MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-13027-PRE/8304

EXPERIMENTO APEX – PROGRAMA DE CIRCUITOS BASEADOS EM FIELD PROGRAMMABLE GATE ARRAY

Shridhar Jayanthi*

*Bolsista ITA

Relatório Final de Projeto de Iniciação Científica (PIBIC/CNPq/INPE), orientado pelos Drs. Hanumant Shankar Sawant e Valdivino A de Santiago Junior

INPE
São José dos Campos
2005

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

EXPERIMENTO APEX – PROGRAMA DE CIRCUITOS BASEADOS EM FIELD PROGRAMMABLE GATE ARRAY

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA
(PIBIC/CNPq/INPE)

Shridhar Jayanthi (ITA, Bolsista PIBIC/CNPq)
E-mail: shridhar@redecasd.ita.br

Dr. Hanumant Shankar Sawant (Orientador, DAS/INPE)
E-mail: sawant@das.inpe.br

Eng. Valdivino A. de Santiago Jr. (Orientador, CEA/INPE)
E-mail: valdivino@cea.inpe.br

Junho de 2005

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO

1.1 Objetivos

Este trabalho, iniciado em agosto de 2005 tem como objetivo um aprendizado sobre o uso de tecnologias de dispositivos de lógica programável do tipo Field Programmable Gate Array – FPGA – em experimentos embarcados em satélites.

A importância deste trabalho reside no fato de que os dispositivos FPGA apresentam robustez maior que circuitos digitais por serem dispositivos únicos capazes de suportar uma imensa capacidade de instrução. Além disso, o uso de FPGAs ao invés de microprocessadores num experimento embarcado também apresenta a vantagem de facilmente absorver a função de vários componentes diversos como contadores e latches, reduzindo a quantidade de peças independentes, e viabilizando aumento do clock de operação da placa.

Na primeira etapa deste trabalho foi feita um estudo do funcionamento de dispositivos de lógica programável e de FPGAs, uma revisão dos conceitos principais envolvidos no desenho de circuitos digitais. Numa segunda fase, houve um estudo um pouco mais aprofundado no relatório técnico do experimento ORCAS e nos documentos do experimento APEX para que uma elaboração do documento de requisitos do FPGA I/O fosse elaborado. Na terceira fase foi feita a implementação do FPGA I/O em ambiente de simulação e elaboração do documento de requisitos.

1.2 Dispositivos de lógica programável

No desenho de projetos de eletrônica digital de alta complexidade, diversas são as soluções existentes no mercado, sendo a escolha de uma delas condicionada pelas características de uso. O surgimento da tecnologia de Circuitos Integrados de Larga Escala (VLSI) e seu uso em produtos da área comercial foi capaz de produzir equipamentos de altíssimo desempenho de maneira massiva e barata, através da gravação de máscaras em chips de silício. Esse tipo de desenvolvimento possui características desejáveis do ponto de vista de produção, mas em se tratando de projeto e prototipação de produtos, a gravação em chips de silício torna-se um processo muito caro.

Existem algumas soluções que são utilizadas por projetistas para reduzir estes custos. A mais imediata é o uso de ambientes de simulação. Essa técnica é bastante interessante e útil na eliminação de erros grosseiros e auxilia na tomada de decisões de projeto mas não é eficaz para fazer um ajuste fino de detalhes de temporização e avaliação do paralelismo dos processos do circuito por causa da natureza seqüencial intrínseca de um microcomputador. Outras soluções comuns até algum tempo atrás era a prototipação utilizando Circuitos Integrados de Pequena Escala (SSI) ou Circuitos Integrados de Média Escala (MSI) e o uso de Masked Programmed Gate Arrays (MPGA), um processo similar ao da elaboração de máscaras para geração direta em silício, porém utilizando matrizes de portas pré determinadas.

Uma opção mais barata e mais viável que surgiu posteriormente foi o uso de Dispositivos de Lógica Programável (PLD). Alguns dos primeiros tipos desse dispositivo foram as Matrizes de Lógica Programável (PLA), e as Lógicas de Matriz Programáveis (PAL).

Os Field Programmable Gate Arrays - FPGA pertencem a uma classe de dispositivos chamada de Field Programmable Logic Devices (FPLD), dispositivos de lógica programável de alta densidade. Esses dispositivos se caracterizam pela grande densidade e capacidade de customização, ideal para prototipação de circuitos lógicos de uma maneira a reduzir custos sem perda de qualidade nos testes.

Na área de pesquisa experimental envolvendo tratamento e análise de dados em tempo real, o uso de elementos de eletrônica digital é necessário para fazer interface com computadores de bordo ou computadores de experimento. Por causa dos altos custos da geração de somente um dispositivo VLSI gerado a partir das técnicas tradicionais de mascara de silício, a solução mais comum adotada era o uso de elementos MSI e SSI ou o uso de softwares embarcados em computadores de bordo. Apesar de este tipo de uso ser satisfatório, ele não apresenta a robustez necessária para alguns experimentos em ambientes hostis, como o cenário de um equipamento de coleta de dados embarcado em satélites. Pare este tipo de experimento, os FPGAs se mostram uma possibilidade bastante atraente.

1.3 Linguagens de descrição de hardware

Desde o início dos projetos de eletrônica eram tipicamente feitos através do desenho de esquemas elétricos onde cada dispositivo escolhido e cada ligação era feita pelo projetista. Este tipo de maneira mais tradicional, bastante funcional e bastante utilizado até recentemente, apresenta limitações óbvias. Quando o projeto do circuito digital começa a ganhar tamanho, é difícil para o responsável pelo projeto se concentrar nas idéias macroscópicas do projeto em um esquema elétrico com várias ligações.

Um avanço no campo de projetos de eletrônica foi o esquema hierárquico, onde um conjunto de dispositivos eletrônicos com uma mesma função era agrupado em um só bloco hierárquico. Este tipo de projeto desenho melhorou a legibilidade de projetos mas esbarrou em uma outra limitação. Com o crescimento da complexidade das funções a serem exercidas pelos sistemas, como a implantação de algoritmos de criptografia, compressão e processamento de sinais digitais, o desenho de esquemas elétricos foi ficando cada vez mais distante da concepção ideológica da função do circuito projetado.

O surgimento das linguagens de descrição de hardware tornou-se um padrão no projeto de sistemas digitais de complexidade alta. Além de apresentar a vantagem de ser um sistema naturalmente voltado para as necessidades de alto nível dos projetos, ele também tem a característica de ser portável, ou seja, a transferência de um produto desenhado para uma tecnologia para outra é bastante suave.

1.4 Interfaces para o FPGA I/O

Experimentos embarcados em satélites costumam ser controlados por progamas instalados no computador de bordo do satélite. Por isso é necessário que exista uma interface de comunicação entre o experimento e os barramentos padrão especificados no satélite. Além disso, o FPGA I/O, desenhado neste projeto, tem a função de enviar comandos aos componentes internos do experimento. Fica portanto, como responsabilidade deste componente estas duas comunicações.

CAPÍTULO 2 – FUNDAMENTAÇÃO TEÓRICA

2.1 Desenho de dispositivos lógicos digitais

Dispositivos de lógica digital são elementos eletrônicos que são capazes de gerar um resultado discreto a partir de estímulos de entrada discretos e uma memória de trabalho finita. Estes circuitos trabalham sempre com sinais binários representados por nível ou por variação de nível e são máquinas determinísticas finitas. A analogia evidente entre a definição de elementos de lógica digital e a acepção fundamental do que são programas de computador não é coincidência; programas de computador são instruções a ser executadas por nada mais que máquinas digitais extremamente complexas – os computadores digitais.

O desenho de dispositivos de lógica digital é uma área da engenharia elétrica que une diversos campos do conhecimento com um propósito prático. O componente mais fundamental deste campo é, evidentemente, a lógica matemática, mas é também importante estudar análise de teoria de circuitos e sistemas eletrônicos, técnicas de ciência de computação, além da importância grande da experiência neste ramo.

As técnicas mais eficientes de dispositivos lógicos digitais costumam seguir a filosofia de projetos top-down. Nesta técnica, as funcionalidades de mais alto nível são descritas e somente depois as peculiaridades de baixo nível são analisadas.

