 e 11.

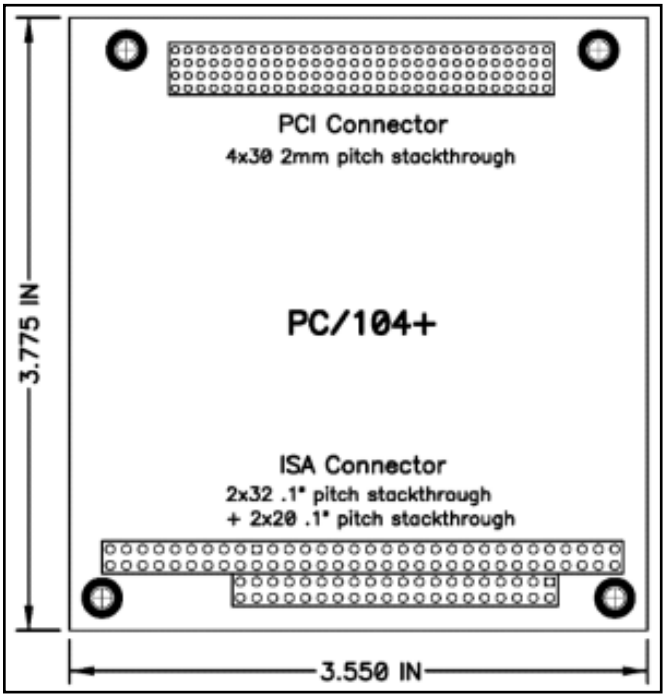


Figura 10: Padrão de placa PC-104 com conector PCI acoplado; Fonte: [5].

Figura 11: Configuração típica de módulo de CPU no formato PC-104; Fonte: [5].

6.2. O SISTEMA ELETRÔNICO

Ambos os módulos MAPD e MPR têm seu projeto centralizado no uso de *FPGAs*. No MPR o *FPGA* é responsável pela comunicação via *PC104 ISA Bus*, pelo armazenamento das formas de onda a serem transmitidas e pelo disparo dos sinais de acionamento dos transdutores. No MAPD o *FPGA* é responsável pela recepção dos sinais dos ecos e detecção dos *TOAs* usando *DSP*, armazenamento dos dados processados em memória *EEPROM* e comunicação via barramento.

O *FPGA* pode ser facilmente reconfigurado, o que torna simples, rápido e barato implementar novos circuitos com o módulo. Para testar um novo algoritmo de processamento digital, necessita-se apenas traduzi-lo em circuitos digitais e transferi-lo para a memória de configuração do *FPGA*. Desta forma, diversas técnicas podem ser testadas sem a necessidade de fazer novas placas e circuitos. Uma das principais dificuldades no desenvolvimento deste módulo eletrônico é a geração da onda de excitação dos transdutores. Para se conseguir sinais de eco com amplitude suficientemente grande para a aquisição, é necessário que o transdutor seja excitado com tensões elevadas, próximas a 200V pico a pico. Como os transdutores usados nesta aplicação requerem uma freqüência de excitação entre 5 e 10MHz, surge o problema de se conseguir chavear altas tensões a altas freqüências, levando-se em consideração as limitações de espaço do projeto inerentes a um sistema embarcado. A solução para este problema é representada na figura 12, que mostra um diagrama esquemático simplificado do circuito de controle de um transdutor, ou seja, o circuito implementado para um canal, possível graças à aplicação de componentes eletrônicos específicos para o acionamento de transdutores de ultra-som ponte de transistores *CMOS* de alta isolação e alta velocidade *TC6320 (SUPERTEX, EUA)* e respectivo *gate driver MD1210K6, (SUPERTEX, EUA).*

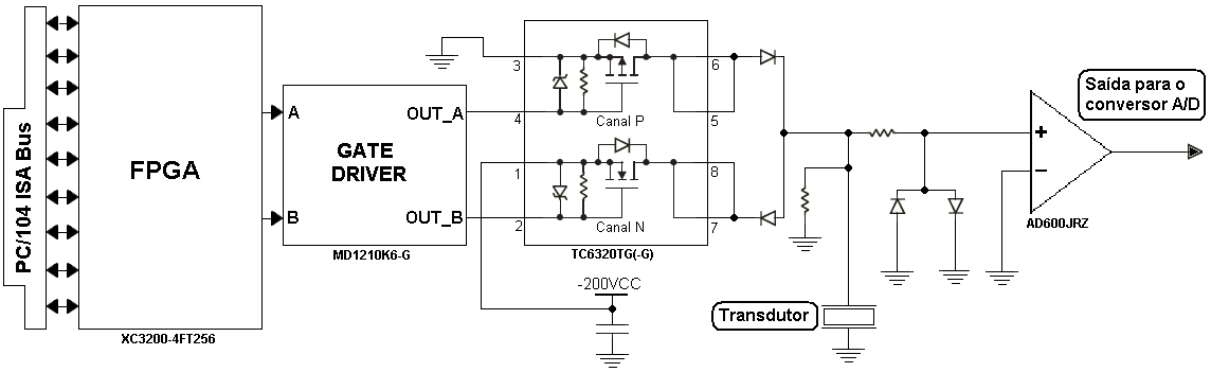


Figura 12: Circuito de controle e recepção de sinais para um transdutor (um canal).

A razão pela qual o processamento, via *DSP*, dos algoritmos descritos é possível decorre de uma simplificação de cálculo gerada pela discretização do sinal do eco. Após a digitalização do sinal recebido *r(t)*, a integral da equação (2) passa a ser um somatório, pois o sinal de entrada é transformado em seu equivalente no tempo discreto, de forma que a função $\Lambda(r(t),\tau)$ passa a ser:

$$\Lambda(r_j, k) = \sum_{j=k}^{k+l} r_j . s_j \quad (4)$$

Onde:

- r_j – *j*-ésima amostra do sinal recebido
- s_j – *j*-ésimo elemento do sinal de referência discretizado
- *l* – número de elementos do sinal de referência discretizado

Neste caso, cada amostra recebida r_j será simplesmente somada ou subtraída ao valor de $\Lambda(k)$, dependendo dos elementos s_j que representam o sinal de referência discretizado que podem assumir os valores (+1) ou (-1). Portanto a expressão (4) pode ser implementada digitalmente utilizando-se apenas somadores, sem a necessidade de multiplicações. Esta característica do sinal discretizado s_j permite que todo o processamento do sinal seja realizado em *Sampling Time*, ou seja, a taxa de processamento do sinal recebido é igual à taxa de aquisição de sinais, sendo que a saída do filtro apresenta um atraso no tempo que é fixo e conhecido. A figura 13 mostra um diagrama de blocos simplificado do MAPD.

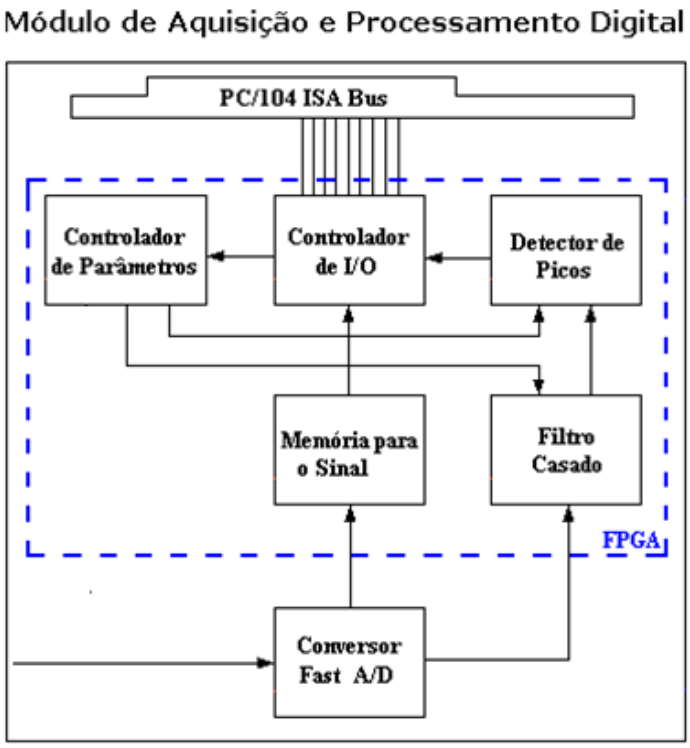


Figura 13: Diagrama de blocos simplificado do MAPD.

O MPR, cuja função é enviar os sinais de excitação aos transdutores e receber e amplificar os ecos de retorno apresenta um esquema básico como o mostrado na figura 14.

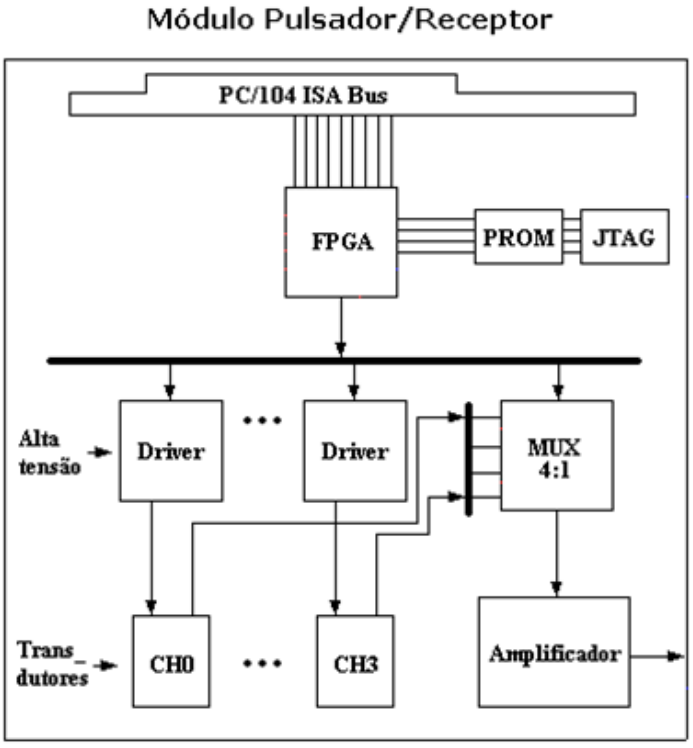


Figura 14: Diagrama simplificado do MPR.

O esquemático do MPR mostra os canais de conexão da placa com os transdutores, os drivers de alta tensão para o acionamento dos transdutores, um multiplexador para chavear os sinais dos quatro canais suportados pela placa, já que o MPR transmite os dados de leitura ao MAPD através de um único canal, com conector *SMA*. Há ainda um amplificador para amplificar o sinal do eco recebido antes de enviá-lo ao MAPD, o barramento para comunicação com o MCPU, um conector *JTAG* usado para carregar o firmware do sistema, uma memória *PROM* que armazena o firmware, já que o *FPGA* perde o circuito nela carregado quando é desenergizada, e, por fim o próprio *FPGA*, que roda o firmware e gera os sinais de excitação.

O MPR apresenta dois modos de configuração, que podem ser escolhidos através dos *jumpers* J2, J3 e J4, correspondentes aos sinais M2, M1 e M0, respectivamente. A tabela 2 mostra estas configurações com mais detalhes.

Modo de Configuração <M0:M1:M2>	Jumpers <J2:J3:J4>	Descrição
Master Serial <0:0:0>		Padrão. O FPGA realiza o boot a partir da memória EEPROM automaticamente.
JTAG <1:0:1>		O FPGA espera pela configuração através da interface JTAG de 4 fios

Tabela 2: Modos de configuração do MPR

Os jumpers J2, J3 e J4 podem ser vistos em detalhe na figura 15.

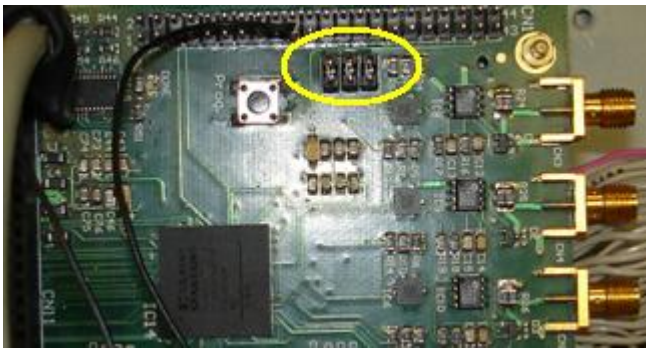


Figura 15: Detalhe mostrando posição dos jumpers na placa.

6.3. PARÂMETROS DO FIRMWARE DO MPR E DO FILTRO CASADO

Tanto o firmware do MPR (Módulo Pulsador Receptor) quanto o filtro casado presente no MAPD (Módulo de Aquisição e Processamento Digital) devem permitir a configuração do sinal a ser enviado e seu filtro respectivo.

Como o sinal enviado usa *BPSK* tendo os códigos de Barker como padrão de modulação, os principais parâmetros de configuração do sinal são:

1. **Número de bits**: Especifica o código de Barker a ser utilizado. Este parâmetro pode conter os valores 2, 3, 4, 5, 7, 11, 13, que correspondem aos códigos apresentados na tabela 1.
2. **Ciclos por bit**: Especifica o número de ciclos da onda portadora que serão agrupados por bit do código de Barker, ou seja, se o número de ciclos por bit é três, então a fase da onda será chaveada a cada três ciclos da onda portadora. A figura 16 mostra um exemplo de um sinal de dois bits com três ciclos por bit.

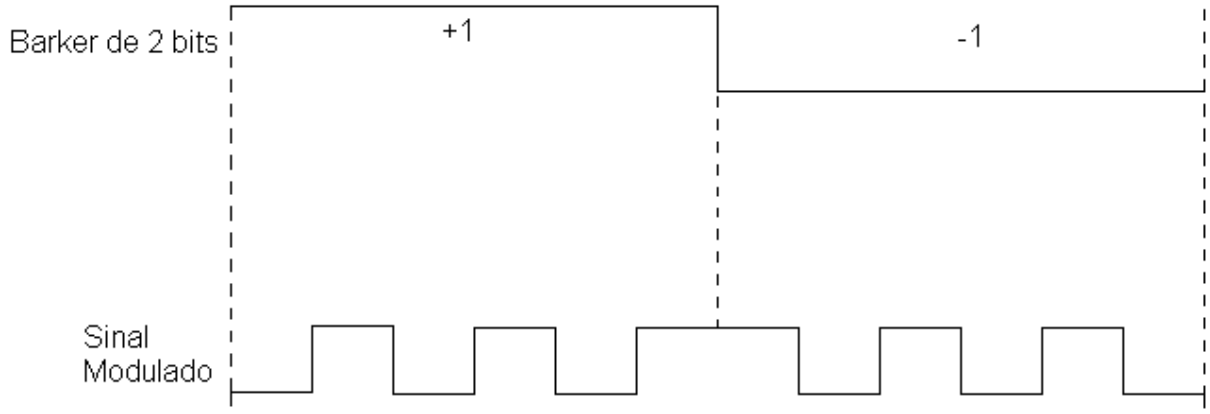


Figura 16: Exemplo de sinal com 2 bits e 3 ciclos por bit.

3. **Amostras por Ciclo**: Especifica a taxa de amostragem em relação a um ciclo da onda portadora. Um valor de “amostras por ciclo” igual a quatro significa que o sistema colhe quatro amostras a cada ciclo da onda portadora. Este parâmetro se aplica apenas ao filtro casado.

7. METODOLOGIA

A abordagem de desenvolvimento para este projeto seguiu uma seqüência estrutural modulada em três etapas principais, como se segue:

7.1. ESTUDO TEÓRICO E TREINAMENTO

Nesta primeira etapa estarão concentrados os processos de estudo, treinamento e capacitação em uso de dispositivos essenciais ao desenvolvimento do projeto

7.1.1. ESTUDO DA LINGUAGEM VHDL E PROGRAMAÇÃO DE FPGAs

O início do processo de elaboração do projeto se dará com o estudo intensivo da linguagem-padrão *VHDL* de descrição de circuitos eletrônicos digitais, e estudo de aplicações em circuitos lógicos programáveis de alta densidade do tipo *FPGA (Field Programmable Gate Array)* para o desenvolvimento de controladores e processadores digitais de sinais. Nesta fase inicial será utilizada a placa de desenvolvimento *SPARTAN-3*, do fabricante *DIGILENT INC*., equipada com o *FPGA SPARTAN*, modelo *XC3S200*, contando com o software de desenvolvimento *ISE 9.2i*, ambos do fabricante *XILINX USA*.

7.1.2. ESTUDO DA ARQUITETURA-PADRÃO PC-104

Estudo da estrutura *PC-104*, pinagem, barramento padrão *ISA*, portas de acesso, consumo de energia, vantagens de uso e aplicações típicas

7.1.3. ESTUDO DO ULTRA-SOM E USO DE TRANSDUTORES

Estudo da teoria física de propagação, reflexão e transmissão de ondas acústicas em faixas de freqüência de ultra-som, considerando o fenômeno de atenuação. Estudo das características mecânicas e elétricas de transdutores de ultra-som e suas aplicações em emissão e recepção de sinais.

7.1.4. ESTUDO DAS TÉCNICAS DE PROCESSAMENTO DIGITAL DE SINAIS

Nesta etapa serão estudadas técnicas de controle e processamento digital de sinais, envolvendo a dinâmica de conversores Analógico/Digital e Digital/Analógico e Amplificação. Estudo de técnicas estatísticas para detecção do sinal e tratamento de ruídos de leitura, envolvendo análise de Máxima Verossimilhança com aplicação de Filtros Casados e controle do sinal com Modulação Binária Por Mudança de Fase, usando Códigos de Barker.

7.2. DESENVOLVIMENTO E PROJETO

7.2.1. DESENVOLVIMENTO DO FIRMWARE DO MPR

Projeto e implementação do firmware do módulo de emissão e recepção de sinais, aplicando técnicas de Processamento Digital de Sinais (*DSP – Digital Signal Processing*) para a obtenção de resultados em tempo real de aquisição, através do desenvolvimento de algoritmos implementados em *FPGA* com linguagem *VHDL*.

7.2.2. MONTAGEM DO MÓDULO PULSADOR/RECEPTOR

Estudo do projeto eletrônico do módulo pulsador/receptor desenvolvido pelo Eng. Douglas Daniel Sampaio Santana, do departamento de eng. Mecatrônica da EPUSP. Compra dos componentes eletrônicos da placa, soldagem dos componentes e testes de verificação do funcionamento.

7.2.3. PROJETO DO FILTRO CASADO VIA DSP

Projeto, implementação e simulação do filtro casado usando *digital signal processing.*

8. RESULTADOS E ANÁLISES

8.1. PROJETO E IMPLEMENTAÇÃO DO FIRMWARE DO MPR

O projeto do *firmware* implementado no módulo pulsador/receptor foi iniciado pelo mapeamento de conexões da *FPGA*. Após o mapeamento, as conexões foram identificadas segundo sua aplicação para que fosse possível dar tratamento coerente para cada sinal com relação aos buffers de entrada e saída utilizados internamente na *FPGA*, em especial no que se refere a sinais que requerem tipos específicos de buffer, como sinais de *clock*, saídas tipo *tri-state* ligadas diretamente ao barramento e portas de comunicação bidirecionais que requerem uma lógica adicional para o controle de seu modo de operação (*Input* ou *Output*). Superada esta etapa, a estrutura de conexões, contendo os *buffers* de todas as entradas e saídas do circuito, foi construída. Neste projeto, apenas o primeiro nível (o de maior prioridade na hierarquia) foi implementado em diagrama esquemático para facilitar a compreensão. Todos os outros níveis foram implementados em *VHDL.* Após a implementação das conexões foram executados testes de verificação do funcionamento do código com o uso do kit de desenvolvimento *SPARTAN-3*. Cada porta foi testada separadamente.

Com a estrutura de conexões prontas seguiu-se o projeto da lógica de controle dos transdutores. O módulo VHDL denominado PULSADOR é o responsável por este controle. Este módulo recebe como entrada os sinais CLK e TRIG, sendo CLK a onda portadora (onda a ser modulada) e TRIG o sinal de controle que dispara o sinal modulado para os transdutores.

O Pulsador é capaz de modular a onda portadora usando todos os sete códigos de Barker apresentados na tabela 1, além de permitir a escolha do número de ciclos por bit do sinal modulado. Estas opções são configuradas em parâmetros presentes no próprio código, os quais são:

- **cpb**: Número de ciclos da onda portadora por bit do código de Barker.
- **nbits**: Número de bits do código de Barker escolhido.

- **bt1 a bt13**: Valores dos bits do código de Barker escolhido, de acordo com a tabela 1. Estes valores podem ser 1 ou 0 (onde os zeros correspondem aos valores -1 presentes na tabela 1).

O código do firmware foi testado no simulador do software *ISE 9.2i* tendo como entrada uma onda quadrada de 50Mhz. Esta frequência foi escolhida apenas para demonstrar a capacidade do circuito de responder em altas frequências, já que para as condições projetadas o circuito deverá trabalhar com ondas de 5MHz, que é a frequência central do transdutor utilizado. As simulações realizadas levam em conta a lógica do circuito e *timing* de forma a avaliar sua resposta no tempo.

Cada simulação foi realizada tendo a onda portadora modulada por cada um dos códigos de Barker presentes na tabela 1. Todas as simulações foram realizadas com três ciclos de onda portadora para cada bit do código de Barker.

O diagrama esquemático do nível principal e os códigos em *VHDL* dos outros níveis do *firmware* do MPR, além das simulações do circuito são apresentados no APÊNDICE A.