Seguindo a heurística top-down, por exemplo, podemos desenhar um registrador bastante simples de 8 bits acessado por um barramento tri-state de 8 bits, um comando de leitura e um comando de escrita. As funcionalidades deste registrador são três: (i) armazenar 8 bits, (ii) controlar o acesso ao barramento de entrada e saída do circuito e (iii) controlar os direitos de gravação e leitura do conteúdo do registrador. A primeira funcionalidade tem como resposta adequada o uso de flip-flop, a segunda funcionalidade pode ser instalada utilizando um buffer tri-state e a terceira função aparecerá com a especificação mais detalhada do buffer tri-state e do flip-flop utilizado.

A importância desta técnica é que aqui, a necessidade é quem pede o uso de estratégias evitando a tentação existente de utilizar estratégias e dispositvos mais familiares aos projetistas que podem não ser as mais adequadas.

No projeto de baixo nível, o desenho de circuitos lógicos se reduz basicamente a escolher um conjunto de portas lógicas e conexões entre estas portas que gerem um resultado desejado. Por se tratar de um desenho de baixo nível, estas especificações podem ser descritas em tabelas simples. O exemplo abaixo é o de um somador de 1 bit com saídas de 2 bits. A expressão resultante é S = X + Y. Como a saída de um circuito é binária, o resultado de S, em dois bits, é representado por seus dígitos binários S(1) e S(0).

X	Y	S(1)	S(0)
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	1

Agrupando os resultados desejados por bit, podemos dizer que:
- S(0) é igual a 0 somente quando os bits X e Y são 0.
- S(1) é igual a 1 somente quando os bits X e Y são 1.

Com essas duas informações, é possível estabelecer uma relação lógica entre os sinais de entrada X e Y e os sinais de saída S(1) e S(2):

- S(0) = X ou Y
- S(1) = X e Y

O desenho das portas lógicas, feitos através do uso de transistores foge do escopo de um projeto de sistema digital, que tem como nível mais básico as "equações lógicas".

Existem técnicas mais específicas de desenho, como as minimizações através dos diagramas de Karnaugh e a escolha entre máquinas seqüenciais de Moore ou Mealy, mas não é objetivo deste relatório ser uma referência para desenho de circuitos digitais.

2.2 Tecnologia de Field Programmable Gate Arrays – FPGA

Conforme foi dito na introdução, os Field Programmable Logic Devices e mais particularmente os Field Programmable Gate Arrays são dispositivos bastante apropriados para uso em prototipação de projetos de lógica digital. Os principais pontos fortes destes dispositivos são a capacidade de customização, o baixo custo de projeto e o alto desempenho destes componentes perante outras soluções VLSI.

Os primeiros dispositivos de lógica programável foram os PLAs e os PALs. Estes dois dispositivos, bastante semelhantes entre si eram conjuntos de portas cujas ligações eram escolhidas e gravadas; as portas eram fixas. Os PLAs, mostrados na figura XXX, eram constituído de um conjunto de entradas que eram levadas a um conjunto de portas lógicas AND, que por sua vez eram levadas a um conjunto de portas lógicas OR. Desta forma, o somador de dois bits acima poderia ser facilmente implementado utilizando uma destas portas lógicas AND e um das portas lógicas OR. Este tipo de dispositivo era capaz de, num único elemento, realizar a função de dezenas de elementos MSI, apresentando-se como uma solução interessante para prototipação.

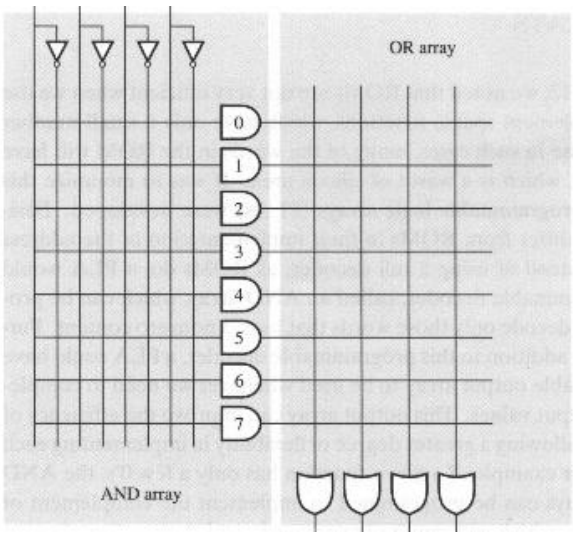


Ilustração 1 - Construção de PLAs. Quando queremos programar um circuito nos PLAs, o procedimento é escolher quais são os cruzamentos nos quais uma conexão entre os sinais devem ser feitas. Desta forma, podemos desenhar um componente lógico.

Após um maior desenvolvimento da pesquisa em materiais semi-condutores, chegou-se a o FPGA. O FPGA é similar aos PLAs e PALs por ser também um conjunto de portas lógicas já definidas no chip de silício entre as quais são feitas as ligações para realizar a lógica que queremos desenhar. Todavia os FPGAs, mostrado na figura XXX apresentaram um grande avanço em dois aspectos.

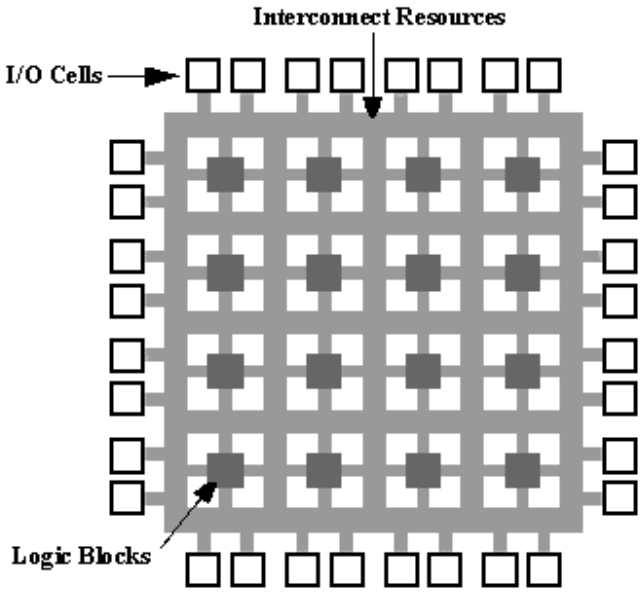


Ilustração 2 - Esquema de um FPGA. Aqui podemos ver as células lógicas (Logic Blocks) os barramentos internos (Interconnect Resources). A programação em FPGAs, diferente da de PLAs, consiste em escolher quais são as conexões entre as portas e a saídas e quais serão as funções exercidas pelas células lógicas. Repare no aumento de conexões que potenciais a serem feitas. Esta versatilidade reflete-se no aumento da capacidade de dispositivos feitos em FPGA.

O primeiro diz respeito ao arranjo das portas. As portas nos FPGAs são alinhadas em uma matriz, ao contrário das portas em PLA, o que aumenta a flexibilidade de uso dessas portas. Com isso, é possível a um conjunto de sinais passarem por um conjunto de portas quantas vezes for viável dentro da arquitetura do sistema, não sendo somente limitado como no PLA, onde só era permitido o uso de duas portas concatenadas.

O segundo avanço foi na idéia de células lógicas. As portas agora não tem somente uma única função definida, elas podem ter capacidade de implementar várias funções e a função a ser exercida passa a ser uma escolha do projetista. Com isso, as portas ganharam uma versatilidade que, juntamente com o arranjo matricial, aumentaram a capacidade de uso de FPGAs de forma quase infinita.

A complexidade dos dispositivos de FPGA mais modernos permitem o uso de milhares de portas e com quantidades de pinos de entrada/saída da ordem de dezenas, o que permitiria até que um subconjunto de instruções de um microprocessador pudesse ser colocado dentro de um FPGA. Esta complexidade também gerou uma necessidade de se ter mecanismos automáticos de projetos de FPGAs, soluções essas fornecidas pelos próprios fabricantes, em geral.

2.3 Linguagem de descrição de hardware VHDL

Pouco após o início do desenvolvimento de projetos de eletrônica digital, viu-se que o desenho de circuitos usando somente esquemas elétricos e diagramas de bloco hierárquicos não era suficiente para desenhar componentes complexos que traziam consigo elementos puramente algorítmicos como circuitos de compressão de dados. A necessidade foi logo respondida com a criação das Linguagens de Descrição de Hardware (HDL).

As HDL são conceitualmente semelhantes às tradicionais linguagens de programação correntemente utilizadas em software. Mas existem alguns pontos de divergência que são bastante importantes de se destacarem. As HDL são capazes de descrever um componente eletrônico em termos de função ou de arquitetura enquanto que uma linguagem de computação tradicional costuma permitir somente descrição funcional. Além disso, um componente eletrônico pode operar múltiplas instruções paralelamente, enquanto um programa de computador é seqüencial. Isto evidencia uma necessidade de um ambiente de simulação capaz de gerar um falso paralelismo, que é implementado através de operações orientadas por evento e atraso. Neste modelo, o programa a ser rodado, ao invés de executa sequencialmente cada uma das instruções, gera uma "agenda" de processos e executa estes processos no momento agendado (atraso) e verifica a cada iteração se existe algum processo que sofreu alteração (evento).

Existe um número grande de HDLs, dentre as quais se destacam as linguagens Verilog e VHDL, esta última sendo a utilizada no trabalho.

A linguagem VHDL – VLSI Hardware Description Language, definida pela norma IEEE 1076-1987 e revisada na norma IEEE 1076-1993, é uma linguagem padronizada por uma força tarefa e é utilizada por vários fornecedores de tecnologias VLSI, para a geração de máscaras para gravação em pastilhas de silício, FPGA ou até mesmo para geração de circuitos constituídos de MSI ou LSI.

Uma distribuição que utiliza VHDL é constituída de várias partes:

- A definição da linguagem propriamente dita.
- Tipos de dados padrão utilizados (signais, barramentos, etc...) constantes da norma IEEE 1164.
- Uma biblioteca de trabakho WORK reservada para desenho correntemente projetado.
- Pacotes de bibliotecas específicas do fornecedor.
- Pacotes e bibliotecas específicas da indústria/projetista.

Um programa em VHDL pode conter quatro níveis de detalhamento ou abstração de um componente. No nível mais alto, podemos descrever o elemento em termos de comportamento e algoritmo. Num segundo nível, podemos descrevê-lo em termos de transferência de informação nos registradores, método adotado neste projeto. Num terceiro nível, é possível determinar cada uma das funções com especificação de atrasos. Num nível ainda mais profundo podemos determinar detalhadamente os diagramas de temporização. Essa especificação das funções é dependente da tecnologia e geralmente está descrita nas bibliotecas do fornecedor. No caso de um FPGA, por exemplo, este detalhamento equivale a especificar que função será exercida por cada uma das células lógicas. Num desenho de máscara de silício, isto equivaleria a um grupamento de transistores responsáveis por uma função específica, como a de um comparador ou a de um flip-flop.

O uso da VHDL privilegia a metodologia de projeto top-down, mencionada anteriormente, ao dar a possibilidade de se variar o approach em níveis diferentes da abstração. Este processo se dá através da forma como, de maneira semelhante ao da

orientação a objetos, a linguagem permite se separar uma entidade da maneira como ela irá funcionar. Podemos definir uma Entity através das interfaces que ela apresenta, isto é, os sinais com as quais a Entity se comunica com outros componentes e uma Architecture que descreve o comportamento interno desta Entidade, que pode ser comportamental ou estrutural. É possível para uma Entity atribuir-se diferentes Architectures e cada combinação dessas e chamada de Configuration. Finalmente, um conjunto de diversas Entitys e suas Configuraritions constituem uma Package, que é o nível mais alto de uma entidade.

A familiarização com a VHDL se deu através da distribuição Actel embutida no Ambiente de Desenvolvimento integrado Libero, fornecido juntamente com o FPGA.

CAPÍTULO 3 – MATERIAIS E MÉTODOS

3.1 Dispositivo FPGA da Actel

O dispositivo utilizado durante o projeto e simulação do FPGA I/O foi um componente genérico da família APA. O projeto não foi feito direcionado para nenhuma família específica por esta ser uma decisão de projeto ainda não tomada. Todavia, a generalidade das tarefas a serem realizadas pela FPGA I/O permite que esta decisão possa ser tomada mais adiante.

Devido à existência de um ambiente capaz de integrar todos as seqüências de um projeto de hardware digital, não foi necessário se ater aos detalhes técnicos do dispositivo utilizado, sendo suficiente desenhar o projeto em VHDL do protótipo.

3.2 Ambiente de Desenvolvimento Libero

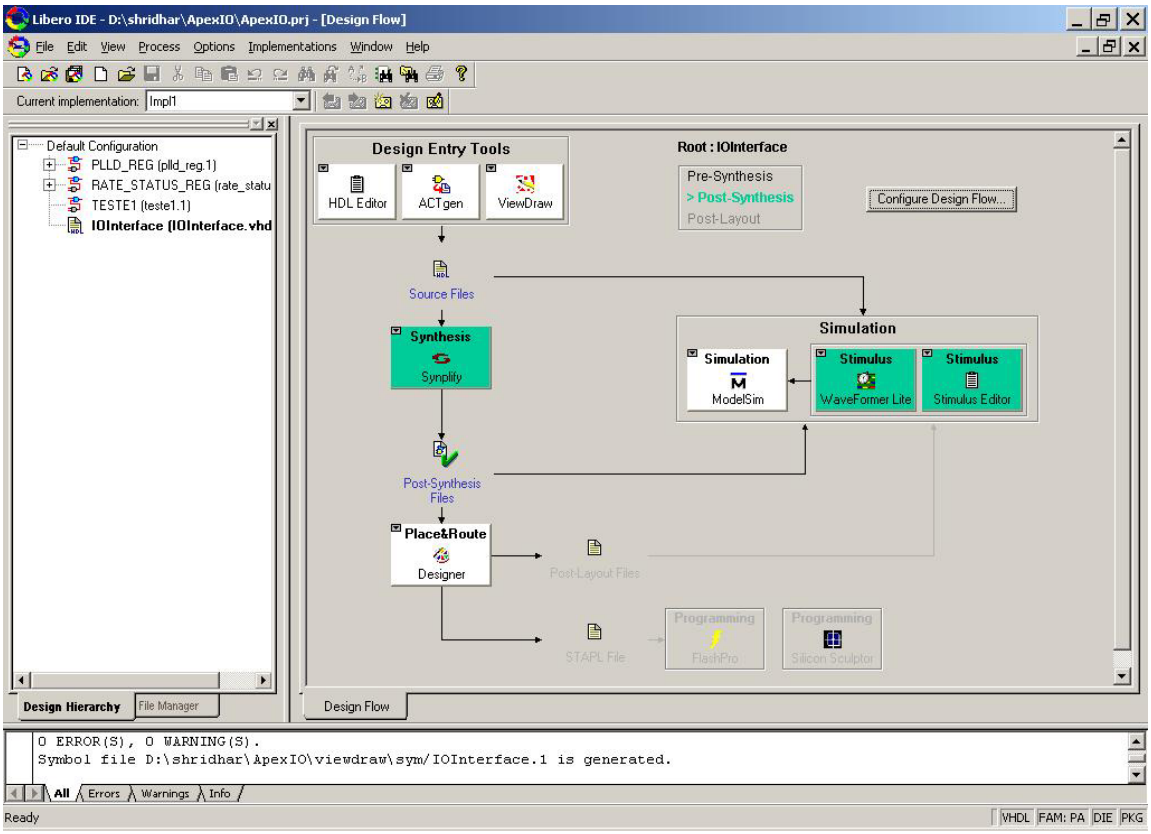


Ilustração 3 - Ambiente de simulação Libero mostrando o fluxo de projeto

O ambiente de desenvolvimento Libero, produzido pela fornecedora de FPGAs Actel tem uma estrutura que lembra ambientes de desenvolvimento integrado de software como os ambientes da Borland. A complexidade de cada uma das fases de um processo de desenho de hardware (projeto, testes pré-síntese, síntese, testes pós síntese) faz com que haja a necessidade de se agrupar vária soluções "terceirizadas". Desta forma a suíte Libero traz consigo os softwares Synplicity, responsável pelas sínteses, ModelSim, responsável

pelas simulações, WaveForm Lite, responsável pela geração de conjuntos de teste, além de diversos outros programas que cuidam de otimização do layout, gravação de FPGAs, temporização detalhada, geração de circuitos a partir de Schematics.