8.2. PROJETO E IMPLEMENTAÇÃO DO FILTRO CASADO

Para explicar o desenvolvimento do filtro casado, tomemos como exemplo o filtro para um código de dois bits, com um ciclo por bit e quatro amostras por ciclo, como mostrado na figura 17.

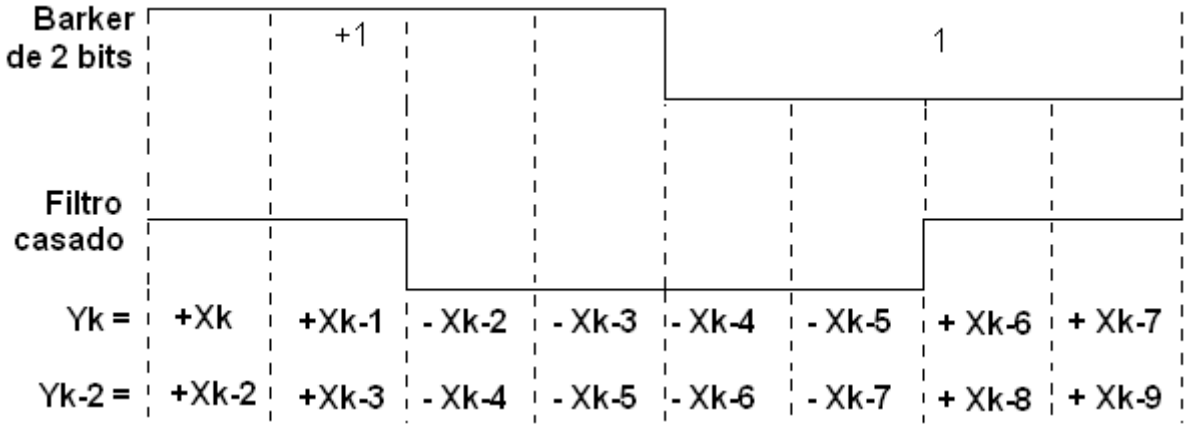


Figura 17: Exemplo de filtro casado.

Para o filtro mostrado na figura 17 x representa as amostras a cada instante e y representa a saída do filtro, também a cada instante. Como explicado na seção 6.2, a convolução do filtro casado com o sinal recebido se resume ao somatório apresentado na equação (4), cuja extensão corresponde ao número de amostras de um sinal completo. Neste exemplo, a extensão do sinal é de oito amostras. Desta forma, como se pode observar da figura 17, o filtro *FIR* (*Finite Impulse Response*) correspondente a este sistema é dado por

$$y_k = x_k + x_{k-1} - x_{k-2} - x_{k-3} - x_{k-4} - x_{k-5} + x_{k-6} + x_{k-7} \qquad (5).$$

Tomando a saída do filtro duas amostras anteriores a este instante, ou seja, meio período da onda portadora antes de y_k, temos

$$y_{k-2} = x_{k-2} + x_{k-3} - x_{k-4} - x_{k-5} - x_{k-6} - x_{k-7} + x_{k-8} + x_{k-9} \qquad (6).$$

Somando as duas expressões, a correspondente a y_k e a correspondente a meio ciclo de onda tomado anteriormente, obtemos

$$y_k = -y_{k-2} + x_k + x_{k-1} - 2(x_{k-4} + x_{k-5}) + x_{k-8} + x_{k-9} \qquad (7),$$

que corresponde ao filtro *IIR* (*Infinite Impulse Response*) do mesmo sistema.

Na equação (7) o primeiro termo do lado direito da equação corresponde à saída do filtro meio ciclo da onda portadora antes do instante k, os dois termos seguintes (x_k e x_{k-1}) correspondem ao primeiro meio ciclo do filtro, o termo seguinte ($-2(x_{k-4}+x_{k-5})$) corresponde ao primeiro meio ciclo após a inversão de fase. Este termo surgirá na equação tantas vezes quantas forem as ocorrências de inversão de fase no código de Barker. A tabela 3 mostra a quantidade de transições e o bit imediatamente anterior à transição para cada código de barker. Por fim, os dois termos finais (x_{k-8} e x_{k-9}) correspondem ao último meio ciclo do filtro.

Cód. De Barker (bits)	Bit anterior à transição i						Total de Transições
	1	2	3	4	5	6	
2	1	-	-	-	-	-	1
3	2	-	-	-	-	-	1
4	2	3	-	-	-	-	2
5	3	4	-	-	-	-	2
7	3	5	6	-	-	-	3
11	3	6	7	9	10	-	5
13	5	7	9	10	11	12	6

Tabela 3: Número de transições e bits de transição de fase dos códigos de Barker.

Para efeito de análise do método vamos considerar a função de transferência do filtro casado utilizado no exemplo comparando, matematicamente, as saídas dos filtros *FIR* e *IIR*, de forma a verificar a validade matemática do sistema.

Aplicando a transformada Z à equação (5), temos:

$$Y(z) = X(z)(1 + z^{-1} - z^{-2} - z^{-3} - z^{-4} - z^{-5} + z^{-6} + z^{-7}) \qquad (8)$$

Ou seja, a função de transferência deste filtro é

$$G_{FIR}(z) = (1 + z^{-1} - z^{-2} - z^{-3} - z^{-4} - z^{-5} + z^{-6} + z^{-7}) \qquad (9)$$

Aplicando a transformada z à equação (7), temos:

$$Y(z)(1 + z^{-2}) = X(z)(1 - z^{-1} - 2z^{-4} - 2z^{-5} + z^{-8} + z^{-9}) \qquad (10)$$

então, a função de transferência deste filtro é

$$G_{IIR}(z) = \frac{(1 - z^{-1} - 2z^{-4} - 2z^{-5} + z^{-8} + z^{-9})}{(1 + z^{-2})} \qquad (11)$$

Aplicando como entrada um *Delta de Kronecker* para ambos os sistemas, observa-se que tanto o filtro *FIR* quanto o filtro *IIR* apresentam a mesma resposta no tempo, dada por

$$y_0 = 1; y_1 = 1; y_2 = -1; y_3 = -1; y_4 = -1; y_5 = -1; y_6 = 1; y_7 = 1; y_n = 0, \forall\, n \in Z | n > 7$$

Isto demonstra que ambas são representações do mesmo sistema (neste caso o filtro representado na figura 17), como era esperado. Este fato também pode ser demonstrado realizando a divisão de polinômios presente na equação (11), o que mostrará que as funções de transferência $G_{FIR}(z)$ e $G_{IIR}(z)$ são iguais.

Seguindo a lógica apresentada neste exemplo, foi projetado um circuito digital, configurável capaz de implementar filtros casados para sinais modulados com qualquer um dos sete códigos de Barker apresentados na tabela 1. Este circuito implementa o filtro *IIR* do sinal desejado, usando linguagem *VHDL*.

A escolha do filtro a ser usado é feita através dos parâmetros de configuração presentes no código *VHDL* do circuito, os quais são:

- **cpb**: Número de ciclos por bit
- **nbits**: Número de bits do código de Barker
- **bt1, bt2, bt3, bt4, bt5, bt6**: Bits imediatamente anteriores a cada uma das transições de fase presentes no código de Barker, que podem ser de 1 a 6.
- **A**: Número de transições de fase presentes no código de Barker.
- **barr**: Número de bits no barramento de dados.

O código completo do circuito que implementa o filtro casado encontra-se transcrito no APÊNDICE B.

A título de verificação, foram realizadas simulações com o circuito projetado, através do simulador do *software ISE 9.2i*

Primeiramente, foram realizadas simulações lógicas do circuito para que fosse possível verificar se o mesmo apresentava o comportamento projetado. Nestas simulações todos os filtros foram excitados com a função *Delta de Kronecker*, de forma que suas respostas no tempo pudessem ser comparadas com as respostas previstas. Diversas iterações de simulação e mudança do código foram realizadas até que o circuito apresentasse o comportamento esperado. É importante observar que as simulações lógicas não levam em conta o tempo de propagação do sinal no circuito, de forma que problemas de *hazard* e outros problemas possíveis com a aplicação de sinais de alta frequência não são detectados, em outras palavras, estas simulações apenas garantem que a lógica do circuito está correta. Os resultados das simulações lógicas dos filtros encontram-se no APÊNDICE C.

Seguindo-se às simulações lógicas, foram realizadas as simulações de *Timing* do circuito, de forma a se detectar possíveis falhas dada a frequência do sinal e o tempo de resposta do circuito. Todos os testes foram realizados usando a função

Delta de Kronecker para a excitação do sistema, e plotados os gráficos da resposta do filtro no tempo.

A primeira simulação foi realizada com o código de 2 bits, 1 ciclo por bit, 10 amostras por ciclo, ciclo de *clock* simétrico de 20 ns e *Input Setup Time* de 1 ns. O resultado é apresentado na figura 18.

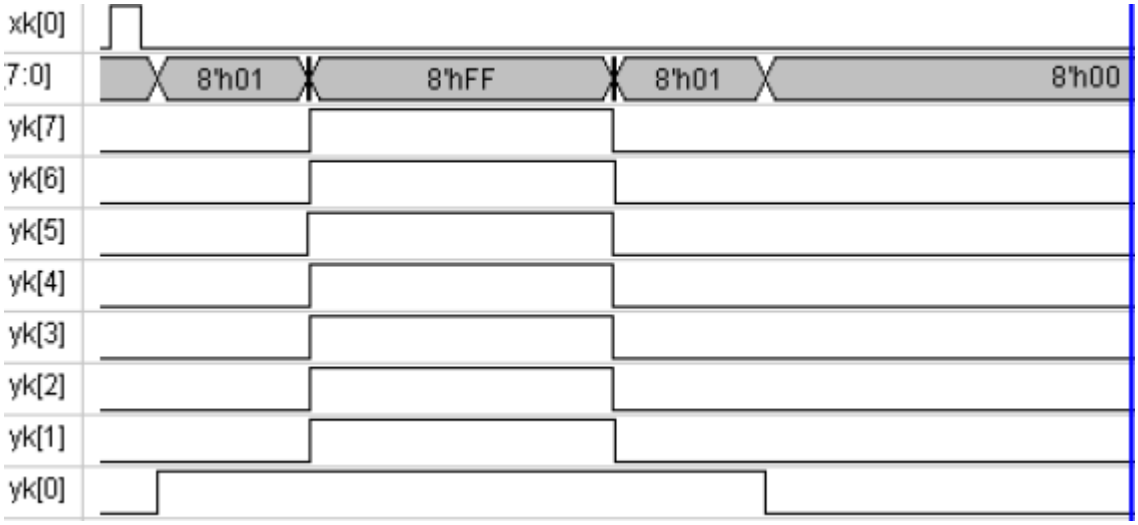


Figura 18: Resposta no tempo do filtro de 2 bits e 1 ciclo por bit.

Como mostrado na figura 18 este filtro atende aos requisitos de tempo especificados na simulação, o que não acontece nos testes subsequentes, como será mostrado a seguir.

A figura 18 mostra o resultado da simulação para o mesmo filtro da figura (código de 2 bits, 1 ciclo por bit, 10 amostras por ciclo, ciclo de *clock* simétrico de 20 ns), porém com um *Input Setup Time* de 2 ns. Observa-se que a resposta do circuito foi totalmente distorcida, indicando que o mesmo não é capaz de funcionar com um *Input Setup Time* de 2 ns.

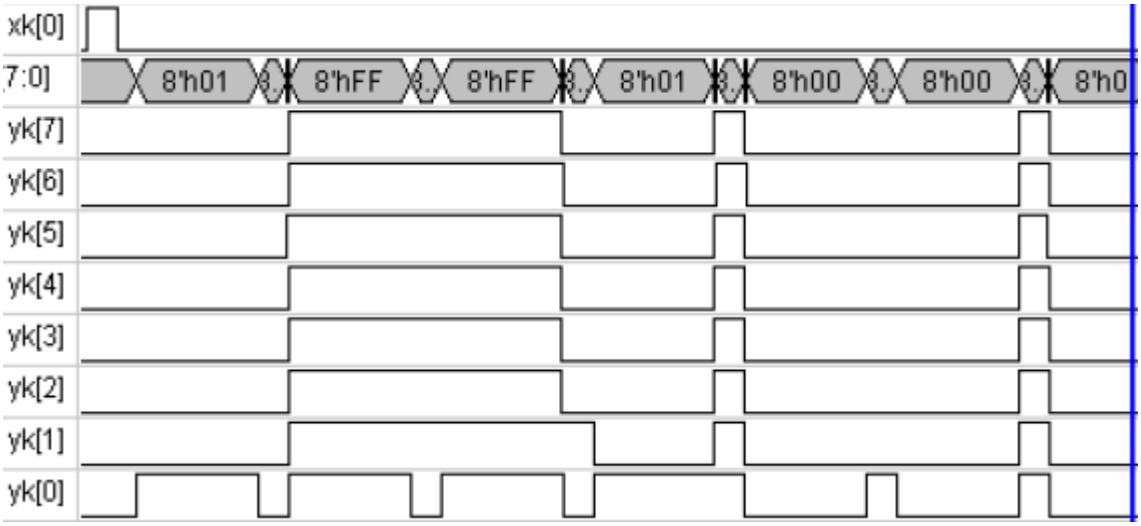


Figura 19: Resposta no tempo do filtro de 2 bits e 1 ciclo por bit com input setup time de 2ns.

Resultados semelhantes ao apresentado na figura 19 foram obtidos em outras simulações, indicando dificuldade do circuito de responder em 50 MHz com *Input Setup Time* diferente de 1ns. Em especial, para os códigos de 4 bits ou mais as simulações apresentaram resultados negativos mesmo com o *Input Setup Time* de 1 ns. De qualquer forma estes resultados mostram uma necessidade de se realizar testes reais com o circuito para que se possa verificar se o mesmo é ou não capaz de funcionar sem falhas a 50 MHz. O APÊNDICE D apresenta algumas das simulações de *timing* realizadas com os filtros.

8.3. MONTAGEM DO SISTEMA

Terminados os testes com os códigos do firmware e do filtro casado, iniciou-se o processo de montagem do protótipo pulsador/receptor, cujos esquemas elétricos são apresentados no ANEXO A. A primeira etapa deste processo consiste na revisão das listas de componentes e verificação de disponibilidade nos fornecedores e posteriormente, a compra destes componentes. Seguindo-se à compra deu-se início ao processo de soldagem dos componentes na placa. A figura 20 mostra a vista superior e inferior do protótipo, em estágio intermediário de montagem com alguns componentes já soldados.

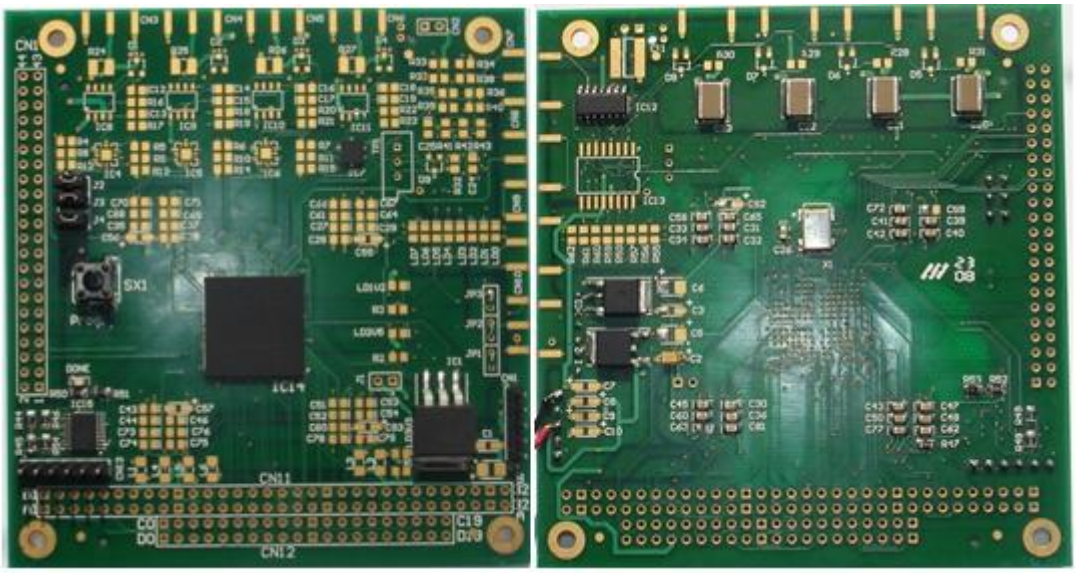


Figura 20: MPR com soldagem parcial.

8.4. TESTES

O MAPD foi testado numa amostra de duto de aço corroído, executando varreduras lineares em diferentes seções representadas pelas linhas brancas de A a H, como pode ser observado na figura 21. O transdutor utilizado é do modelo *Aerotech ALPHA* de 5MHz, com diâmetro de 0.25” (*GE*, EUA). Para os testes, a amostra foi imersa num tanque de água e o transdutor, também imerso em água, foi fixado a um conjunto de movimentação acionado por motores de passo, de forma a manter o transdutor a uma distância constante da amostra ao longo da varredura.

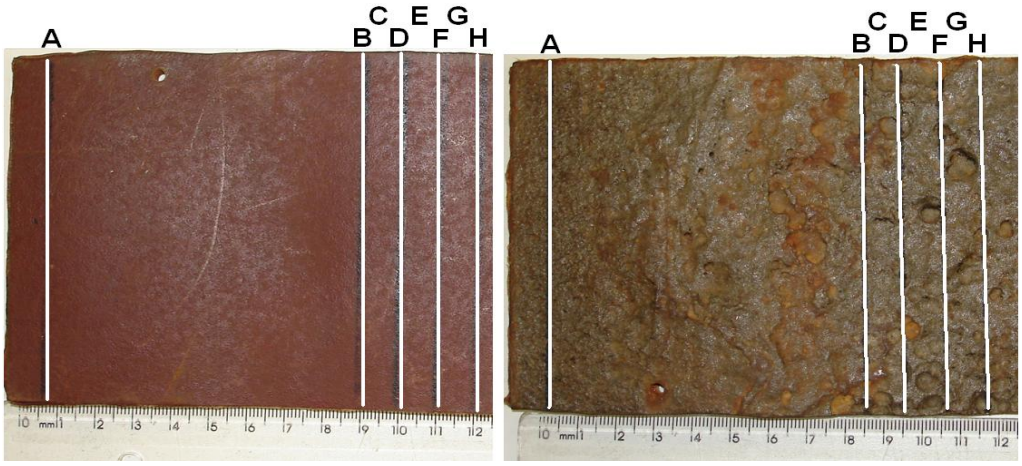


Figura 21: Fotografias das superfícies externa (esq.) e interna (dir) da amostra planificada de duto utilizada nos testes.

Para a exposição e discussão, foram escolhidos dois pontos de inspeção. Um sobre a linha A, a 1mm da linha de referência do início de inspeção (A1), e outro sobre a linha E, a 35mm da linha de referência (E35). Estes pontos foram escolhidos propositalmente por se situarem em relevos bastante diferentes: O ponto A1 encontra-se sobre uma área pouco sujeita à corrosão, enquanto o ponto E35 encontra-se sobre uma área extremamente erodida. Para estes dois pontos são apresentados os testes com dois sinais diferentes: Primeiro um sinal modulado com um código de Barker de 2 bits e 3 ciclos de onda quadrada por bit, e segundo um sinal modulado com um código de Barker de 7 bits e 1 ciclos de onda quadrada por bit. A figura 22 mostra o sinal de 7 bits enviado.

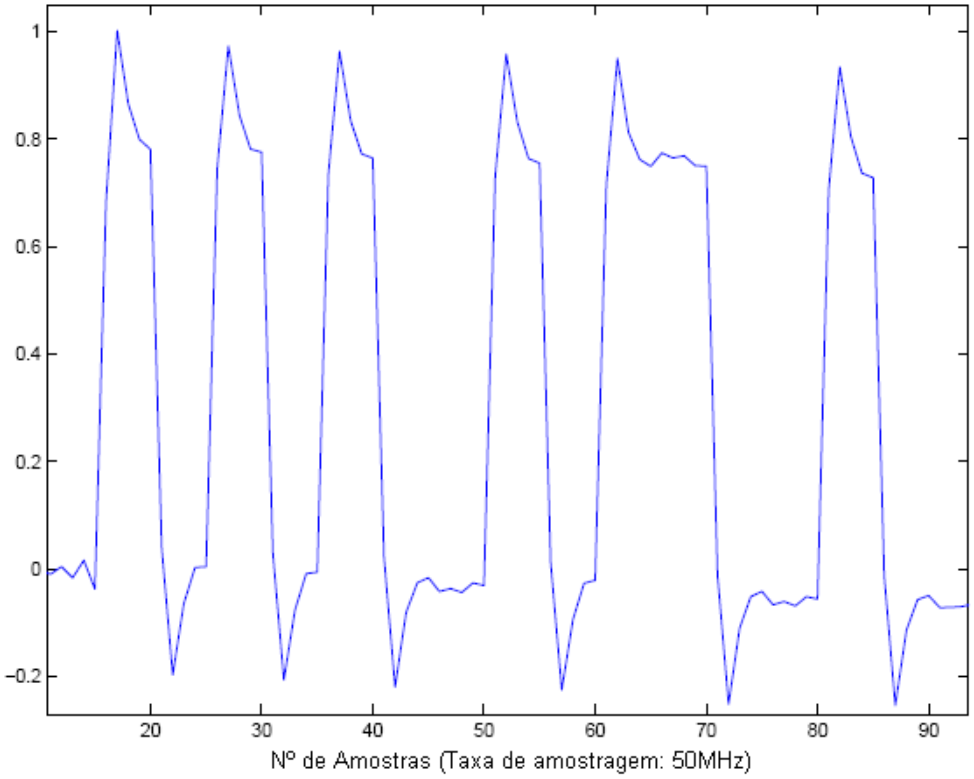


Figura 22: Sinal de input utilizado nos testes de 7 bits/1 ciclo por bit.

As figuras de 23 a 26 mostram os resultados dos testes realizados. Em todas as figuras a) apresenta o eco recebido sem a filtragem e b) apresenta o mesmo sinal de a) após a filtragem com o filtro casado, colocando em destaque os dois primeiros *TOAs* estimados. Sendo a intensidade representada no eixo das ordenadas e o número de amostras no eixo das abscissas.

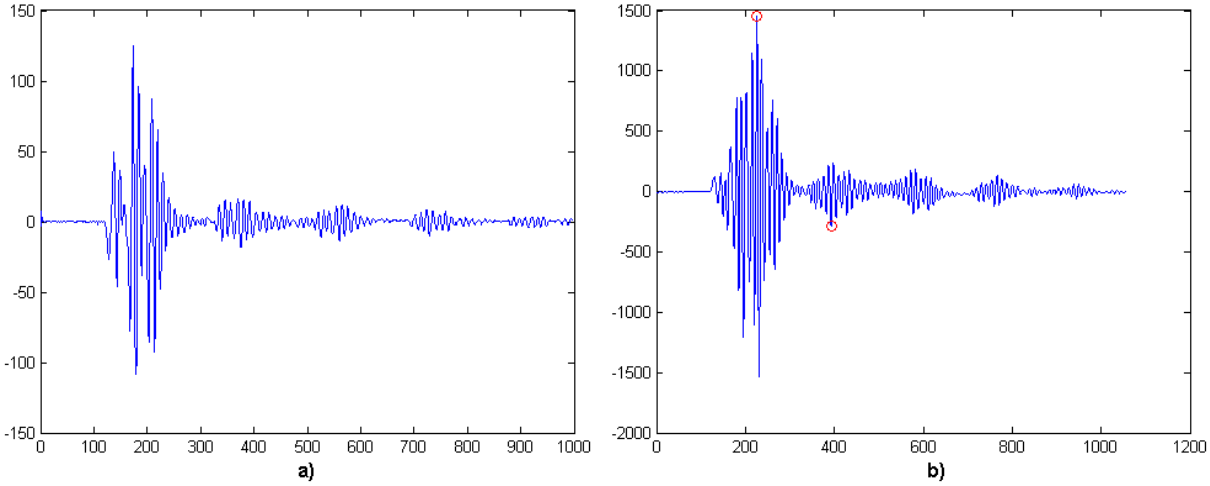


Figura 23: Teste no ponto A1, usando um sinal de 2 bits/3 ciclos por bit; a) sinal recebido e b) sinal após filtragem.

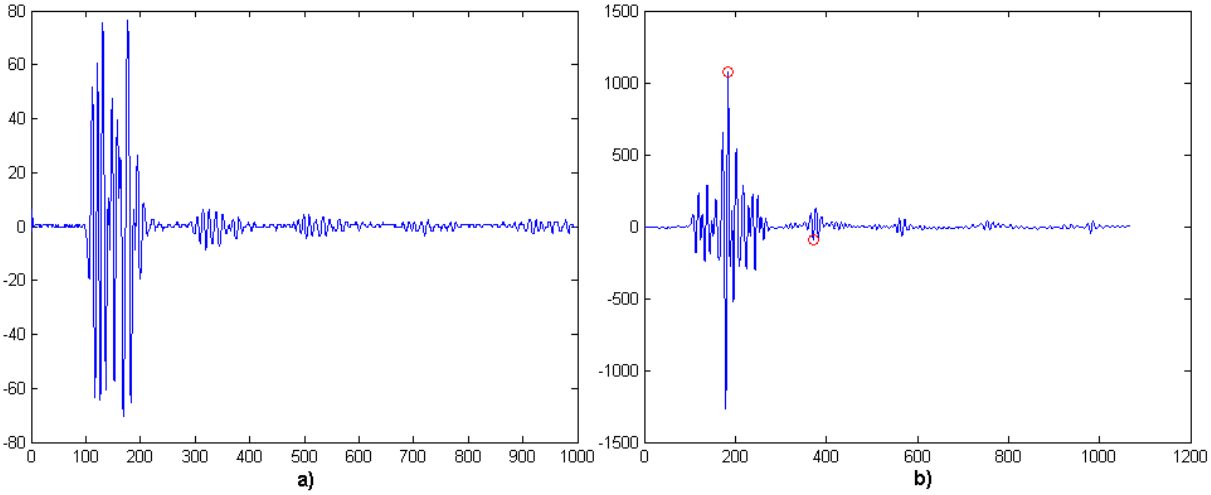


Figura 24: Teste no ponto A1, usando um sinal de 7 bits/1 ciclo por bit; a) sinal recebido e b) sinal após filtragem.

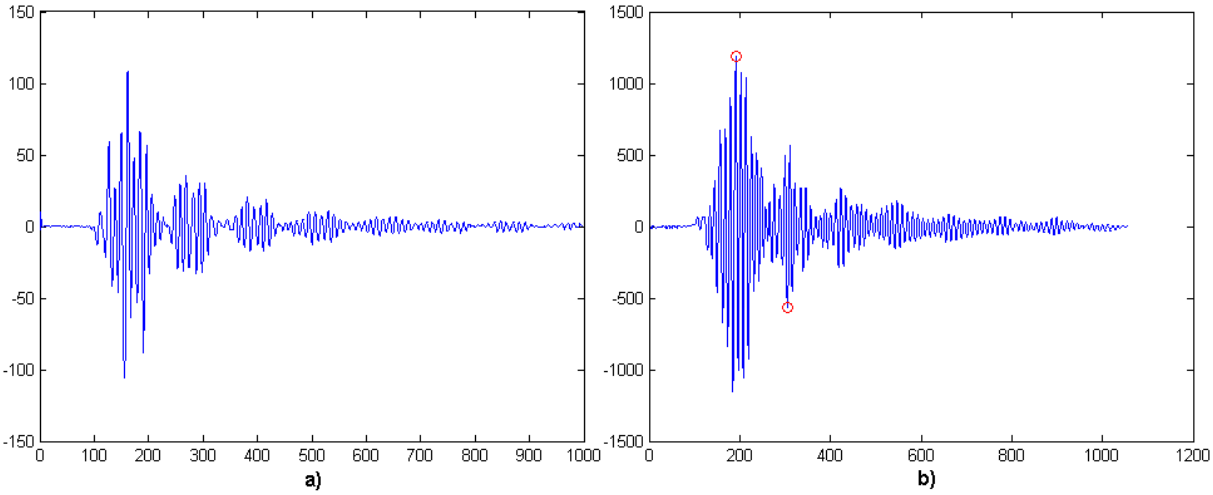


Figura 25: Teste no ponto E35, usando um sinal de 2 bits/3 ciclos por bit; a) sinal recebido e b) sinal após filtragem.

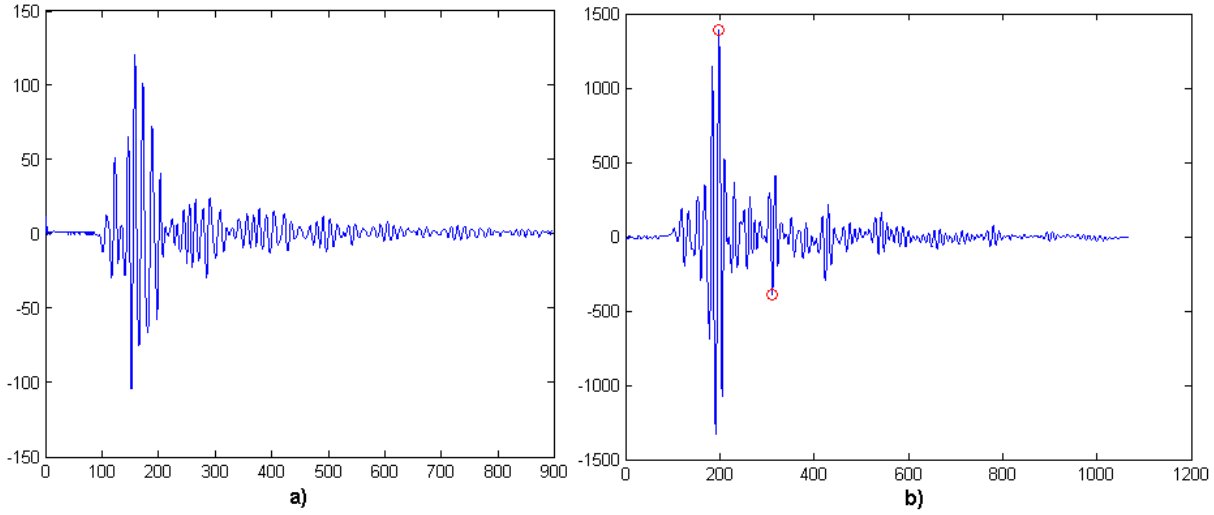


Figura 26: Teste no ponto E35, usando um sinal de 7 bits/1 ciclo por bit; a) sinal recebido e b) sinal após filtragem.

Comparando os resultados gerados pelos sinais de 2 bits e de 7 bits, nota-se uma consistente superioridade do sinal de 7 bits em relação ao de 2 bits, principalmente considerando que o comprimento do sinal completo é quase o mesmo nos dois casos: São 6 ciclos de onda quadrada para o sinal de 2 bits e 7 ciclos para o sinal de 7 bits. A vantagem do sinal de 7 bits fica evidente nos dois pontos mostrados, já que os picos observados nos gráficos dos sinais filtrados de 7 bits são consideravelmente mais pronunciados que os picos observados nos sinais filtrados de 2 bits.

Outro ponto a se destacar é a clara vantagem em se utilizar o filtro casado para o tratamento do eco, já que em todos os gráficos mostrados observa-se um melhor condicionamento no sinal filtrado do que no sinal original. Além disso, nos casos observados, em especial para os sinais de 7 bits o pico do *TOA* é mais pronunciado no sinal filtrado do que no eco de origem. Particularizando para o caso da figura 24, observa-se que o sinal do primeiro eco chega bastante deformado e apresentando 2 picos proeminentes, o que poderia gerar leituras errôneas na estimativa do primeiro *TOA*. Isto, porém, não ocorre com o sinal filtrado, no qual se observa um sinal limpo e com um único pico claro.

Observa-se das figuras 23 a 26 que ambos os sinais testados (2 bits/3 ciclos por bit e 7 bits/1 ciclo por bit) apresentam bons resultados tanto na superfície pouco erodida (A1) quanto na superfície muito erodida (E35), já que não se observam

interferências relevantes de ruído ou de ecos espúrios gerados pelas imperfeições da superfície e reflexões não previstas, o que demonstra que a técnica é eficaz e pode apresentar bons resultados também em superfícies irregulares.

A maior robutez da inspeção com a aplicação do filtro pode ser evidenciada no caso do teste mostrado na figura 24, cujo sinal do eco apresenta dois picos distintos com amplitudes próximas, o que geraria leituras incorretas de *TOAs* em algoritmos tradicionais como os detectores de máximo e detectores de limiar, e que, porém, não geram leituras errôneas se filtradas por um filtro casado. Esta robustez é, em boa parte, fruto de um modelo matemático mais complexo e refinado em que o ponto de máximo que aparece após o filtro é o ponto de máxima verossimilhança procurado, ou seja, o ponto de maior probabilidade de se encontrar o valor real do *TOA.*

9. CONCLUSÕES

Os testes com os circuitos e algoritmos desenvolvidos mostraram-se promissores. O módulo pulsador/receptor projetado foi capaz de excitar o transdutor de forma a produzir sinais claros para a recepção. A considerável diferença nos resultados obtidos com os sinais de 2 e 7 bits para os mesmos pontos confirma a importância de se estudar e aplicar a modulação de sinais para a obtenção de melhores resultados.

A aplicação do filtro casado no processamento do sinal aquisitado mostrou-se bastante eficaz, indicando uma possível elevação no nível de acurácia da estimativa efetuada, quando comparada à estimativa feita sem o uso do filtro.

Com o sistema operando corretamente, o desenvolvimento de novas tecnologias de inspeção por ultra-som será facilitado. Novos algoritmos de processamento digital poderão ser testados, de forma a aprimorar o próprio sistema. O desenvolvimento de sistemas de inspeção com tecnologia nacional poderá permitir a redução de custos para as empresas que dependem deste serviço, além de colaborar na prevenção de danos ambientais causados por vazamentos de óleo.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] CANALES, R. V. et al. Digital Ultrasonic System for Internal Corrosion Assessment on Oil Pipelines. In: COBEM, 19, 2007, Brasília. **Proceedings of COBEM.** Brasília, 2007.

[2] LIMA, Frederico Vimes Faria de. **Desenvolvimento de Um Sistema Acústico de Posicionamento Submarino**. 2003. 139 p. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, São Paulo, 2003.

[3] VAN TREES, H.L. **Detection, Estimation, and Modulation Theory - Part I**. New York: John Wiley, 1968. 712 p.

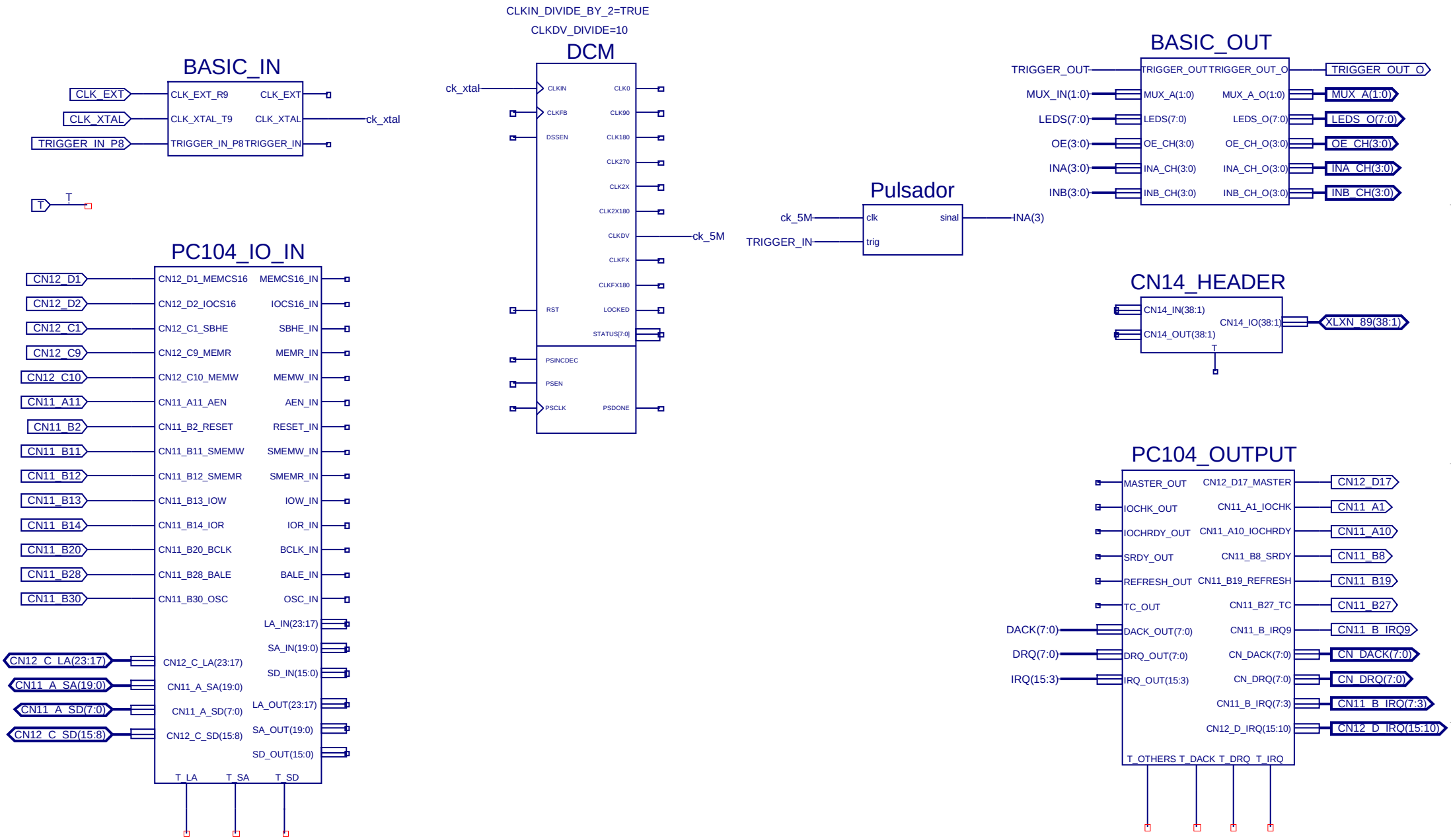
[4] VAN TREES, H.L. **Detection, Estimation, and Modulation Theory - Part III**. New York: John Wiley, 1971. 648 p.

[5].PC/104 EMBEDDED CONSORTIUM. **Specifications – PC/104**. Disponível em: <http://www.pc104.org/ pc104_specs.php >. Acesso em: 15 de jul. 2008.

APÊNDICE A

Código do *Firmware* do MPR

Simulações do Firmware do MPR



TOP MODULE DO FIRMWARE DO MPR

ANDRÉ LUIZ ZIMMERMANN

CÓDIGO VHDL DO BLOCO PULSADOR

Pulsador.vhd Tue Nov 24 19:23:05 2009
Page 1

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.STD_LOGIC_ARITH.ALL;
4 use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6 ---- Uncomment the following library declaration if instantiating
7 ---- any Xilinx primitives in this code.
8 --library UNISIM;
9 --use UNISIM.VComponents.all;
10
11 entity Pulsador is
12 Generic (cpb : Integer := 3; --Número de ciclos por bit
13 nbits : Integer := 13; --Número de bits
14 --Valor de cada bit do código de Barker
15 bt1 : STD_LOGIC := '1';
16 bt2 : STD_LOGIC := '1';
17 bt3 : STD_LOGIC := '1';
18 bt4 : STD_LOGIC := '1';
19 bt5 : STD_LOGIC := '1';
20 bt6 : STD_LOGIC := '0';
21 bt7 : STD_LOGIC := '0';
22 bt8 : STD_LOGIC := '1';
23 bt9 : STD_LOGIC := '1';
24 bt10 : STD_LOGIC := '0';
25 bt11 : STD_LOGIC := '1';
26 bt12 : STD_LOGIC := '0';
27 bt13 : STD_LOGIC := '1');
28 Port ( clk : in STD_LOGIC; --Onda portadora
29 trig : in STD_LOGIC; --Trigger para disparo do sinal
30 sinal : out STD_LOGIC); --Sinal de saída para excitação do transdutor
31 end Pulsador;
32
33 architecture Behavioral of Pulsador is
34 signal ponteiro, cciclo, batual : integer :=0;
35 signal go, pare : std_logic :='0';
36 begin
37
38 process (trig, pare)
39 begin
40 if pare'event and pare='1' then
41 go<='0'; --encerra a emissão do sinal
42 end if;
43 if trig='1' then
44 go<='1'; --inicia a emissão do sinal
45 end if;
46 end process;
47
48 process (clk)
49 variable L : integer; -- Comprimento do sinal
50 begin
51 if go='1' then
52 L:=2*nbits*cpb-1;
53
54 if ponteiro=L then -- Reseta os parâmetros no fim do sinal
55 ponteiro<=0;
56 batual<=0;
57 cciclo<=0;
58 pare<='1';
59 else
60 ponteiro<=ponteiro+1;
61 end if;
```

Pulsador.vhd Tue Nov 24 19:23:05 2009
Page 2

```
62
63 if cciclo=(2*cpb-1) and ponteiro/=L then --Reinicia a contagem de ciclos
a cada bit
64 batual<=batual+1;
65 cciclo<=0;
66 elsif ponteiro/=L then
67 cciclo<=cciclo+1;
68 end if;
69 --Modula a onda portadora de acordo com o código
70 case batual is
71 when 0 => sinal<=clk xnor bt1;
72 when 1 => sinal<=clk xnor bt2;
73 when 2 => sinal<=clk xnor bt3;
74 when 3 => sinal<=clk xnor bt4;
75 when 4 => sinal<=clk xnor bt5;
76 when 5 => sinal<=clk xnor bt6;
77 when 6 => sinal<=clk xnor bt7;
78 when 7 => sinal<=clk xnor bt8;
79 when 8 => sinal<=clk xnor bt9;
80 when 9 => sinal<=clk xnor bt10;
81 when 10 => sinal<=clk xnor bt11;
82 when 11 => sinal<=clk xnor bt12;
83 when 12 => sinal<=clk xnor bt13;
84 when others => null;
85 end case;
86 else
87 sinal<='0';
88 pare<='0';
89 end if;
90 end process;
91
92 end Behavioral;
```

CÓDIGO VHDL DO BLOCO BASIC_IN

BASIC_IN.vhd Tue Apr 28 15:00:16 2009
Page 1