O atraso na chegada da licença deste programa foi responsável por um atraso significativo no cronograma do projeto, pois sem esta licença não era possível testar os programas testados.

3.3 O experimento APEX

O experimento APEX – Alpha, Proton, Electron, X-ray – contexto dentro do qual se insere este projeto tem como objetivo a análise da radiação de partículas presas no campo magnético da Terra, ou que estão na altitude do satélite, seguindo as linhas do campo.
Este experimento é constituído de duas placas, detectores de partículas além do software embarcado no satélite que será responsável por controlar o experimento (ver apêndice A). Os detectores de partícula irão enviar dados para um circuito discriminador de energias. Estes dados, após discriminação serão enviados para uma FIFO – First In First Out – que por sua vez irá armazenar os dados e aguardar o computador de bordo que irá ler todos os dados e esvaziar esta pilha.

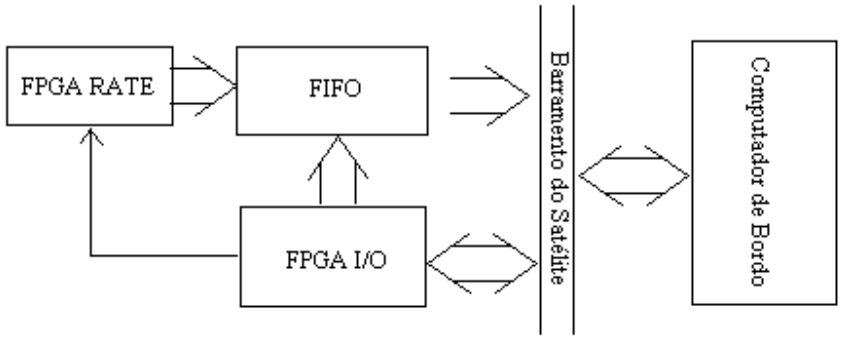


Ilustração 4 - Diagrama de blocos dos componentes do APEX

Neste cenário, a FPGA I/O terá responsabilidade de tratar os comandos enviados pelo computador de bordo e repassar estas informações para os outros componentes do circuito. A figura XXX acima ilustra os componentes com os quais o FPGA I/O deverá se comunicar.

O documento de requisitos do experimento APEX foi baseado no documento de requisitos do experimento ORCAS, também construído no DAS e que foi embarcado nos satélites SACI-1 e SACI-2, uma vez que ambos os experimentos guardam semelhanças fortes. O reaproveitamento de várias idéias e componentes permite uma redução nos erros uma vez que aquele experimento já fora bem sucedida em testes. Os sinais que devem ser tratados pelo I/O FPGA seguem na figura XXX.

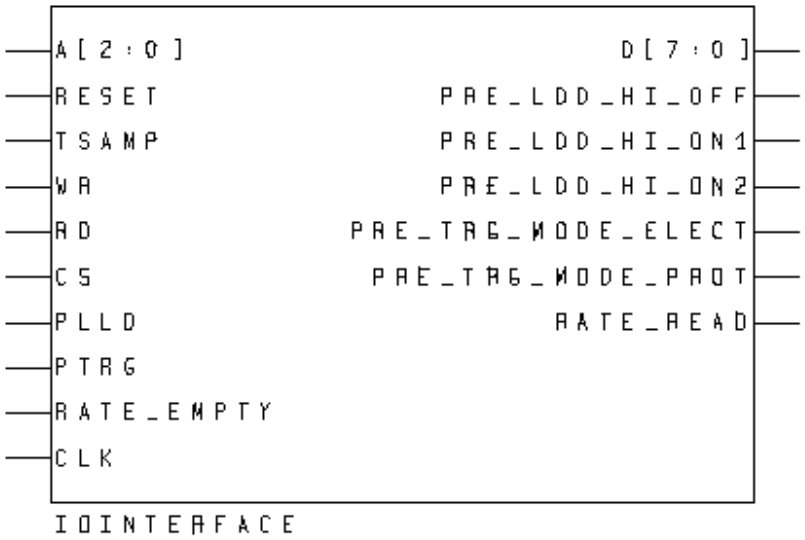


Ilustração 5 - Sinais de interface do FPGA I/O

3.4 Interface do experimento com o computador de bordo

A interface com o computador de bordo constitui-se basicamente de um conjunto pequeno de instruções simples e tradicionais na área de eletrônica digital (Chip Select, Read, Write, Reset), uma entrada advinda de um barramento de endereços e um barramento tri-state de dados compartilhado pelo computador, FPGA I/O e o componente de FIFO.

Os comandos de Read e Write nunca devem ser acionados de maneira simultânea. O comando Chip Select deve ser utilizado sempre que o computador de bordo quiser se comunicar com o computador de bordo. Os comandos Reset, Read e Write são todos assíncronos. O computador de bordo deve sempre seguir a seqüência de temporização bem definida para os processos de leitura e escrita, conforme ilustrado abaixo. O FPGA I/O e a FIFO não podem competir pelo barramento de dados e nunca devem utiliza-lo quando este está ocupado.

Ilustração 6 - Ciclo de Escrita do Experimento APEX

Ilustração 7 - Ciclo de Leitura do Experimento APEX

3.5 Conexão do componente FPGA I/O com outros componentes do circuito

O FPGA I/O deverá se comunicar, dentro do experimento, principalmente com FPGA Rate, responsável pelo controle do fluxo interno de informações e discriminação dos sinais provenientes dos detectores, e com a FIFO responsável pelo armazenamento temporário dos dados do experimento, antes de serem colhidos pelo computador de bordo.

Com o FPGA Rate, as funções de comunicação são basicamente as de repassar os comandos enviados pelo computador de bordo. Esses comandos incluem o nível do comparador que faz as discriminações que servem para identificar qual é a partícula detectada e o intervalo de tempo de amostragem.

Com a FIFO a função é um pouco mais complexa. O FPGA I/O deverá controlar o fluxo de dados na FIFO, verificar se ela está cheia ou vazia (polling), responder ao computador quando este pergunta os resultados do processo de polling e controlar o envio de dados da FIFO para o computador de bordo.

CAPÍTULO 4 – RESULTADOS

4.1 Documento de Especificações do FPGA I/O

Num projeto de Engenharia, como é o caso deste FPGA, um dos produtos mais importantes é o documento de requisitos. Um documento de especificações do FPGA está em desenvolvimento e deverá ser concluído antes do término do período deste trabalho.

4.2 Resultados das simulações

As simulações do projeto feito foram bem sucedidas, mas foram feitas apenas no ambiente do ModelSim. Testes com gravação física da FPGA deverão ainda ser feitos, bem como algumas alterações causadas por mudanças no projeto do experimento APEX que não estão ainda presentes no esquema elétrico. As formas de onda resultantes dos testes de simulação com o projeto atual estão ilustrados na figura abaixo.