```
1 ----------------------------------------------------------------------------
---------
2 -- Company:
3 -- Engineer:
4 --
5 -- Create Date: 12:31:09 03/20/2009
6 -- Design Name:
7 -- Module Name: BASIC_IN - Behavioral
8 -- Project Name: Firmware Módulo Pulsador
9 -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------
----------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity BASIC_IN is
31 Port ( CLK_EXT_R9 : in STD_LOGIC;
32 CLK_XTAL_T9 : in STD_LOGIC;
33 TRIGGER_IN_P8 : in STD_LOGIC;
34 CLK_EXT : out STD_LOGIC;
35 CLK_XTAL : out STD_LOGIC;
36 TRIGGER_IN : out STD_LOGIC);
37 end BASIC_IN;
38
39 architecture Behavioral of BASIC_IN is
40 ------------------------------------------------------------
41 component IBUF
42 port (I : in STD_ULOGIC;
43 O : out STD_ULOGIC);
44 end component;
45 ------------------------------------------------------------
46 component IBUFG
47 port (I : in STD_ULOGIC;
48 O : out STD_ULOGIC);
49 end component;
50 ------------------------------------------------------------
51 begin
52 CLK_EXT1 : IBUFG port map(I=>CLK_EXT_R9, O=>CLK_EXT);
53 CLK_XTAL1 : IBUFG port map(I=>CLK_XTAL_T9, O=>CLK_XTAL);
54 TRIGGER_IN1 : IBUF port map(I=>TRIGGER_IN_P8, O=>TRIGGER_IN);
55
56 end Behavioral;
57
58
```

CÓDIGO VHDL DO BLOCO BASIC_OUT

BASIC_OUT.vhd Tue Apr 28 15:00:50 2009
Page 1

```
1 ----------------------------------------------------------------------------
2 -- Company:
3 -- Engineer:
4 --
5 -- Create Date: 12:57:58 03/20/2009
6 -- Design Name:
7 -- Module Name: BASIC_OUT - Behavioral
8 -- Project Name: Firmware Módulo Pulsador
9 -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity BASIC_OUT is
31 Port ( TRIGGER_OUT : in STD_LOGIC;
32 MUX_A : in STD_LOGIC_VECTOR (1 downto 0);
33 LEDS : in STD_LOGIC_VECTOR (7 downto 0);
34 OE_CH : in STD_LOGIC_VECTOR (3 downto 0);
35 INA_CH : in STD_LOGIC_VECTOR (3 downto 0);
36 INB_CH : in STD_LOGIC_VECTOR (3 downto 0);
37 TRIGGER_OUT_O : out STD_LOGIC;
38 MUX_A_O : out STD_LOGIC_VECTOR (1 downto 0);
39 LEDS_O : out STD_LOGIC_VECTOR (7 downto 0);
40 OE_CH_O : out STD_LOGIC_VECTOR (3 downto 0);
41 INA_CH_O : out STD_LOGIC_VECTOR (3 downto 0);
42 INB_CH_O : out STD_LOGIC_VECTOR (3 downto 0));
43 end BASIC_OUT;
44
45 architecture Behavioral of BASIC_OUT is
46 ------------------------------------------------------
47 component OBUF
48 port (I : in STD_ULOGIC;
49 O : out STD_ULOGIC);
50 end component;
51 ------------------------------------------------------
52 begin
53 TRIGGER_OUT1 : OBUF port map(I=>TRIGGER_OUT, O=>TRIGGER_OUT_O);
54 --MULTIPLEXADOR
55 MUX0 : OBUF port map(I=>MUX_A(0), O=>MUX_A_O(0));
56 MUX1 : OBUF port map(I=>MUX_A(1), O=>MUX_A_O(1));
57 --LEDS
58 LEDS0 : OBUF port map(I=>LEDS(0), O=>LEDS_O(0));
59 LEDS1 : OBUF port map(I=>LEDS(1), O=>LEDS_O(1));
60 LEDS2 : OBUF port map(I=>LEDS(2), O=>LEDS_O(2));
61 LEDS3 : OBUF port map(I=>LEDS(3), O=>LEDS_O(3));
```

```
62 LEDS4 : OBUF port map(I=>LEDS(4), O=>LEDS_O(4));
63 LEDS5 : OBUF port map(I=>LEDS(5), O=>LEDS_O(5));
64 LEDS6 : OBUF port map(I=>LEDS(6), O=>LEDS_O(6));
65 LEDS7 : OBUF port map(I=>LEDS(7), O=>LEDS_O(7));
66 --PULSADOR 0
67 OE_CH0 : OBUF port map(I=>OE_CH(0), O=>OE_CH_O(0));
68 INA_CH0 : OBUF port map(I=>INA_CH(0), O=>INA_CH_O(0));
69 INB_CH0 : OBUF port map(I=>INB_CH(0), O=>INB_CH_O(0));
70 --PULSADOR 1
71 OE_CH1 : OBUF port map(I=>OE_CH(1), O=>OE_CH_O(1));
72 INA_CH1 : OBUF port map(I=>INA_CH(1), O=>INA_CH_O(1));
73 INB_CH1 : OBUF port map(I=>INB_CH(1), O=>INB_CH_O(1));
74 --PULSADOR 2
75 OE_CH2 : OBUF port map(I=>OE_CH(2), O=>OE_CH_O(2));
76 INA_CH2 : OBUF port map(I=>INA_CH(2), O=>INA_CH_O(2));
77 INB_CH2 : OBUF port map(I=>INB_CH(2), O=>INB_CH_O(2));
78 --PULSADOR 3
79 OE_CH3 : OBUF port map(I=>OE_CH(3), O=>OE_CH_O(3));
80 INA_CH3 : OBUF port map(I=>INA_CH(3), O=>INA_CH_O(3));
81 INB_CH3 : OBUF port map(I=>INB_CH(3), O=>INB_CH_O(3));
82
83 end Behavioral;
84
85
```

CÓDIGO VHDL DO BLOCO CN14_HEADER

CN14_HEADER.vhd Tue Apr 28 15:02:18 2009
Page 1

```
1 ----------------------------------------------------------------------------
---------
2 -- Company:
3 -- Engineer:
4 --
5 -- Create Date: 13:25:27 03/20/2009
6 -- Design Name:
7 -- Module Name: CN14_HEADER - Behavioral
8 -- Project Name: Firmware Módulo Pulsador
9 -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------
----------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity CN14_HEADER is
31 Port ( CN14_IN : out STD_LOGIC_VECTOR (38 downto 1);
32 CN14_OUT : in STD_LOGIC_VECTOR (38 downto 1);
33 CN14_IO : inout STD_LOGIC_VECTOR (38 downto 1);
34 T : in STD_LOGIC);
35 end CN14_HEADER;
36
37 architecture Behavioral of CN14_HEADER is
38 ------------------------------------------------------
39 component IOBUF
40 port (I : in STD_ULOGIC;
41 T : in STD_ULOGIC;
42 O : out STD_ULOGIC;
43 IO: inout STD_ULOGIC);
44 end component;
45 ------------------------------------------------------
46 begin
47 CN1401 : IOBUF port map(I=>CN14_OUT(1) , T=>T, O=>CN14_IN(1) ,
IO=>CN14_IO(1));
48 CN1402 : IOBUF port map(I=>CN14_OUT(2) , T=>T, O=>CN14_IN(2) ,
IO=>CN14_IO(2));
49 CN1403 : IOBUF port map(I=>CN14_OUT(3) , T=>T, O=>CN14_IN(3) ,
IO=>CN14_IO(3));
50 CN1404 : IOBUF port map(I=>CN14_OUT(4) , T=>T, O=>CN14_IN(4) ,
IO=>CN14_IO(4));
51 CN1405 : IOBUF port map(I=>CN14_OUT(5) , T=>T, O=>CN14_IN(5) ,
IO=>CN14_IO(5));
52 CN1406 : IOBUF port map(I=>CN14_OUT(6) , T=>T, O=>CN14_IN(6) ,
IO=>CN14_IO(6));
53 CN1407 : IOBUF port map(I=>CN14_OUT(7) , T=>T, O=>CN14_IN(7) ,
IO=>CN14_IO(7));
54 CN1408 : IOBUF port map(I=>CN14_OUT(8) , T=>T, O=>CN14_IN(8) ,
IO=>CN14_IO(8));
```

```
55 CN1409 : IOBUF port map(I=>CN14_OUT(9) , T=>T, O=>CN14_IN(9) ,
IO=>CN14_IO(9));
56 CN1410 : IOBUF port map(I=>CN14_OUT(10) , T=>T, O=>CN14_IN(10) ,
IO=>CN14_IO(10));
57 CN1411 : IOBUF port map(I=>CN14_OUT(11) , T=>T, O=>CN14_IN(11) ,
IO=>CN14_IO(11));
58 CN1412 : IOBUF port map(I=>CN14_OUT(12) , T=>T, O=>CN14_IN(12) ,
IO=>CN14_IO(12));
59 CN1413 : IOBUF port map(I=>CN14_OUT(13) , T=>T, O=>CN14_IN(13) ,
IO=>CN14_IO(13));
60 CN1414 : IOBUF port map(I=>CN14_OUT(14) , T=>T, O=>CN14_IN(14) ,
IO=>CN14_IO(14));
61 CN1415 : IOBUF port map(I=>CN14_OUT(15) , T=>T, O=>CN14_IN(15) ,
IO=>CN14_IO(15));
62 CN1416 : IOBUF port map(I=>CN14_OUT(16) , T=>T, O=>CN14_IN(16) ,
IO=>CN14_IO(16));
63 CN1417 : IOBUF port map(I=>CN14_OUT(17) , T=>T, O=>CN14_IN(17) ,
IO=>CN14_IO(17));
64 CN1418 : IOBUF port map(I=>CN14_OUT(18) , T=>T, O=>CN14_IN(18) ,
IO=>CN14_IO(18));
65 CN1419 : IOBUF port map(I=>CN14_OUT(19) , T=>T, O=>CN14_IN(19) ,
IO=>CN14_IO(19));
66 CN1420 : IOBUF port map(I=>CN14_OUT(20) , T=>T, O=>CN14_IN(20) ,
IO=>CN14_IO(20));
67 CN1421 : IOBUF port map(I=>CN14_OUT(21) , T=>T, O=>CN14_IN(21) ,
IO=>CN14_IO(21));
68 CN1422 : IOBUF port map(I=>CN14_OUT(22) , T=>T, O=>CN14_IN(22) ,
IO=>CN14_IO(22));
69 CN1423 : IOBUF port map(I=>CN14_OUT(23) , T=>T, O=>CN14_IN(23) ,
IO=>CN14_IO(23));
70 CN1424 : IOBUF port map(I=>CN14_OUT(24) , T=>T, O=>CN14_IN(24) ,
IO=>CN14_IO(24));
71 CN1425 : IOBUF port map(I=>CN14_OUT(25) , T=>T, O=>CN14_IN(25) ,
IO=>CN14_IO(25));
72 CN1426 : IOBUF port map(I=>CN14_OUT(26) , T=>T, O=>CN14_IN(26) ,
IO=>CN14_IO(26));
73 CN1427 : IOBUF port map(I=>CN14_OUT(27) , T=>T, O=>CN14_IN(27) ,
IO=>CN14_IO(27));
74 CN1428 : IOBUF port map(I=>CN14_OUT(28) , T=>T, O=>CN14_IN(28) ,
IO=>CN14_IO(28));
75 CN1429 : IOBUF port map(I=>CN14_OUT(29) , T=>T, O=>CN14_IN(29) ,
IO=>CN14_IO(29));
76 CN1430 : IOBUF port map(I=>CN14_OUT(30) , T=>T, O=>CN14_IN(30) ,
IO=>CN14_IO(30));
77 CN1431 : IOBUF port map(I=>CN14_OUT(31) , T=>T, O=>CN14_IN(31) ,
IO=>CN14_IO(31));
78 CN1432 : IOBUF port map(I=>CN14_OUT(32) , T=>T, O=>CN14_IN(32) ,
IO=>CN14_IO(32));
79 CN1433 : IOBUF port map(I=>CN14_OUT(33) , T=>T, O=>CN14_IN(33) ,
IO=>CN14_IO(33));
80 CN1434 : IOBUF port map(I=>CN14_OUT(34) , T=>T, O=>CN14_IN(34) ,
IO=>CN14_IO(34));
81 CN1435 : IOBUF port map(I=>CN14_OUT(35) , T=>T, O=>CN14_IN(35) ,
IO=>CN14_IO(35));
82 CN1436 : IOBUF port map(I=>CN14_OUT(36) , T=>T, O=>CN14_IN(36) ,
IO=>CN14_IO(36));
83 CN1437 : IOBUF port map(I=>CN14_OUT(37) , T=>T, O=>CN14_IN(37) ,
IO=>CN14_IO(37));
84 CN1438 : IOBUF port map(I=>CN14_OUT(38) , T=>T, O=>CN14_IN(38) ,
IO=>CN14_IO(38));
85
86 end Behavioral;
87
88
```

CÓDIGO VHDL DO BLOCO PC104_IO_IN

PC104_IO_IN.vhd Tue Apr 28 14:55:43 2009
Page 1

```
1 ----------------------------------------------------------------------------
---------
2 -- Company:
3 -- Engineer:
4 --
5 -- Create Date: 15:40:44 03/19/2009
6 -- Design Name:
7 -- Module Name: PC104_IO_IN - Behavioral
8 -- Project Name: Firmware Módulo Pulsador
9 -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------
----------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity PC104_IO_IN is
31 Port (--OTHERS_IN-----------------------------------------
32 CN12_D1_MEMCS16 : in STD_LOGIC;
33 MEMCS16_IN : out STD_LOGIC;
34 CN12_D2_IOCS16 : in STD_LOGIC;
35 IOCS16_IN : out STD_LOGIC;
36 CN12_C1_SBHE : in STD_LOGIC;
37 SBHE_IN : out STD_LOGIC;
38 CN12_C9_MEMR : in STD_LOGIC;
39 MEMR_IN : out STD_LOGIC;
40 CN12_C10_MEMW : in STD_LOGIC;
41 MEMW_IN : out STD_LOGIC;
42 CN11_A11_AEN : in STD_LOGIC;
43 AEN_IN : out STD_LOGIC;
44 CN11_B2_RESET : in STD_LOGIC;
45 RESET_IN : out STD_LOGIC;
46 CN11_B11_SMEMW : in STD_LOGIC;
47 SMEMW_IN : out STD_LOGIC;
48 CN11_B12_SMEMR : in STD_LOGIC;
49 SMEMR_IN : out STD_LOGIC;
50 CN11_B13_IOW : in STD_LOGIC;
51 IOW_IN : out STD_LOGIC;
52 CN11_B14_IOR : in STD_LOGIC;
53 IOR_IN : out STD_LOGIC;
54 CN11_B20_BCLK : in STD_LOGIC;
55 BCLK_IN : out STD_LOGIC;
56 CN11_B28_BALE : in STD_LOGIC;
57 BALE_IN : out STD_LOGIC;
58 CN11_B30_OSC : in STD_LOGIC;
59 OSC_IN : out STD_LOGIC;
60 --LA_MODULE-------------------------------------------
61 LA_OUT : in STD_LOGIC_VECTOR (23 downto 17);
```

```
62 T_LA : in STD_LOGIC;
63 LA_IN : out STD_LOGIC_VECTOR (23 downto 17);
64 CN12_C_LA : inout STD_LOGIC_VECTOR (23 downto 17);
65 --SA_MODULE-----------------------------------------
66 SA_OUT : in STD_LOGIC_VECTOR (19 downto 0);
67 T_SA : in STD_LOGIC;
68 SA_IN : out STD_LOGIC_VECTOR (19 downto 0);
69 CN11_A_SA : inout STD_LOGIC_VECTOR (19 downto 0);
70 --SD_MODULE-----------------------------------------
71 SD_OUT : in STD_LOGIC_VECTOR (15 downto 0);
72 T_SD : in STD_LOGIC;
73 SD_IN : out STD_LOGIC_VECTOR (15 downto 0);
74 CN11_A_SD : inout STD_LOGIC_VECTOR (7 downto 0);
75 CN12_C_SD : inout STD_LOGIC_VECTOR (15 downto 8));
76 ----------------------------------------------------
77 end PC104_IO_IN;
79 architecture Behavioral of PC104_IO_IN is
80 ----------------------------------------------------
81 component IBUF
82 port (I : in STD_ULOGIC;
83 O : out STD_ULOGIC);
84 end component;
85 ----------------------------------------------------
86 component IBUFG
87 port (I : in STD_ULOGIC;
88 O : out STD_ULOGIC);
89 end component;
90 ----------------------------------------------------
91 component IOBUF
92 port (I : in STD_ULOGIC;
93 T : in STD_ULOGIC;
94 O : out STD_ULOGIC;
95 IO: inout STD_ULOGIC);
96 end component;
97 ----------------------------------------------------
98 begin
99 --OTHERS_IN
100 BCLK : IBUFG port map(I=>CN11_B20_BCLK, O=>BCLK_IN);
101 OSC : IBUFG port map(I=>CN11_B30_OSC, O=>OSC_IN);
102
103 MEMCS16 : IBUF port map(I=>CN12_D1_MEMCS16, O=>MEMCS16_IN);
104 IOCS16 : IBUF port map(I=>CN12_D2_IOCS16, O=>IOCS16_IN);
105 SBHE : IBUF port map(I=>CN12_C1_SBHE, O=>SBHE_IN);
106 MEMR : IBUF port map(I=>CN12_C9_MEMR, O=>MEMR_IN);
107 MEMW : IBUF port map(I=>CN12_C10_MEMW, O=>MEMW_IN);
108 AEN : IBUF port map(I=>CN11_A11_AEN, O=>AEN_IN);
109 RESET : IBUF port map(I=>CN11_B2_RESET, O=>RESET_IN);
110 SMEMW : IBUF port map(I=>CN11_B11_SMEMW, O=>SMEMW_IN);
111 SMEMR : IBUF port map(I=>CN11_B12_SMEMR, O=>SMEMR_IN);
112 IOW : IBUF port map(I=>CN11_B13_IOW, O=>IOW_IN);
113 IOR : IBUF port map(I=>CN11_B14_IOR, O=>IOR_IN);
114 BALE : IBUF port map(I=>CN11_B28_BALE, O=>BALE_IN);
115 ----------------------------------------------------------
116 --LA_MODULE
117 LA17 : IOBUF port map(I=>LA_OUT(17) , T=>T_LA, O=>LA_IN(17) ,
IO=>CN12_C_LA(17)) ;
118 LA18 : IOBUF port map(I=>LA_OUT(18) , T=>T_LA, O=>LA_IN(18) ,
IO=>CN12_C_LA(18)) ;
119 LA19 : IOBUF port map(I=>LA_OUT(19) , T=>T_LA, O=>LA_IN(19) ,
IO=>CN12_C_LA(19)) ;
120 LA20 : IOBUF port map(I=>LA_OUT(20) , T=>T_LA, O=>LA_IN(20) ,
IO=>CN12_C_LA(20)) ;
121 LA21 : IOBUF port map(I=>LA_OUT(21) , T=>T_LA, O=>LA_IN(21) ,
IO=>CN12_C_LA(21)) ;
122 LA22 : IOBUF port map(I=>LA_OUT(22) , T=>T_LA, O=>LA_IN(22) ,
IO=>CN12_C_LA(22)) ;
```

```
123 LA23 : IOBUF port map(I=>LA_OUT(23) , T=>T_LA, O=>LA_IN(23) ,
IO=>CN12_C_LA(23)) ;
124 ------------------------------------------------------------------
125 --SA_MODULE
126 SA00 : IOBUF port map(I=>SA_OUT(0) , T=>T_SA, O=>SA_IN(0) ,
IO=>CN11_A_SA(0)) ;
127 SA01 : IOBUF port map(I=>SA_OUT(1) , T=>T_SA, O=>SA_IN(1) ,
IO=>CN11_A_SA(1)) ;
128 SA02 : IOBUF port map(I=>SA_OUT(2) , T=>T_SA, O=>SA_IN(2) ,
IO=>CN11_A_SA(2)) ;
129 SA03 : IOBUF port map(I=>SA_OUT(3) , T=>T_SA, O=>SA_IN(3) ,
IO=>CN11_A_SA(3)) ;
130 SA04 : IOBUF port map(I=>SA_OUT(4) , T=>T_SA, O=>SA_IN(4) ,
IO=>CN11_A_SA(4)) ;
131 SA05 : IOBUF port map(I=>SA_OUT(5) , T=>T_SA, O=>SA_IN(5) ,
IO=>CN11_A_SA(5)) ;
132 SA06 : IOBUF port map(I=>SA_OUT(6) , T=>T_SA, O=>SA_IN(6) ,
IO=>CN11_A_SA(6)) ;
133 SA07 : IOBUF port map(I=>SA_OUT(7) , T=>T_SA, O=>SA_IN(7) ,
IO=>CN11_A_SA(7)) ;
134 SA08 : IOBUF port map(I=>SA_OUT(8) , T=>T_SA, O=>SA_IN(8) ,
IO=>CN11_A_SA(8)) ;
135 SA09 : IOBUF port map(I=>SA_OUT(9) , T=>T_SA, O=>SA_IN(9) ,
IO=>CN11_A_SA(9)) ;
136 SA10 : IOBUF port map(I=>SA_OUT(10), T=>T_SA, O=>SA_IN(10),
IO=>CN11_A_SA(10));
137 SA11 : IOBUF port map(I=>SA_OUT(11), T=>T_SA, O=>SA_IN(11),
IO=>CN11_A_SA(11));
138 SA12 : IOBUF port map(I=>SA_OUT(12), T=>T_SA, O=>SA_IN(12),
IO=>CN11_A_SA(12));
139 SA13 : IOBUF port map(I=>SA_OUT(13), T=>T_SA, O=>SA_IN(13),
IO=>CN11_A_SA(13));
140 SA14 : IOBUF port map(I=>SA_OUT(14), T=>T_SA, O=>SA_IN(14),
IO=>CN11_A_SA(14));
141 SA15 : IOBUF port map(I=>SA_OUT(15), T=>T_SA, O=>SA_IN(15),
IO=>CN11_A_SA(15));
142 SA16 : IOBUF port map(I=>SA_OUT(16), T=>T_SA, O=>SA_IN(16),
IO=>CN11_A_SA(16));
143 SA17 : IOBUF port map(I=>SA_OUT(17), T=>T_SA, O=>SA_IN(17),
IO=>CN11_A_SA(17));
144 SA18 : IOBUF port map(I=>SA_OUT(18), T=>T_SA, O=>SA_IN(18),
IO=>CN11_A_SA(18));
145 SA19 : IOBUF port map(I=>SA_OUT(19), T=>T_SA, O=>SA_IN(19),
IO=>CN11_A_SA(19));
146 ------------------------------------------------------------------
147 --SD_MODULE
148 SD00 : IOBUF port map(I=>SD_OUT(0) , T=>T_SD, O=>SD_IN(0) ,
IO=>CN11_A_SD(0)) ;
149 SD01 : IOBUF port map(I=>SD_OUT(1) , T=>T_SD, O=>SD_IN(1) ,
IO=>CN11_A_SD(1)) ;
150 SD02 : IOBUF port map(I=>SD_OUT(2) , T=>T_SD, O=>SD_IN(2) ,
IO=>CN11_A_SD(2)) ;
151 SD03 : IOBUF port map(I=>SD_OUT(3) , T=>T_SD, O=>SD_IN(3) ,
IO=>CN11_A_SD(3)) ;
152 SD04 : IOBUF port map(I=>SD_OUT(4) , T=>T_SD, O=>SD_IN(4) ,
IO=>CN11_A_SD(4)) ;
153 SD05 : IOBUF port map(I=>SD_OUT(5) , T=>T_SD, O=>SD_IN(5) ,
IO=>CN11_A_SD(5)) ;
154 SD06 : IOBUF port map(I=>SD_OUT(6) , T=>T_SD, O=>SD_IN(6) ,
IO=>CN11_A_SD(6)) ;
155 SD07 : IOBUF port map(I=>SD_OUT(7) , T=>T_SD, O=>SD_IN(7) ,
IO=>CN11_A_SD(7)) ;
156
157 SD08 : IOBUF port map(I=>SD_OUT(8) , T=>T_SD, O=>SD_IN(8) ,
IO=>CN12_C_SD(8)) ;
```

```
158 SD09 : IOBUF port map(I=>SD_OUT(9) , T=>T_SD, O=>SD_IN(9) ,
IO=>CN12_C_SD(9)) ;
159 SD10 : IOBUF port map(I=>SD_OUT(10) , T=>T_SD, O=>SD_IN(10) ,
IO=>CN12_C_SD(10)) ;
160 SD11 : IOBUF port map(I=>SD_OUT(11) , T=>T_SD, O=>SD_IN(11) ,
IO=>CN12_C_SD(11)) ;
161 SD12 : IOBUF port map(I=>SD_OUT(12) , T=>T_SD, O=>SD_IN(12) ,
IO=>CN12_C_SD(12)) ;
162 SD13 : IOBUF port map(I=>SD_OUT(13) , T=>T_SD, O=>SD_IN(13) ,
IO=>CN12_C_SD(13)) ;
163 SD14 : IOBUF port map(I=>SD_OUT(14) , T=>T_SD, O=>SD_IN(14) ,
IO=>CN12_C_SD(14)) ;
164 SD15 : IOBUF port map(I=>SD_OUT(15) , T=>T_SD, O=>SD_IN(15) ,
IO=>CN12_C_SD(15)) ;
165 ------------------------------------------------------------------
166 end Behavioral;
167
168
```

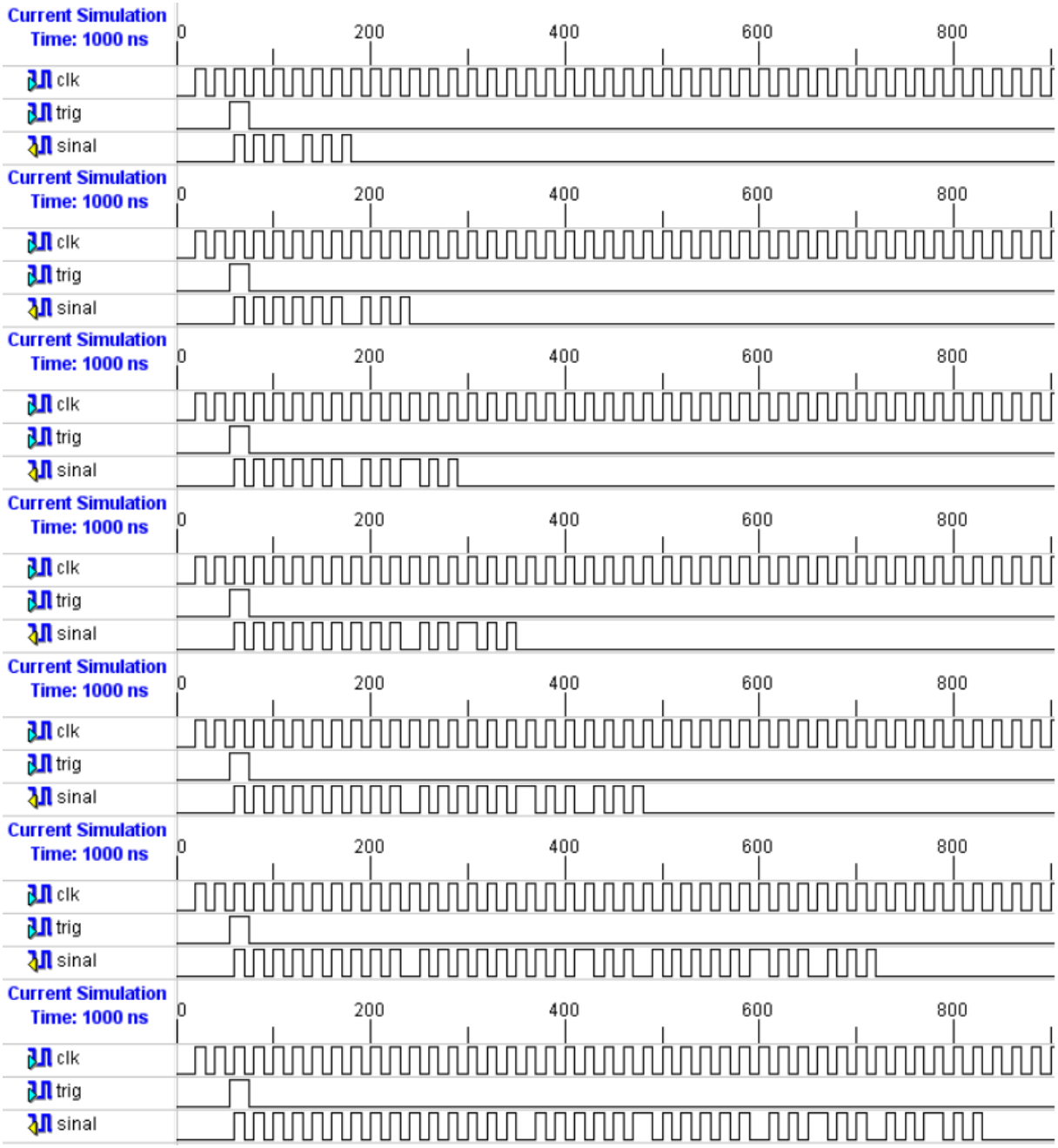
CÓDIGO VHDL DO BLOCO PC104_OUTPUT


```
1 ----------------------------------------------------------------------------
2 -- Company:
3 -- Engineer:
4 --
5 -- Create Date: 16:01:48 03/19/2009
6 -- Design Name:
7 -- Module Name: PC104_OUTPUT - Behavioral
8 -- Project Name: Firmware Módulo Pulsador
9 -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity PC104_OUTPUT is
31 Port ( --OTHERS_OUT-----------------------------------------
32 MASTER_OUT : in STD_LOGIC;
33 CN12_D17_MASTER : out STD_LOGIC;
34 IOCHK_OUT : in STD_LOGIC;
35 CN11_A1_IOCHK : out STD_LOGIC;
36 IOCHRDY_OUT : in STD_LOGIC;
37 CN11_A10_IOCHRDY : out STD_LOGIC;
38 SRDY_OUT : in STD_LOGIC;
39 CN11_B8_SRDY : out STD_LOGIC;
40 REFRESH_OUT : in STD_LOGIC;
41 CN11_B19_REFRESH : out STD_LOGIC;
42 TC_OUT : in STD_LOGIC;
43 CN11_B27_TC : out STD_LOGIC;
44 T_OTHERS : in STD_LOGIC;
45 --DACK_MODULE-----------------------------------------
46 DACK_OUT : in STD_LOGIC_VECTOR (7 downto 0);
47 T_DACK : in STD_LOGIC;
48 CN_DACK : out STD_LOGIC_VECTOR (7 downto 0);
49 --DRQ_MODULE-----------------------------------------
50 DRQ_OUT : in STD_LOGIC_VECTOR (7 downto 0);
51 T_DRQ : in STD_LOGIC;
52 CN_DRQ : out STD_LOGIC_VECTOR (7 downto 0);
53 --IRQ_MODULE-----------------------------------------
54 IRQ_OUT : in STD_LOGIC_VECTOR (15 downto 3);
55 T_IRQ : in STD_LOGIC;
56 CN11_B_IRQ : out STD_LOGIC_VECTOR (7 downto 3);
57 CN11_B_IRQ9 : out STD_LOGIC;
58 CN12_D_IRQ : out STD_LOGIC_VECTOR (15 downto 10));
59 ----------------------------------------------------
60 end PC104_OUTPUT;
61
```

```
62 architecture Behavioral of PC104_OUTPUT is
63 ----------------------------------------------------------
64 component OBUFT
65 port (I : in STD_ULOGIC;
66 T : in STD_ULOGIC;
67 O : out STD_ULOGIC);
68 end component;
69 ----------------------------------------------------------
70 begin
71 --OTHERS_OUT
72 REFRESH : OBUFT port map(I=>REFRESH_OUT, T=>T_OTHERS,
O=>CN11_B19_REFRESH);
73
74 MASTER : OBUFT port map(I=>MASTER_OUT, T=>T_OTHERS, O=>CN12_D17_MASTER);
75 IOCHK : OBUFT port map(I=>IOCHK_OUT, T=>T_OTHERS, O=>CN11_A1_IOCHK);
76 IOCHRDY : OBUFT port map(I=>IOCHRDY_OUT, T=>T_OTHERS,
O=>CN11_A10_IOCHRDY);
77 SRDY : OBUFT port map(I=>SRDY_OUT, T=>T_OTHERS, O=>CN11_B8_SRDY);
78 TC : OBUFT port map(I=>TC_OUT, T=>T_OTHERS, O=>CN11_B27_TC);
79 ----------------------------------------------------------------------
80 --DACK_MODULE
81 DACK0 : OBUFT port map(I=>DACK_OUT(0) , T=>T_DACK, O=>CN_DACK(0));
82 DACK1 : OBUFT port map(I=>DACK_OUT(1) , T=>T_DACK, O=>CN_DACK(1));
83 DACK2 : OBUFT port map(I=>DACK_OUT(2) , T=>T_DACK, O=>CN_DACK(2));
84 DACK3 : OBUFT port map(I=>DACK_OUT(3) , T=>T_DACK, O=>CN_DACK(3));
85
86 DACK5 : OBUFT port map(I=>DACK_OUT(5) , T=>T_DACK, O=>CN_DACK(5));
87 DACK6 : OBUFT port map(I=>DACK_OUT(6) , T=>T_DACK, O=>CN_DACK(6));
88 DACK7 : OBUFT port map(I=>DACK_OUT(7) , T=>T_DACK, O=>CN_DACK(7));
89 ----------------------------------------------------------
90 --DRQ_MODULE
91 DRQ0 : OBUFT port map(I=>DRQ_OUT(0) , T=>T_DRQ, O=>CN_DRQ(0));
92 DRQ1 : OBUFT port map(I=>DRQ_OUT(1) , T=>T_DRQ, O=>CN_DRQ(1));
93 DRQ2 : OBUFT port map(I=>DRQ_OUT(2) , T=>T_DRQ, O=>CN_DRQ(2));
94 DRQ3 : OBUFT port map(I=>DRQ_OUT(3) , T=>T_DRQ, O=>CN_DRQ(3));
95
96 DRQ5 : OBUFT port map(I=>DRQ_OUT(5) , T=>T_DRQ, O=>CN_DRQ(5));
97 DRQ6 : OBUFT port map(I=>DRQ_OUT(6) , T=>T_DRQ, O=>CN_DRQ(6));
98 DRQ7 : OBUFT port map(I=>DRQ_OUT(7) , T=>T_DRQ, O=>CN_DRQ(7));
99 ----------------------------------------------------------
100 --IRQ_MODULE
101 IRQ03 : OBUFT port map(I=>IRQ_OUT(3) , T=>T_IRQ, O=>CN11_B_IRQ(3));
102 IRQ04 : OBUFT port map(I=>IRQ_OUT(4) , T=>T_IRQ, O=>CN11_B_IRQ(4));
103 IRQ05 : OBUFT port map(I=>IRQ_OUT(5) , T=>T_IRQ, O=>CN11_B_IRQ(5));
104 IRQ06 : OBUFT port map(I=>IRQ_OUT(6) , T=>T_IRQ, O=>CN11_B_IRQ(6));
105 IRQ07 : OBUFT port map(I=>IRQ_OUT(7) , T=>T_IRQ, O=>CN11_B_IRQ(7));
106
107 IRQ09 : OBUFT port map(I=>IRQ_OUT(9) , T=>T_IRQ, O=>CN11_B_IRQ9);
108
109 IRQ10 : OBUFT port map(I=>IRQ_OUT(10) , T=>T_IRQ, O=>CN12_D_IRQ(10));
110 IRQ11 : OBUFT port map(I=>IRQ_OUT(11) , T=>T_IRQ, O=>CN12_D_IRQ(11));
111 IRQ12 : OBUFT port map(I=>IRQ_OUT(12) , T=>T_IRQ, O=>CN12_D_IRQ(12));
112
113 IRQ14 : OBUFT port map(I=>IRQ_OUT(14) , T=>T_IRQ, O=>CN12_D_IRQ(14));
114 IRQ15 : OBUFT port map(I=>IRQ_OUT(15) , T=>T_IRQ, O=>CN12_D_IRQ(15));
115 ----------------------------------------------------------
116
117 end Behavioral;
118
119
```

SIMULAÇÕES DO FIRMWARE DO MPR

As simulações foram realizadas com entrada de uma onda quadrada de 50MHz e 3 ciclos de onda portadora por bit do código de Barker. Abaixo são apresentados os gráficos das simulações dos sete códigos de Barker presentes na tabela 1, em ordem crescente (o primeiro é o código de 2 bits e o último do de 13 bits). Em todos os casos “clk” é a onda portadora, “trig” é o gatilho de disparo do sinal e “sinal”, é o sinal de saída do circuito para a excitação do transdutor.



APÊNDICE B

Código do Filtro Casado

CÓDIGO DO FILTRO CASADO

```
main5.vhd Fri Sep 04 11:29:50 2009
Page 1
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.STD_LOGIC_ARITH.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 ---- Uncomment the following library declaration if instantiating
26 ---- any Xilinx primitives in this code.
27 --library UNISIM;
28 --use UNISIM.VComponents.all;
29
30 entity main5 is
31 Generic (cpb : Integer := <arbitrário>; --Número de ciclos por bit
32 nbits : Integer := <2, 3, 4, 5, 7, 11 OU 13>; --Número de bits
33 apmc : Integer := <até 5>; --Amostras Por Meio Ciclo
34 bt1 : Integer := <definido a partir do código pela tabela 3>;
35 bt2 : Integer := <definido a partir do código pela tabela 3>;
36 bt3 : Integer := <definido a partir do código pela tabela 3>;
37 bt4 : Integer := <definido a partir do código pela tabela 3>;
38 bt5 : Integer := <definido a partir do código pela tabela 3>;
39 bt6 : Integer := <definido a partir do código pela tabela 3>;
40 A : Integer := 1; -- # de transições de fase no código de barker
41 barr : Integer := 8); --Número de bits no barramento de dados
42
43 Port ( ck : in STD_LOGIC;
44 rst: in STD_LOGIC;
45 xk : in STD_LOGIC_VECTOR ((barr-1) downto 0);
46 yk : out STD_LOGIC_VECTOR ((barr-1) downto 0));
47 end main5;
48
49 architecture Behavioral of main5 is
50 ------------------------------------------------------------------------
---------------------
51 type Registradores is array (1 to ((2*apmc*cpb*nbits)+apmc)) of
STD_LOGIC_VECTOR((barr-1
downto 0);
52 type VetorB is array (1 to 6) of STD_LOGIC_VECTOR((barr-1) downto 0);
53 type VetorI is array (1 to 6) of Integer;
54 signal ykm1, ykm2, ykm3, ykm4, ykm5 : STD_LOGIC_VECTOR ((barr-1) downto
0) :=("00000000"
55 signal reg: Registradores
:=("00000000","00000000","00000000","00000000","00000000",
"00000000","00000000","00000000","00000000","00000000","00000000","00000000
",
"00000000","00000000","00000000","00000000","00000000","00000000","00000000
",
"00000000","00000000","00000000","00000000");
56 signal bt : VetorI;
57 ------------------------------------------------------------------------
---------------------
main5.vhd Fri Sep 04 11:29:50 2009
Page 2
58 begin
59
60 -- Efetua Reset no
61 --with rst select
62 -- set <= '0' when '1',
63 -- set when others;
64 ------------------------------------------------------------------------
---------------------
65 bt(1)<=bt1;
66 bt(2)<=bt2;
67 bt(3)<=bt3;
68 bt(4)<=bt4;
69 bt(5)<=bt5;
```