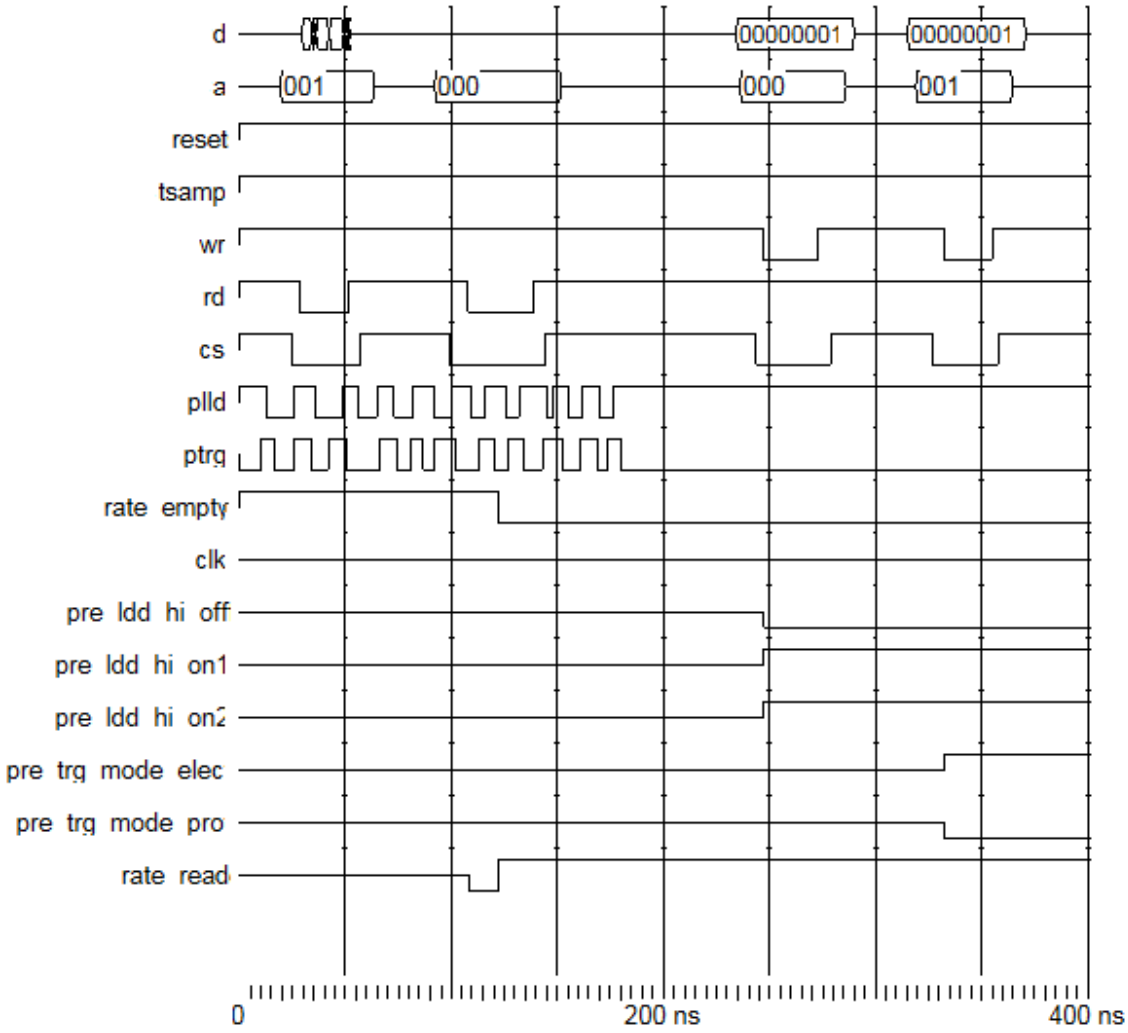


Ilustração 8 - Resultados de simulação. Os tempos na escala são muito menores que os utilizados no experimento embarcado.

4.3 Trabalhos a serem concluídos

Além da finalização do documento de especificações e da atualização dos programas, restam ainda simulações físicas do projeto e adequação do resultado final com os outros componentes. Muitos destes trabalhos deverão ser realizados após o fim do término do período da bolsa, visto que são procedimentos complexos que exigem um tempo maior de dedicação.

CAPÍTULO 5 – CONCLUSÃO

Este trabalho foi satisfatório no aspecto de que ele foi capaz de permitir ao aluno um grande aprendizado no uso de tecnologias sofisticadas de eletrônica digital. Além disso, ele foi capaz de passar conceitos bastante importantes relacionados ao desenvolvimento de experimentos de física prática de complexidade e relevância, além de permitir a agilização do projeto do experimento.

APÊNDICE A – ESQUEMA ELÉTRICO DO EXPERIMENTO APEX

1. INTRODUÇÃO

Estes desenhos esquemáticos simbolizam os componentes eletro/eletrônicos e as ligações da placa AEB1 do experimento ALPHA, PROTON AND ELECTRON MONITORING EXPERIMENT IN MAGNETOSPHERE (APEX), do French Brazilian Microsatellite (FBM).

2 – LISTA DE FIGURAS

FIGURA	DESCRIÇÃO	ARQUIVO	TAMANHO	DATA
01	Pulse Shaping Amplifier - Pre PSA	Prepsajp.PCB	38Kb	05/09/03
02	Discriminator Module - Pre Disc D1	Pred1ad5.PCB	43Kb	25/07/03
03	Discriminator Module - Pre Disc D2	Pred2.PCB	31Kb	25/07/03
04	Discriminator Module - Pre Disc D3	Pred3.PCB	24Kb	25/07/03
05	Discriminator Module - Pre Disc D4	Pred4.PCB	29Kb	25/07/03
06	I/O Control FPGA & Command Relays	Iofpga.PCB	31Kb	25/07/03
07	Rate FPGA e Memórias	Rate32K.PCB	23Kb	12/08/03
08	CPU BUS & FIFO	Cpu.PCB	28Kb	25/07/03
09	Conector AEB1	CNT1.PCB	15Kb	25/07/03

Figura 01 – Pulse Shaping Amplifier - PRE PSA

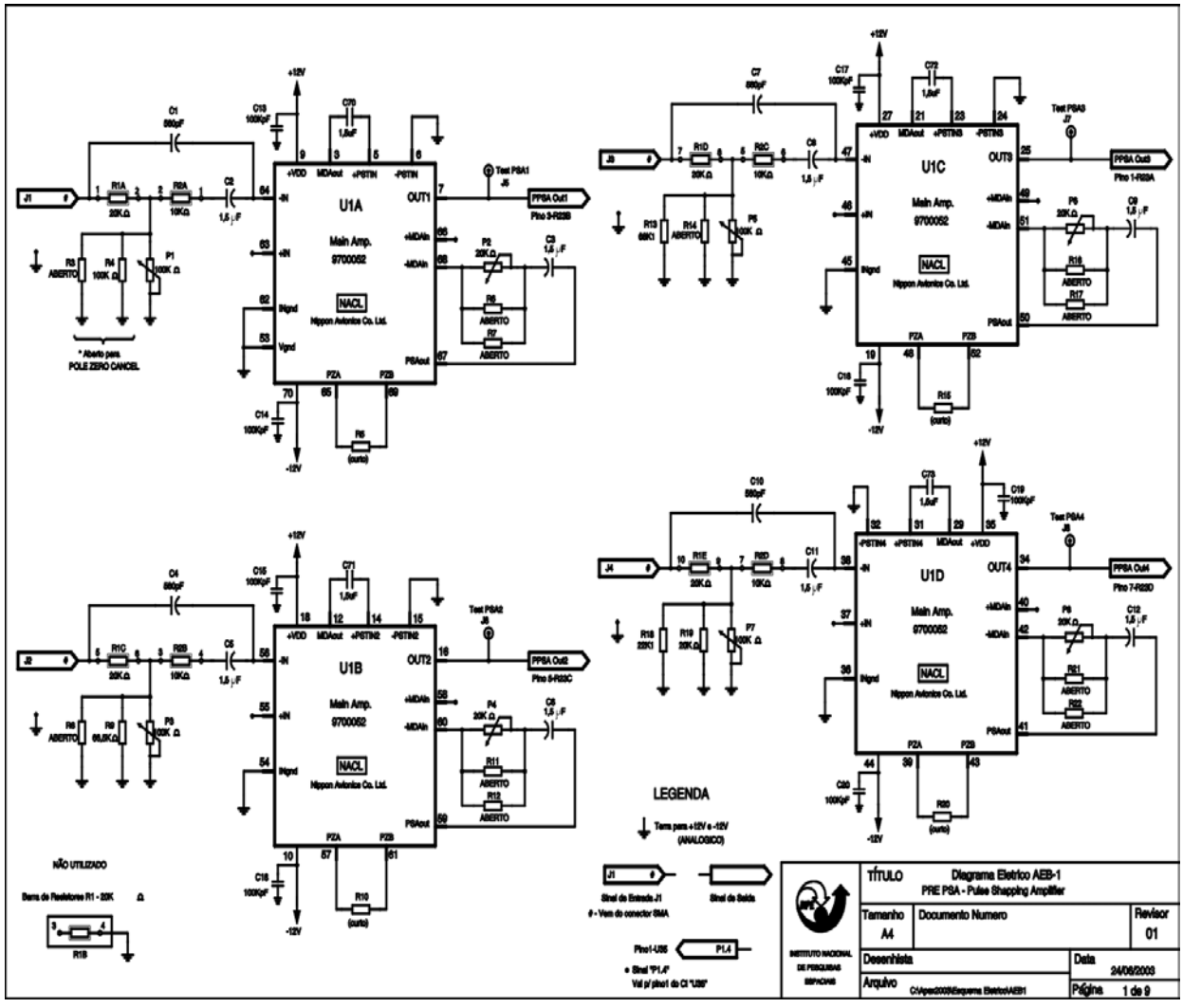


Figura 02 – Discriminator Module - PRE Disc D1

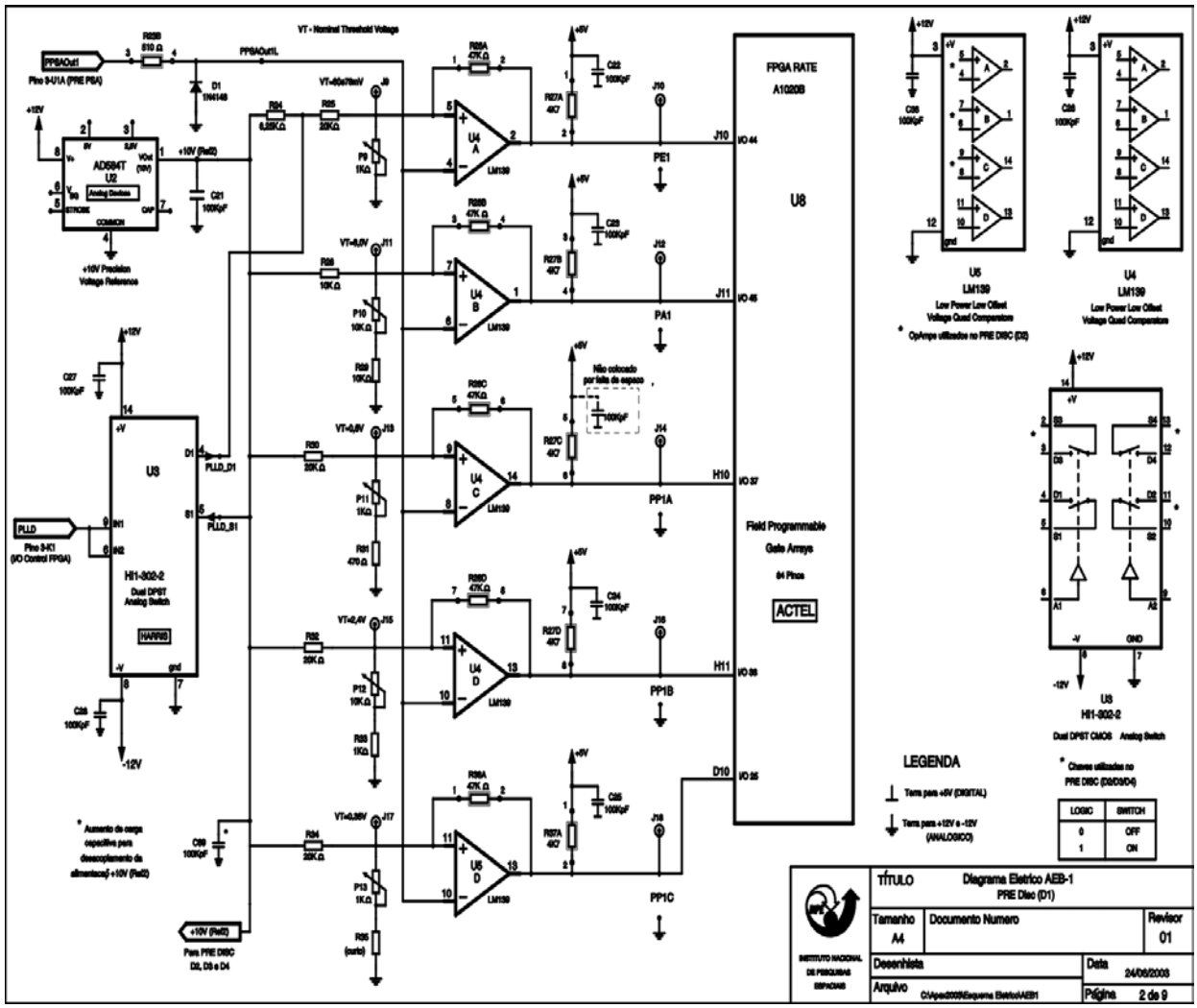


Figura 03 – Discriminator Module - PRE Disc D2