```
70 bt(6)<=bt6;
71
72 Filtro : process(ck)
73 variable mci, mcf, temp, ypn, ypn2, yk0 : STD_LOGIC_VECTOR ((barr-1)
downto 0) :=(
"00000000"); -- Variáveis auxiliares
74 variable L: Integer;
75 variable yp : VetorB
:=("00000000","00000000","00000000","00000000","00000000","00000000
); -- Armazena As somas dos semiciclos imediatamente posteriores às
transições
76 --variable bt : VetorI := (1,7,9,10,11,12); -- Bit anterior à primeira
transição
77 begin
78
79 if rising_edge(ck) then
80 L:=(2*apmc*cpb*nbits)+apmc; -- # de registradores necessários no sistema
81 reg(1)<=xk;
82 for i in 2 to L loop
83 reg(i)<=reg(i-1);
84 end loop;
85
86 mci:=mci+xk-reg(apmc); --Armazena os valores do meio ciclo inicial
87 mcf:=mcf+reg(L-apmc)-reg(L); --Armezena os valores do meio ciclo final
88
89 for i in 0 to (barr-1) loop
90 temp(i):='0';
91 end loop;
92 for i in 1 to A loop
93 yp(i):=yp(i)+reg(2*apmc*cpb*bt(i))-reg((2*apmc*cpb*bt(i))+apmc);
94 ypn((barr-1) downto 1):=yp(i)((barr-2) downto 0);
95 ypn(0):='0';
96 if ((i rem 2)=1) then
97 temp:=temp-ypn;
98 else
99 temp:=temp+ypn;
100 end if;
101
102 end loop;
103
104 yk0:=mci-ykm5+temp+mcf;
105 ykm1<=yk0;
106 ykm2<=ykm1;
107 ykm3<=ykm2;
108 ykm4<=ykm3;
109 ykm5<=ykm4;
110 yk<=yk0;
111 end if;
112 end process Filtro;
113 ----------------------------------------------------------------------------
----------------------
114 end Behavioral;
115
116
```

APÊNDICE C

Simulações Lógicas dos Filtros Casados

Os gráficos apresentam a resposta do filtro usando notação de complemento de 2.

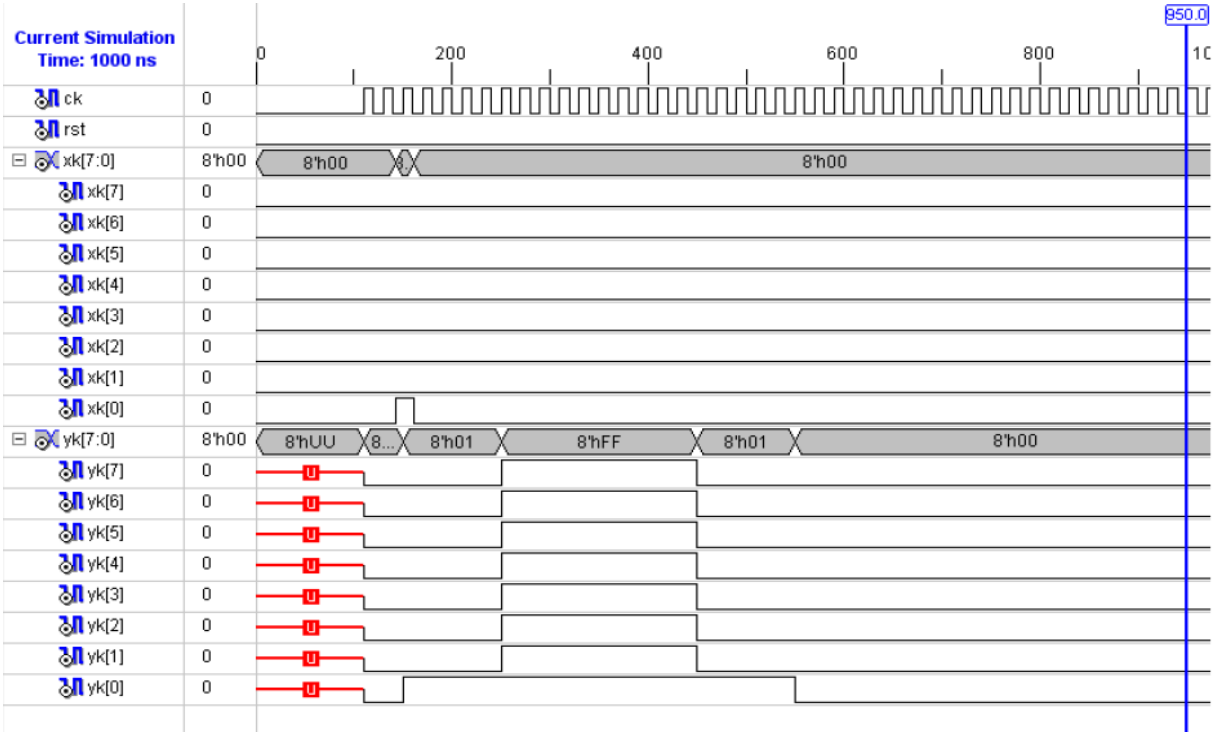


Gráfico 1: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 2 bits, com 1 ciclo por bit e 10 amostras por ciclo.

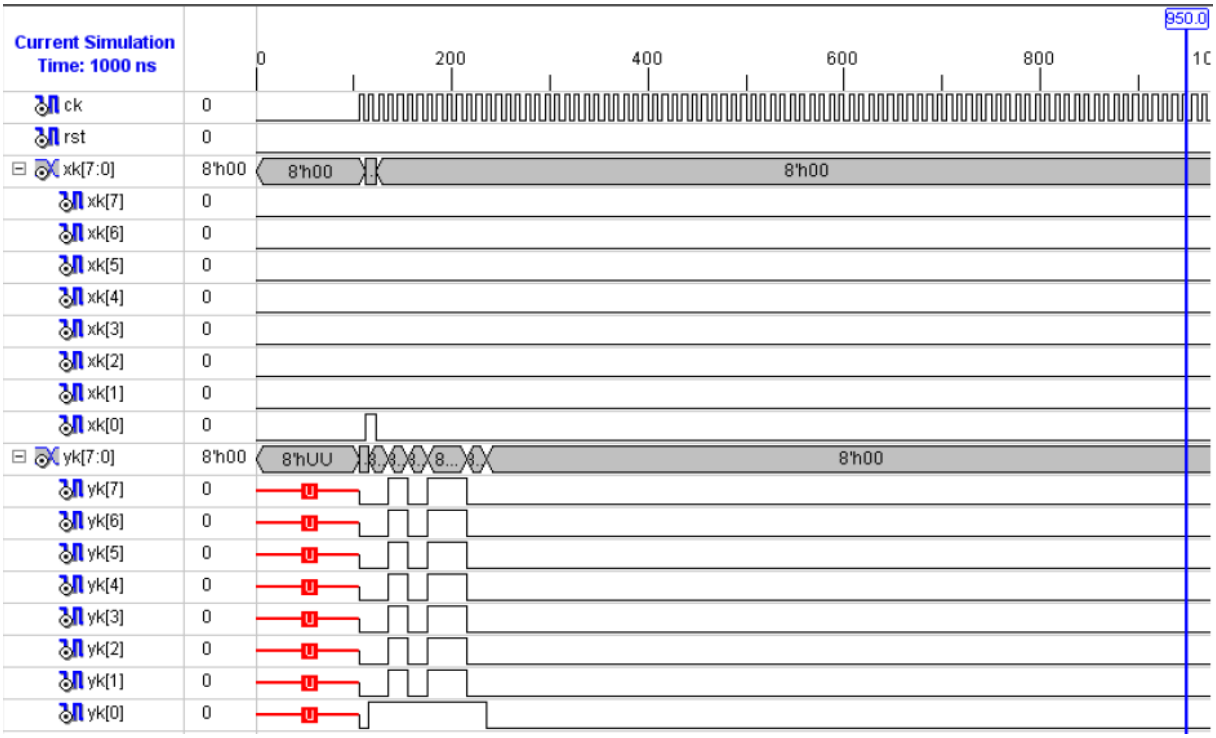


Gráfico 2: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 3 bits, com 1 ciclo por bit e 4 amostras por ciclo.

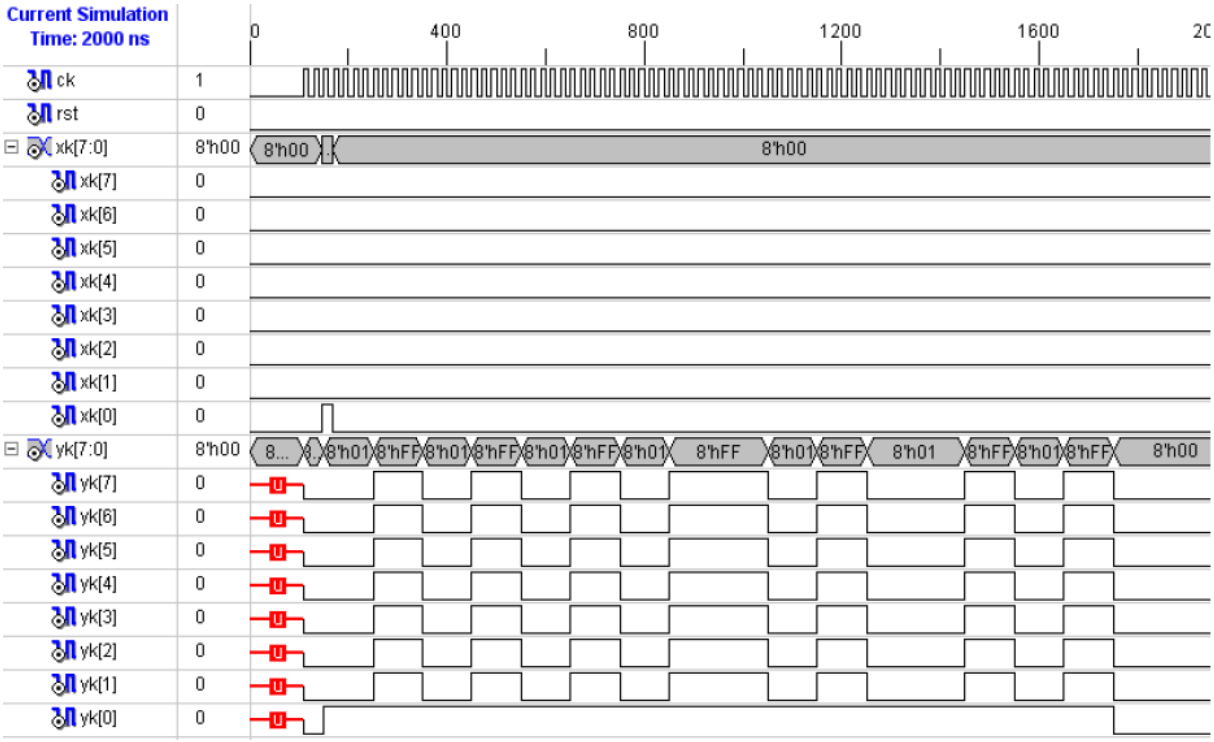


Gráfico 3: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 4 bits, com 2 ciclo por bit e 10 amostras por ciclo.

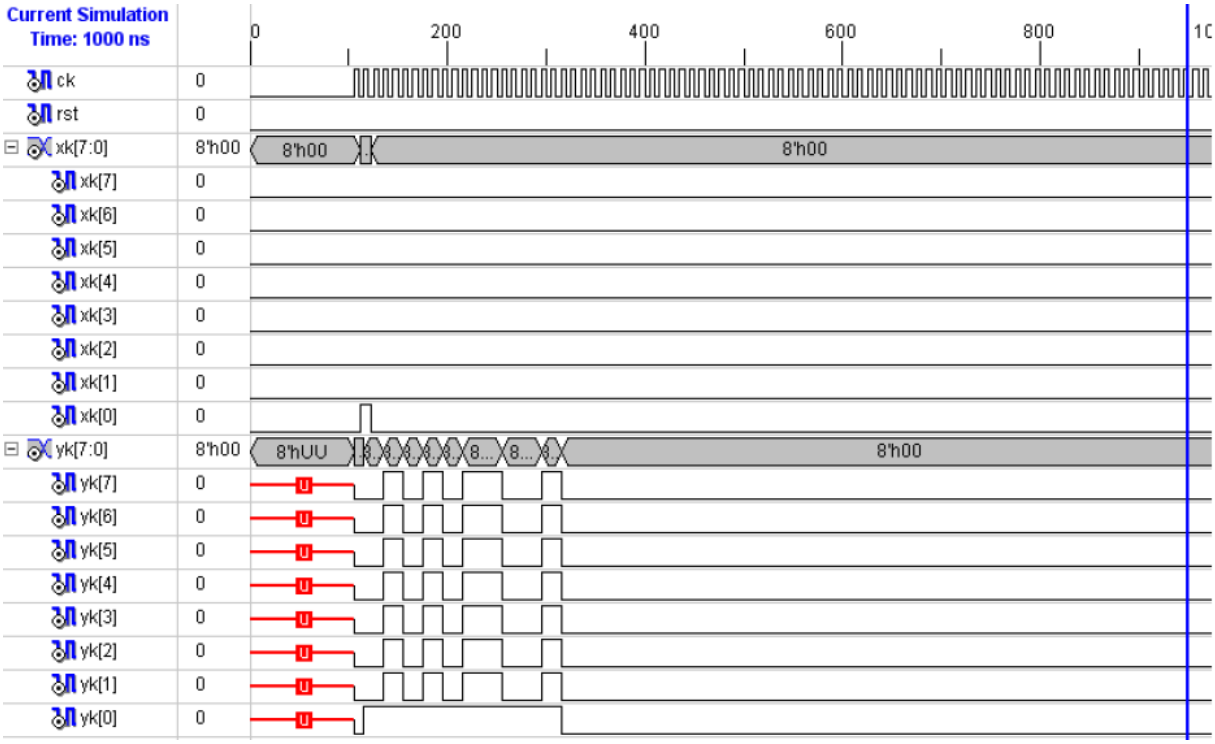


Gráfico 4: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 5 bits, com 1 ciclo por bit e 4 amostras por ciclo.

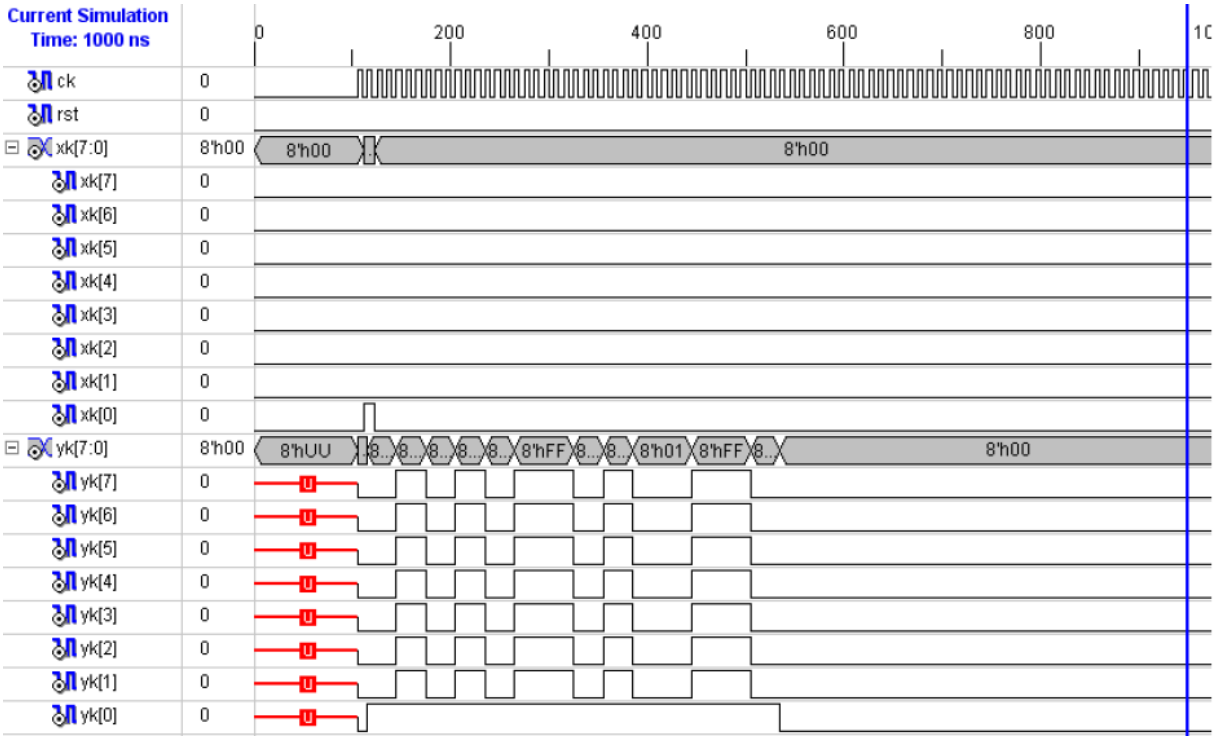


Gráfico 5: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 7 bits, com 1 ciclo por bit e 6 amostras por ciclo.

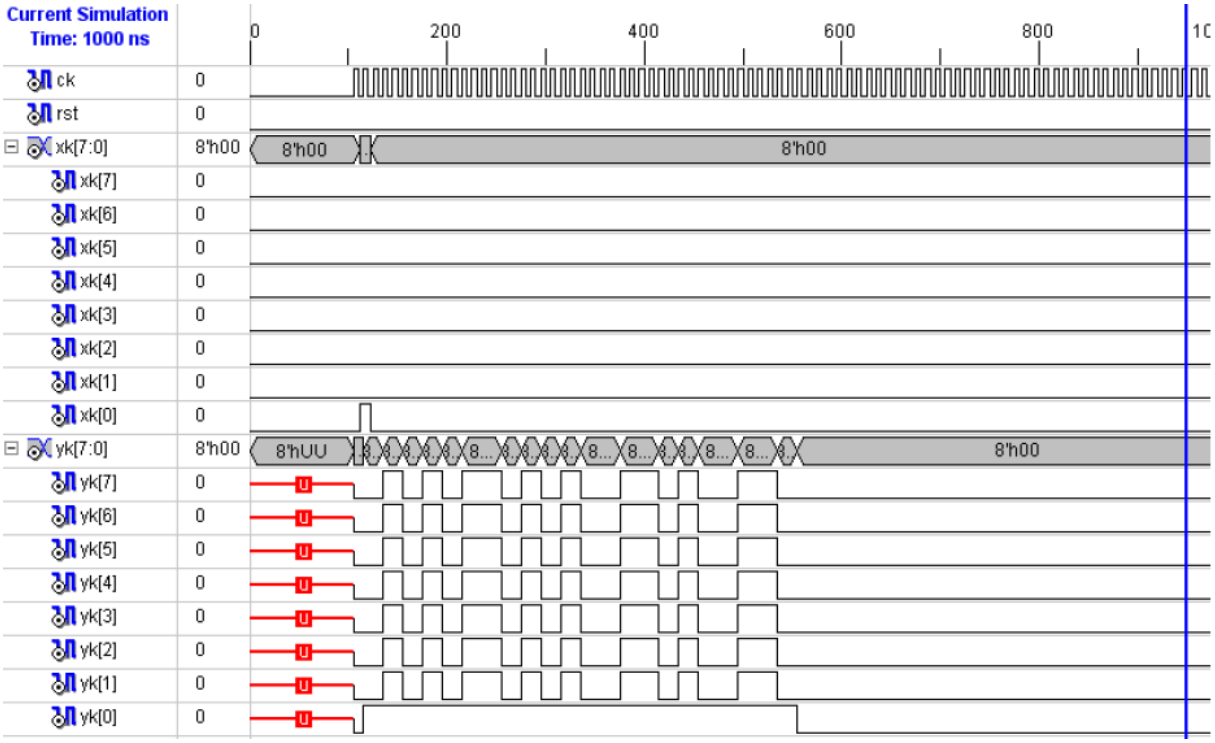


Gráfico 6: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 11 bits, com 1 ciclo por bit e 4 amostras por ciclo.

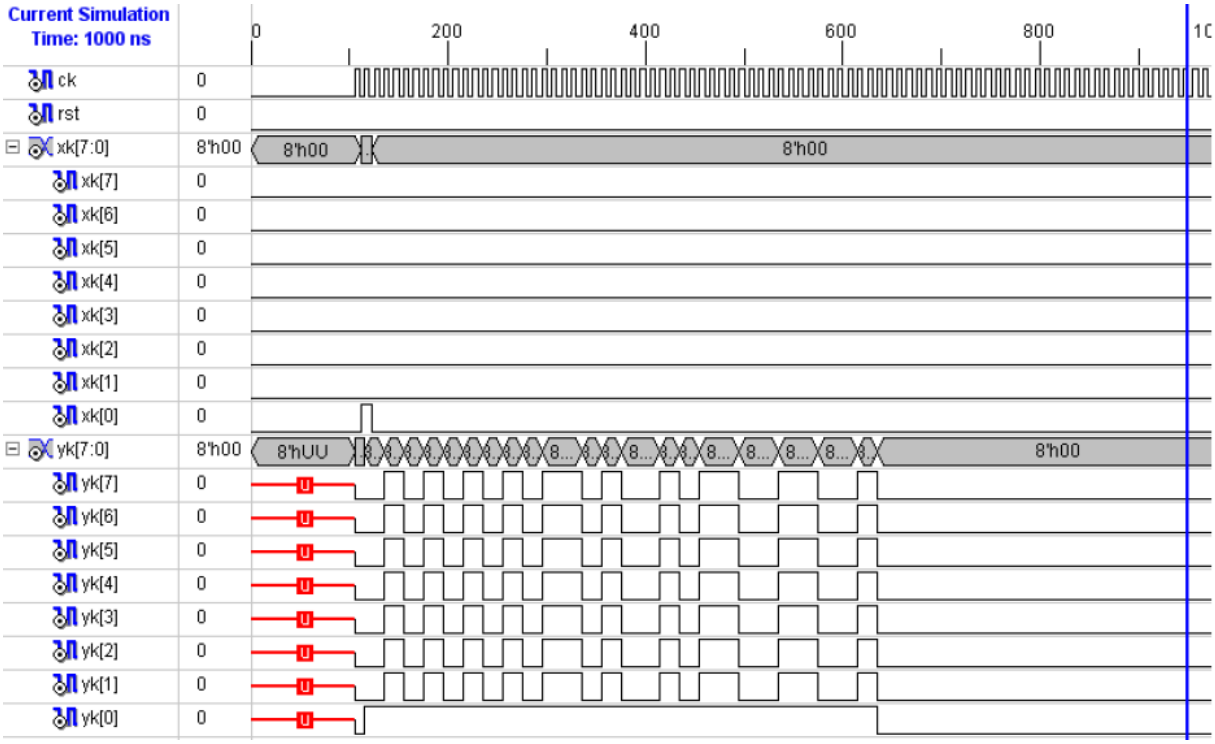


Gráfico 7: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 13 bits, com 1 ciclo por bit e 4 amostras por ciclo.

APÊNDICE D

Simulações de *Timing* dos Filtros Casados

Os gráficos apresentam a resposta do filtro usando notação de complemento de 2.

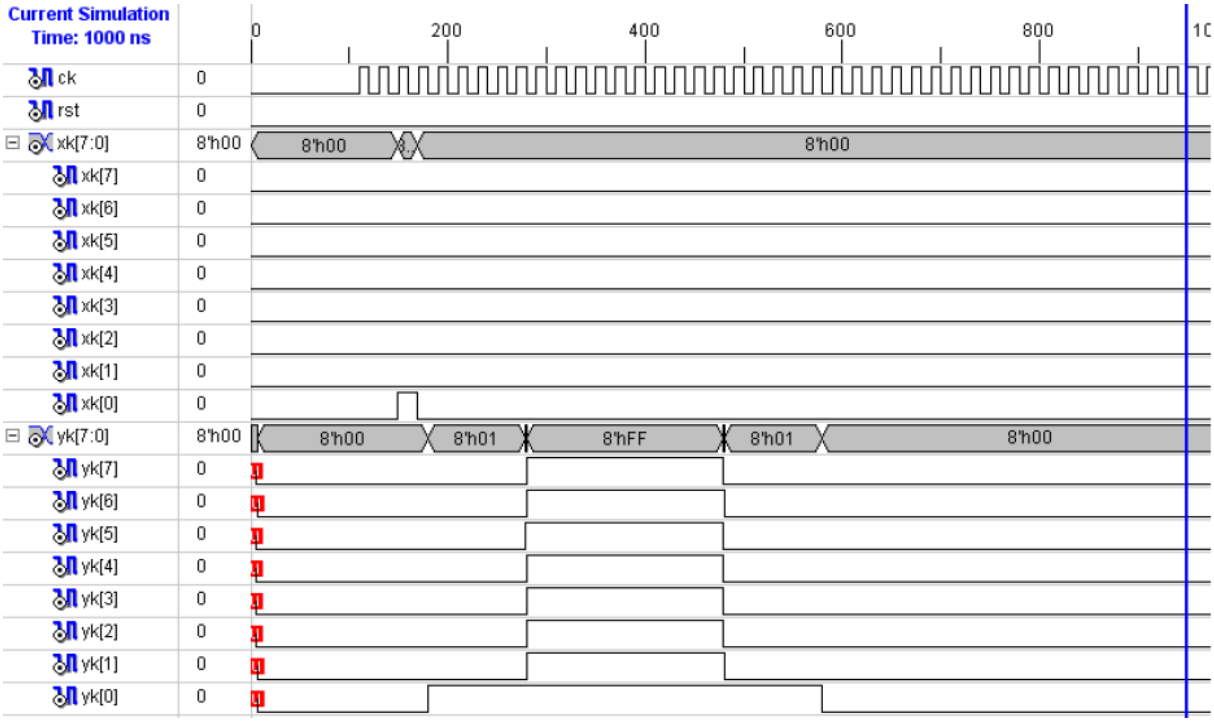


Gráfico 1: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 2 bits, com 1 ciclo por bit, 10 amostras por ciclo, *clock* com ciclo de 20ns e *Input Setup Time* de 1ns.

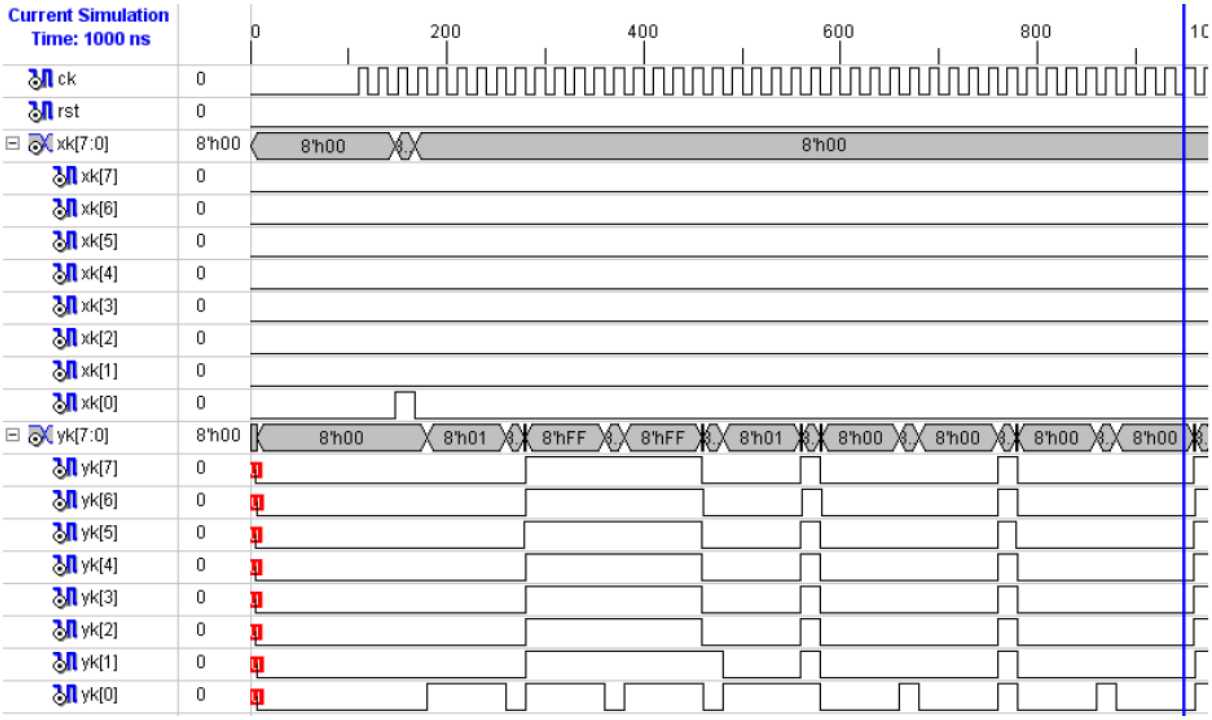


Gráfico 2: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 2 bits, com 1 ciclo por bit, 10 amostras por ciclo, *clock* com ciclo de 20ns e *Input Setup Time* de 2ns.

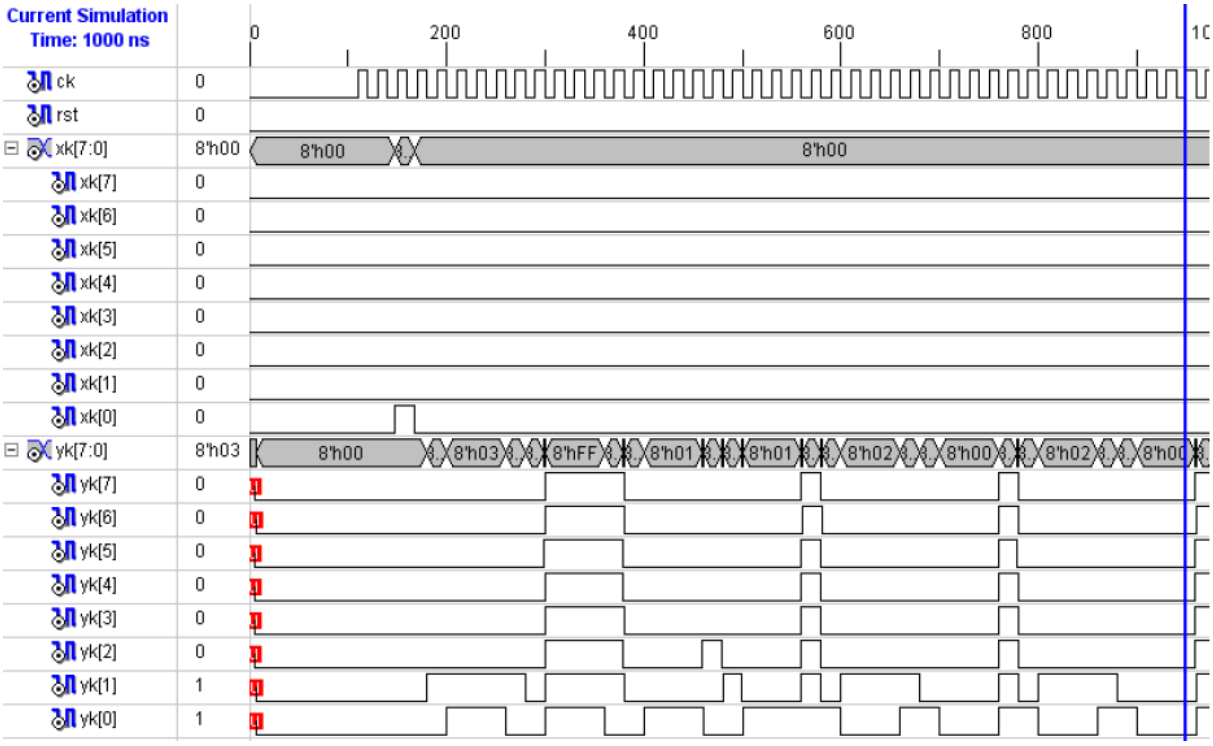


Gráfico 3: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 2 bits, com 1 ciclo por bit, 10 amostras por ciclo, *clock* com ciclo de 20ns e *Input Setup Time* de 3ns.

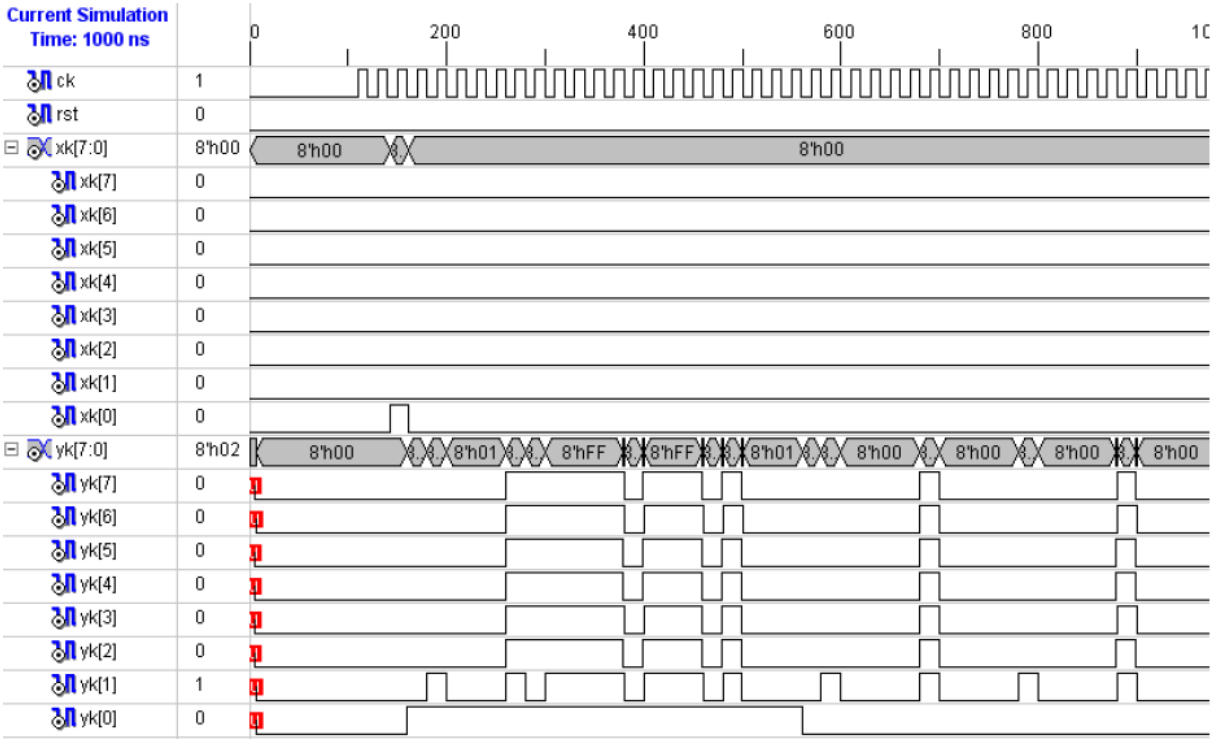


Gráfico 4: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 2 bits, com 1 ciclo por bit, 10 amostras por ciclo, *clock* com ciclo de 20ns e *Input Setup Time* de 8ns.

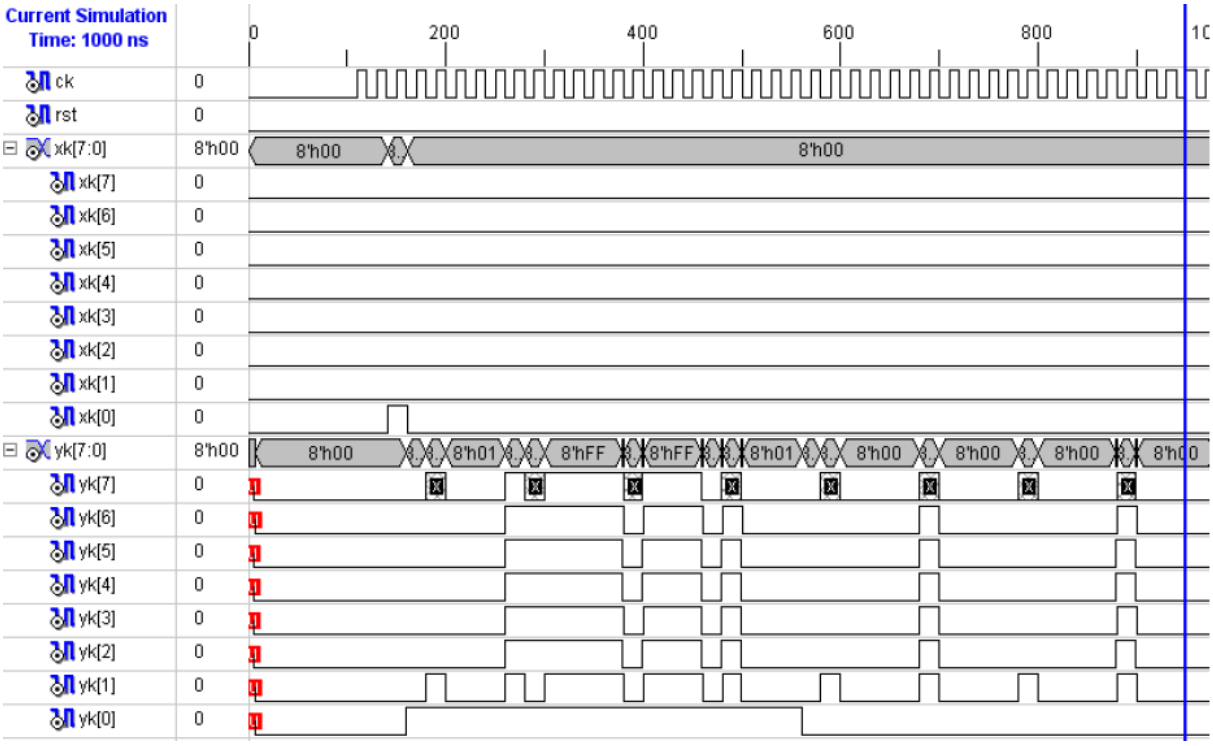


Gráfico 5: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 2 bits, com 1 ciclo por bit, 10 amostras por ciclo, *clock* com ciclo de 20ns e *Input Setup Time* de 9ns.

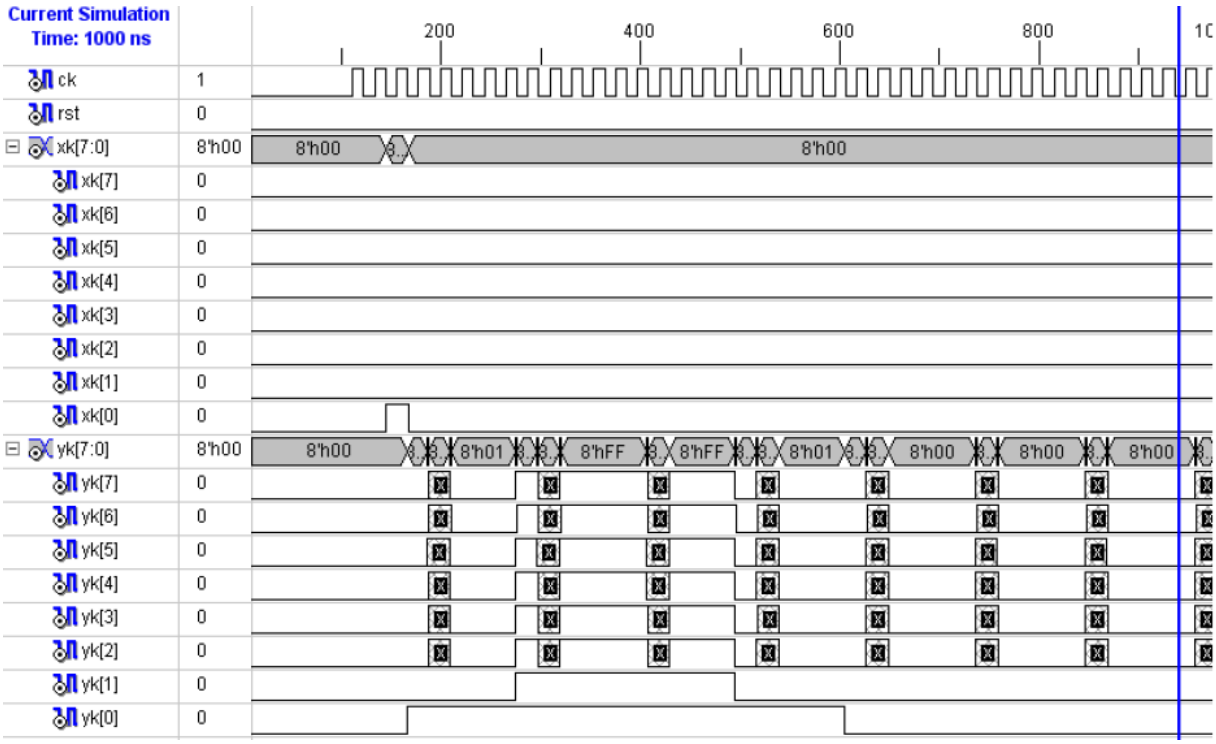


Gráfico 6: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 2 bits, com 1 ciclo por bit, 10 amostras por ciclo, *clock* com ciclo de 22ns e *Input Setup Time* de 10ns.

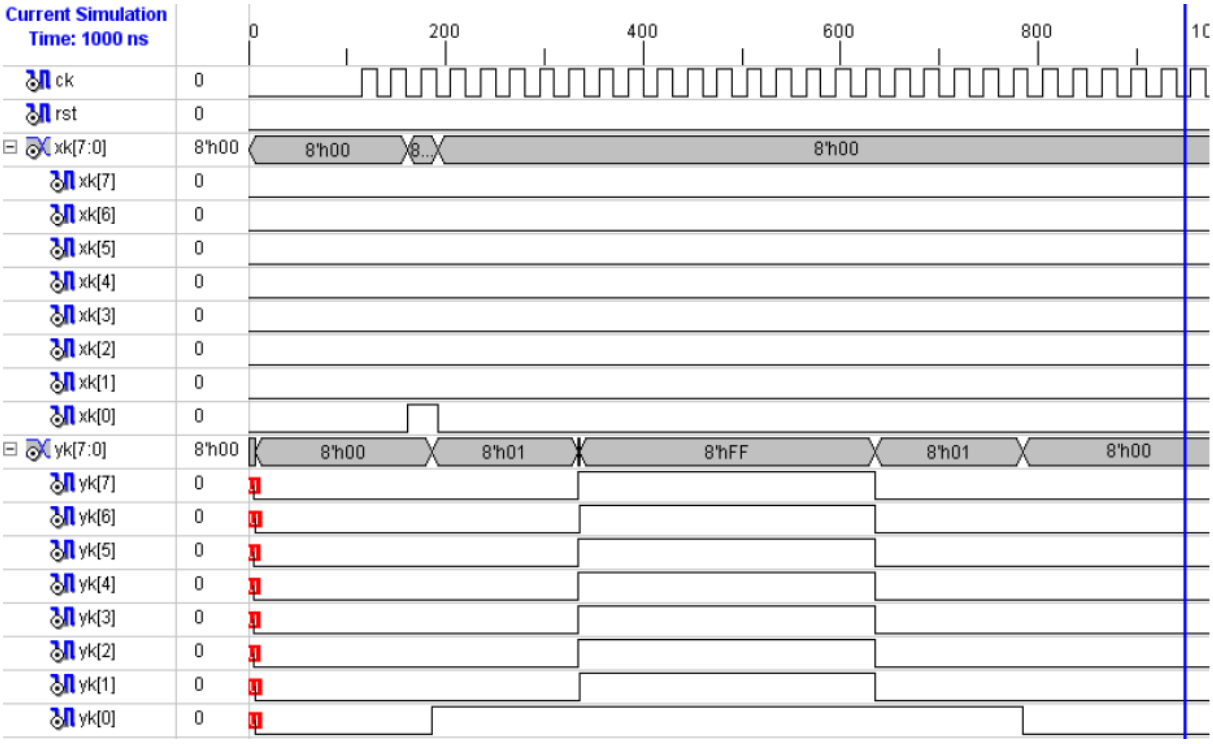


Gráfico 7: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 2 bits, com 1 ciclo por bit, 10 amostras por ciclo, *clock* com ciclo de 30ns e *Input Setup Time* de 13ns.

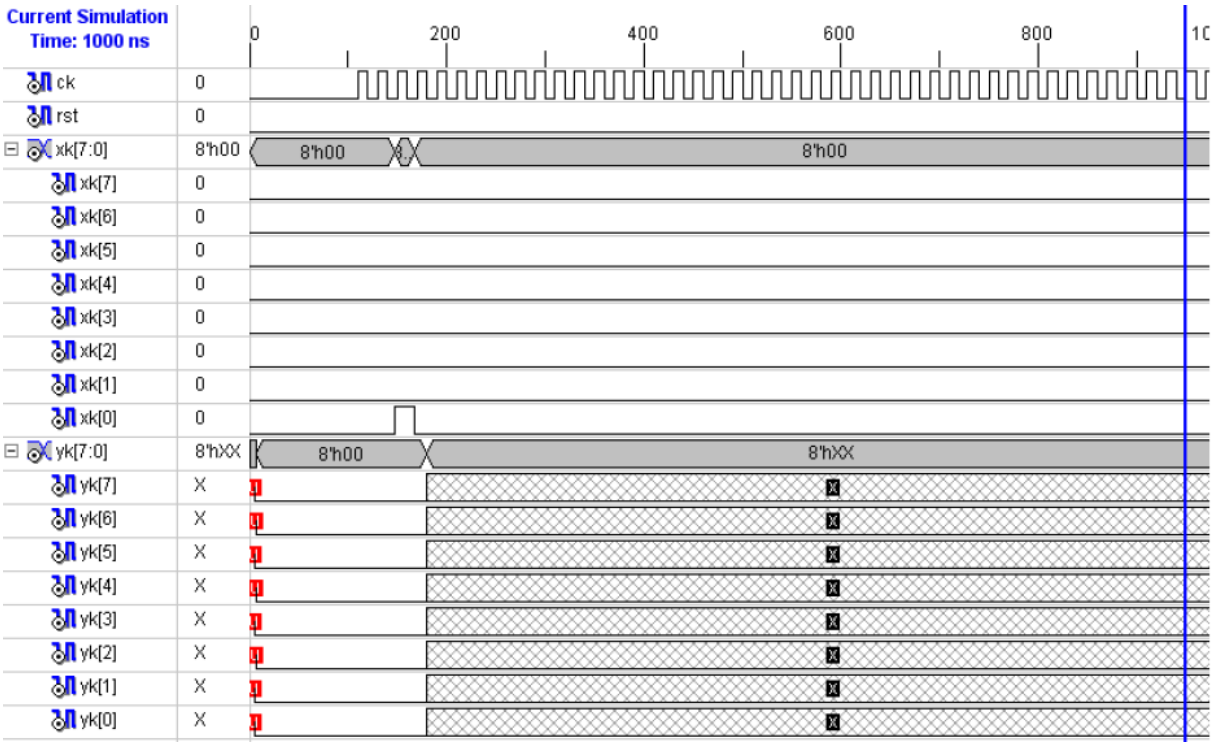
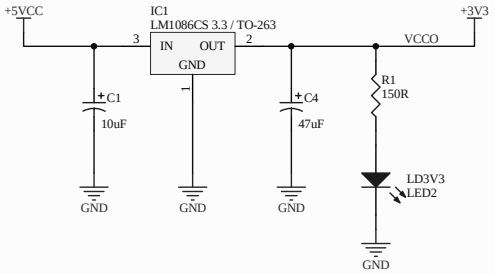


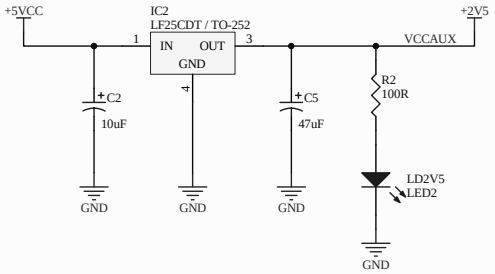
Gráfico 8: Simulação de resposta no tempo do filtro casado de um sinal modulado por código de Barker de 4 bits, com 2 ciclo por bit, 10 amostras por ciclo, *clock* com ciclo de 20ns e *Input Setup Time* de 2ns.

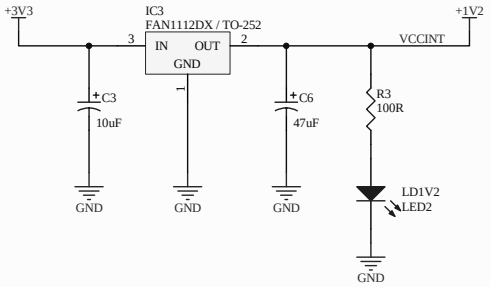
ANEXO A

Esquema Elétrico do Módulo Pulsador/Receptor

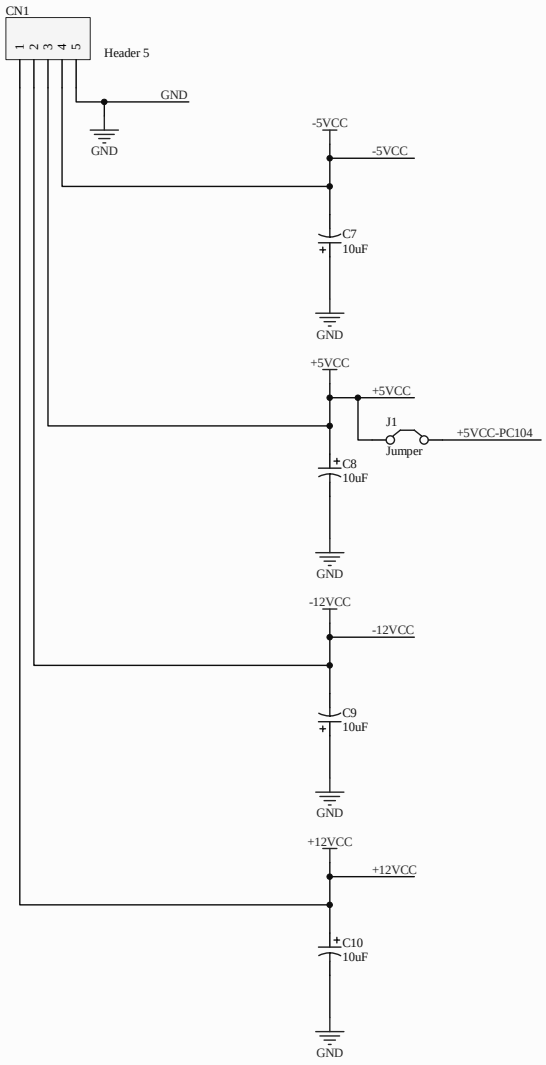
ALIMENTACOES DE +1,2VCC, +2,5VCC e 3,3VCC



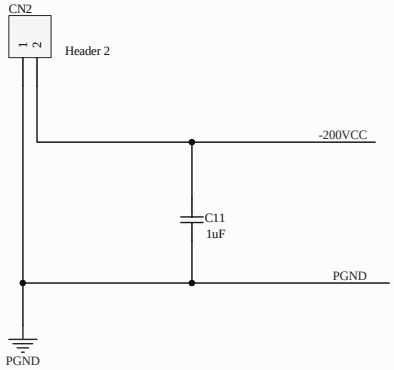


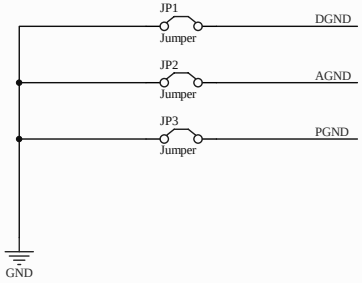


ENTRADA DE ALIMENTACAO AUXILIAR
DE BAIXA POTÊNCIA



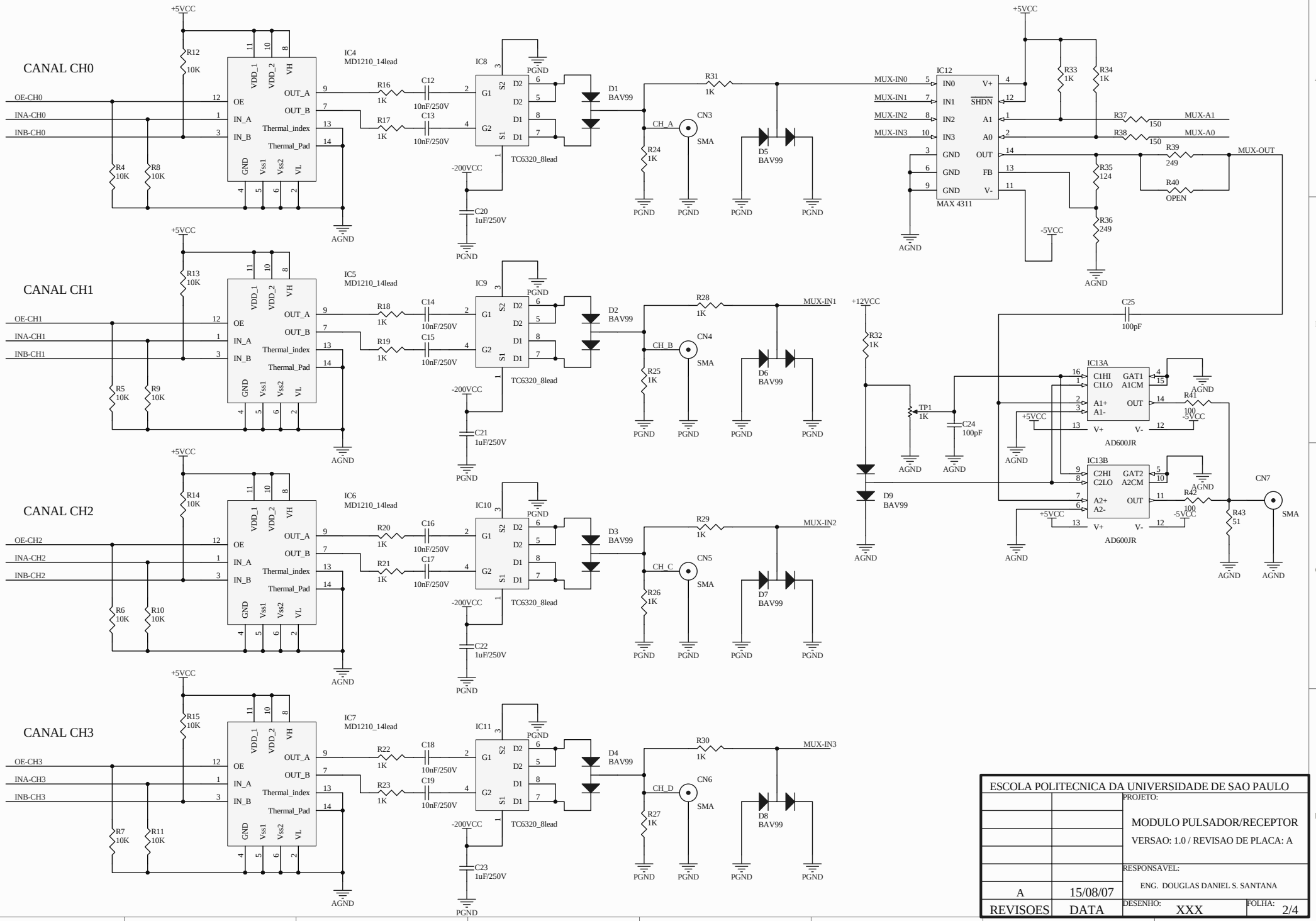
ENTRADA DE ALIMENTACAO DE ALTA TENSÃO





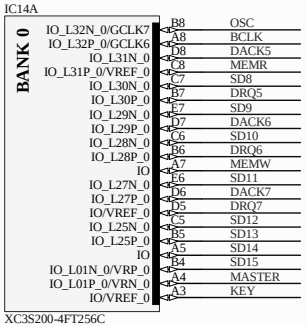
ESCOLA POLITECNICA DA UNIVERSIDADE DE SAO PAULO			
		PROJETO: MODULO PULSADOR/RECEPTOR VERSAO: 1.0 / REVISAO DE PLACA: A	