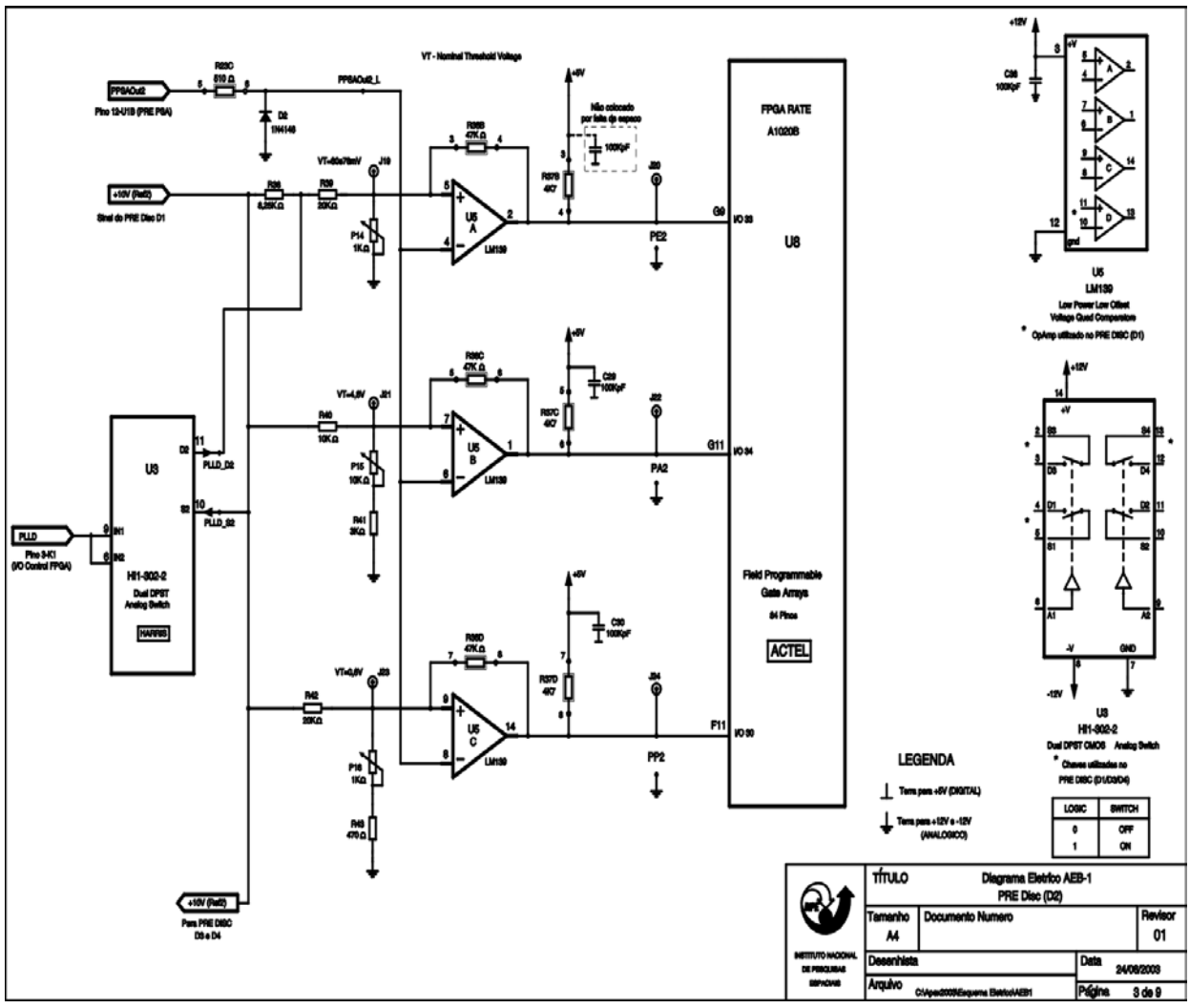


Figura 04 – Discriminator Module - PRE Disc D3

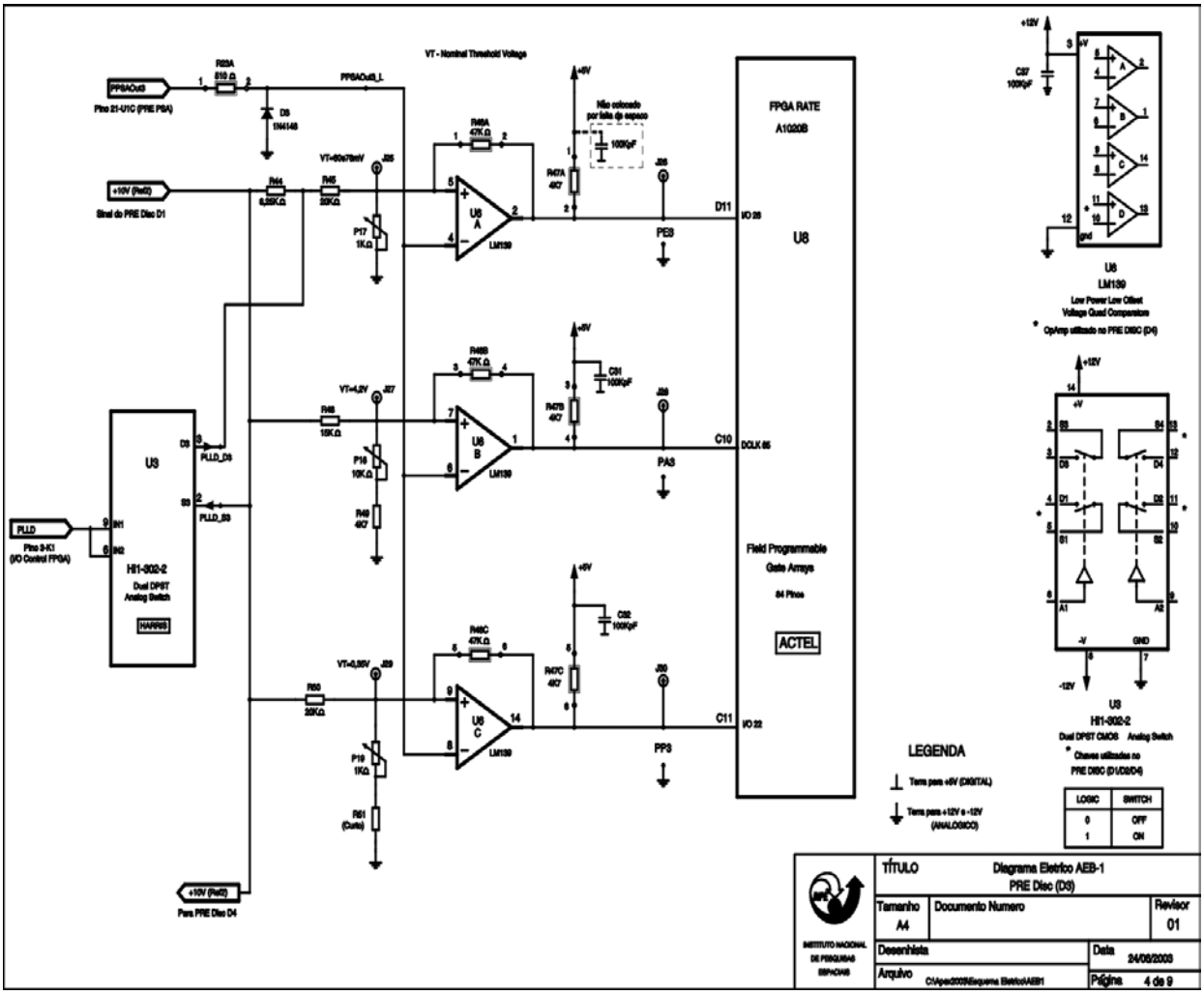


Figura 05 – Discriminator Module - Pre Disc D4

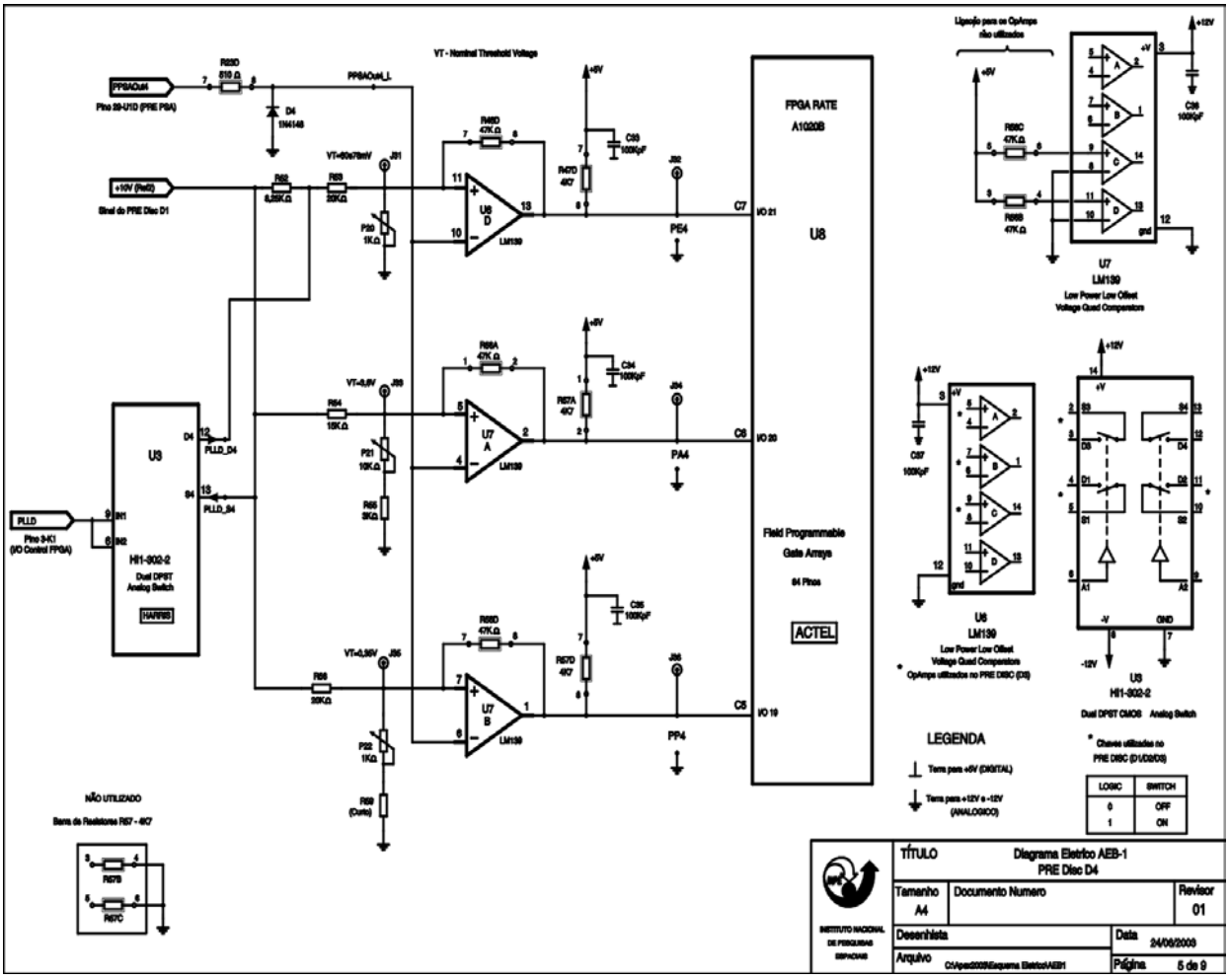