		RESPONSAVEL: ENG. DOUGLAS DANIEL S. SANTANA	
A	15/08/07		
REVISOES	DATA	DESENHO: XXX	FOLHA: 1/4

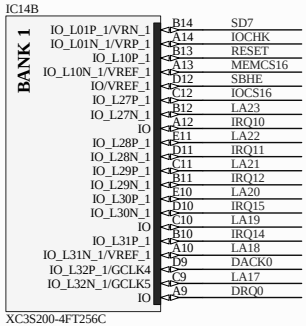
PULSADORES E RECEPTORES - CANAIS CH0 A CH3

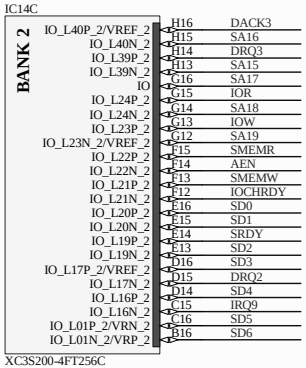


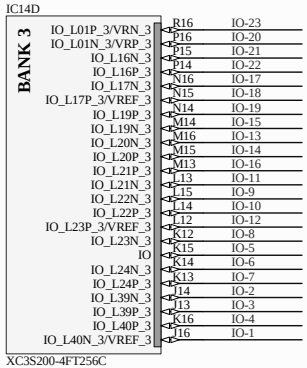
ESCOLA POLITECNICA DA UNIVERSIDADE DE SAO PAULO		
		PROJETO: MODULO PULSADOR/RECEPTOR
		VERSAO: 1.0 / REVISAO DE PLACA: A
		RESPONSAVEL: ENG. DOUGLAS DANIEL S. SANTANA
A	15/08/07	
REVISOES	DATA	DESENHO: XXX — FOLHA: 2/4

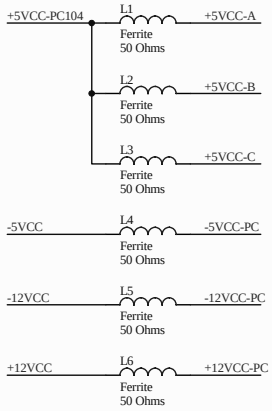
CONECTOR PC104











CN11

DGND	B1	A1	IOCHK
RESET	B2	A2	SD7
+5VCC-A	B3	A3	SD6
IRQ9	B4	A4	SD5
-5VCC-PC	B5	A5	SD4
DRQ2	B6	A6	SD3
-12VCC-PC	B7	A7	SD2
SRDY	B8	A8	SD1
+12VCC-PC	B9	A9	SD0
KEY	B10	A10	IOCHRDY
SMEMW	B11	A11	AEN
SMEMR	B12	A12	SA19
IOW	B13	A13	SA18
IOR	B14	A14	SA17
DACK3	B15	A15	SA16
DRQ3	B16	A16	SA15
DACK1	B17	A17	SA14
DRQ1	B18	A18	SA13
REFRESH	B19	A19	SA12
BCLK	B20	A20	SA11
IRQ7	B21	A21	SA10
IRQ6	B22	A22	SA9
IRQ5	B23	A23	SA8
IRQ4	B24	A24	SA7
IRQ3	B25	A25	SA6
DACK2	B26	A26	SA5
TC	B27	A27	SA4
BALE	B28	A28	SA3
+5VCC-B	B29	A29	SA2
OSC	B30	A30	SA1
DGND	B31	A31	SA0
DGND	B32	A32	DGND

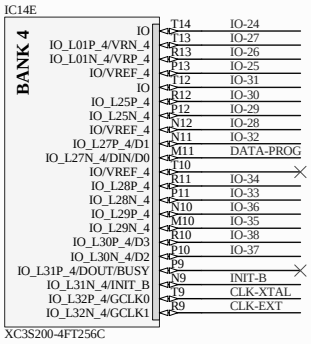
CN12

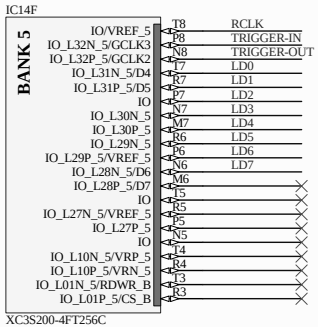
DGND	C0	D0	DGND
SBHE	C1	D1	MEMCS16
LA23	C2	D2	IOCS16
LA22	C3	D3	IRQ10
LA21	C4	D4	IRQ11
LA20	C5	D5	IRQ12
LA19	C6	D6	IRQ15
LA18	C7	D7	IRQ14
LA17	C8	D8	DACK0
MEMR	C9	D9	DRQ0
MEMW	C10	D10	DACK5
SD8	C11	D11	DRQ5
SD9	C12	D12	DACK6
SD10	C13	D13	DRQ6
SD11	C14	D14	DACK7
SD12	C15	D15	DRQ7
SD13	C16	D16	+5VCC-C
SD14	C17	D17	MASTER
SD15	C18	D18	DGND
KEY	C19	D19	DGND

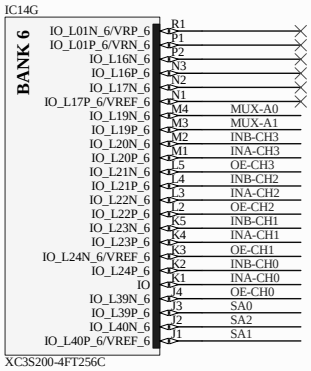
Conector PC104 = 104 pinos totais, 90 pinos de sinais

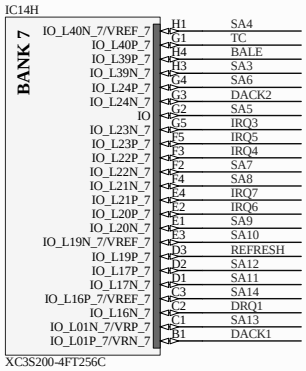
Falta designar pinos na FPGA para acionar os canais e o MUX...

OE-CH0	OE-CH1	OE-CH2	OE-CH3	MUX-A1
INA-CH0	INA-CH1	INA-CH2	INA-CH3	MUX-A0
INB-CH0	INB-CH1	INB-CH2	INB-CH3	

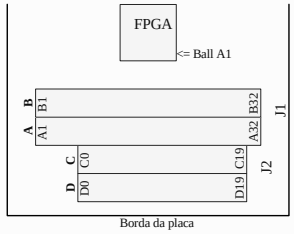


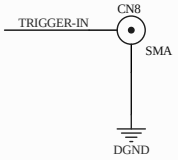


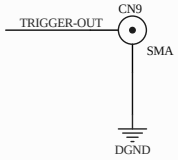


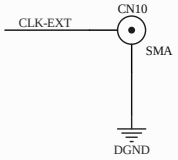


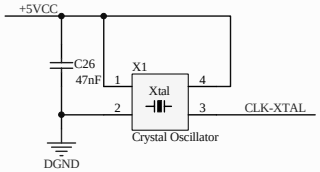
PADRAO MECANICO (VISTA SUPERIOR)



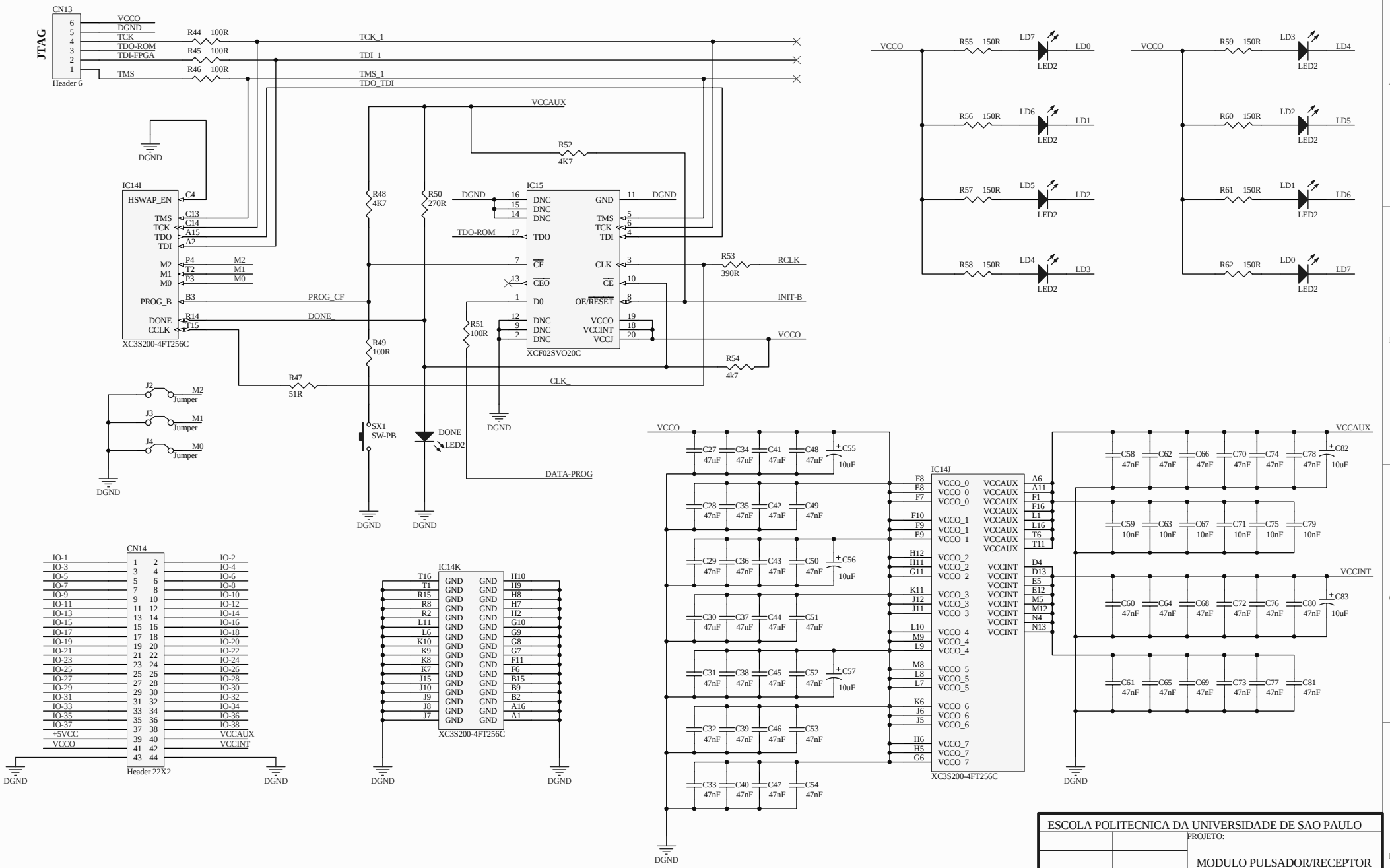








ESCOLA POLITECNICA DA UNIVERSIDADE DE SAO PAULO			
		PROJETO: MODULO PULSADOR/RECEPTOR	
		VERSAO: 1.0 / REVISAO DE PLACA: A	
		RESPONSAVEL: ENG. DOUGLAS DANIEL S. SANTANA	
A	15/08/07		
REVISOES	DATA	DESENHO: XXX	FOLHA: 3/4



ESCOLA POLITECNICA DA UNIVERSIDADE DE SAO PAULO			
		PROJETO: MODULO PULSADOR/RECEPTOR VERSAO: 1.0 / REVISAO DE PLACA: A	
		RESPONSAVEL: ENG. DOUGLAS DANIEL S. SANTANA	
A	15/08/07		
REVISOES	DATA	DESENHO: XXX	FOLHA: 4/4