Figura 06 – I/O Control FPGA & Command Relays

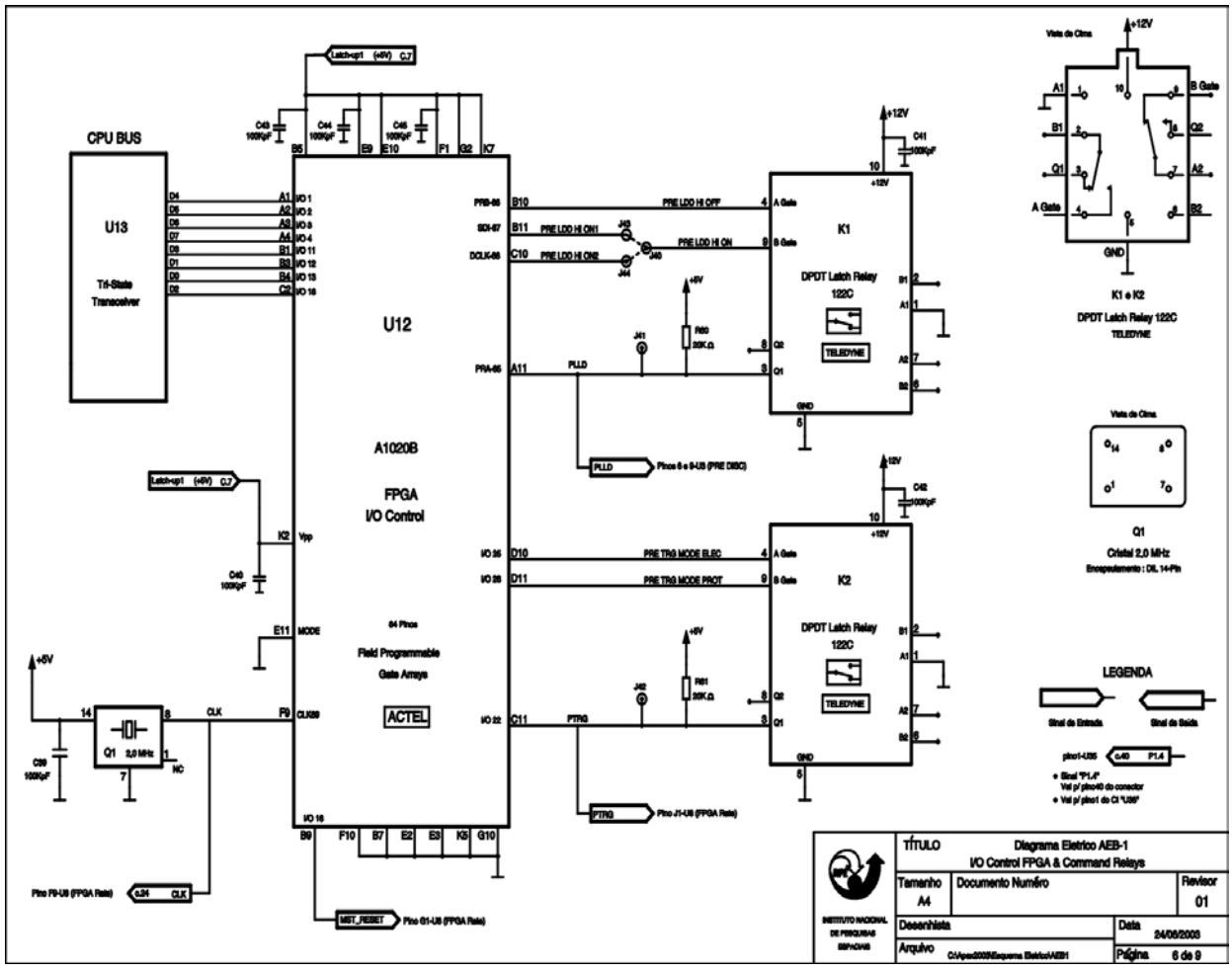


Figura 07 – Rate FPGA e Memórias

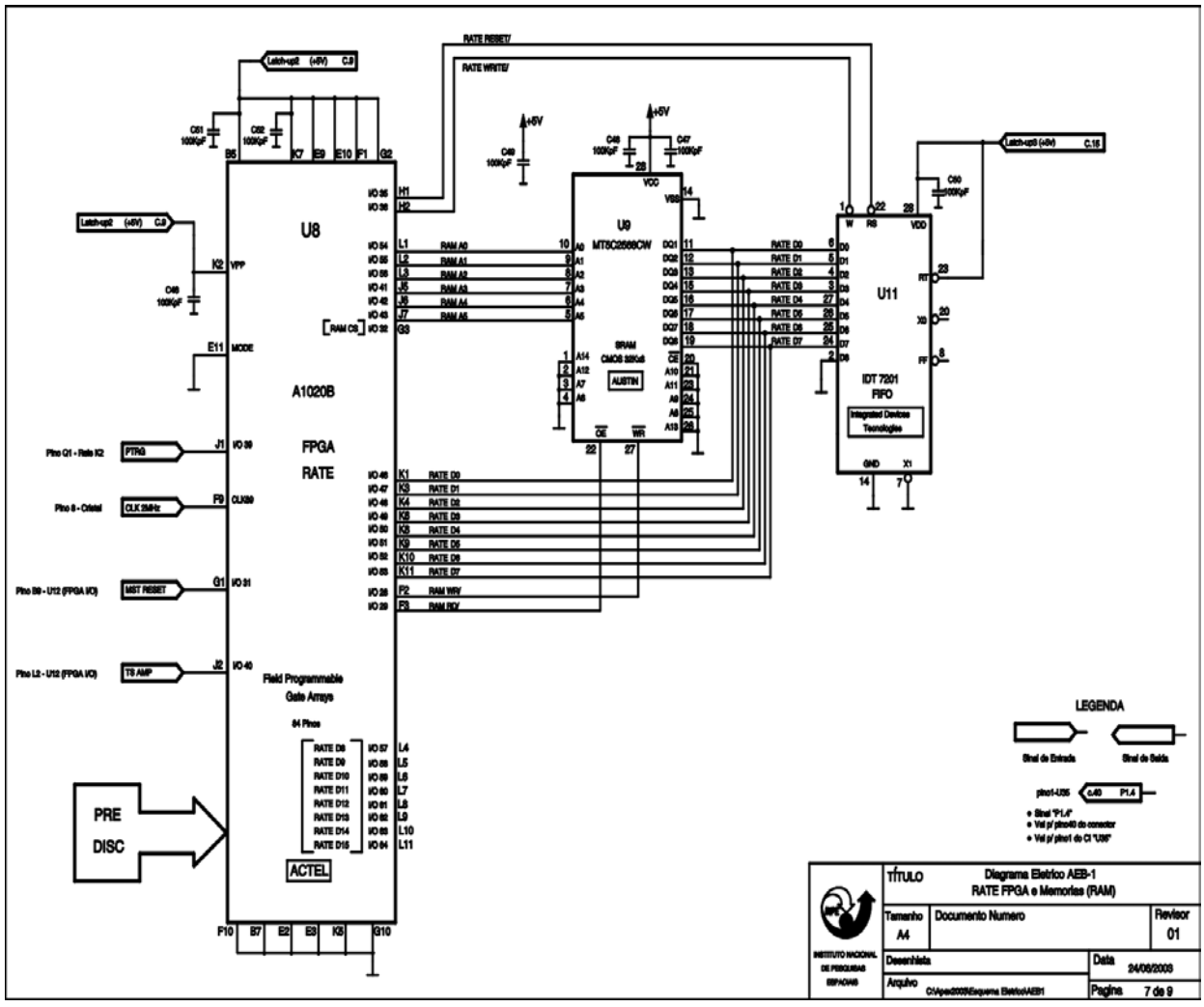


Figura 08 – CPU BUS & FIFO

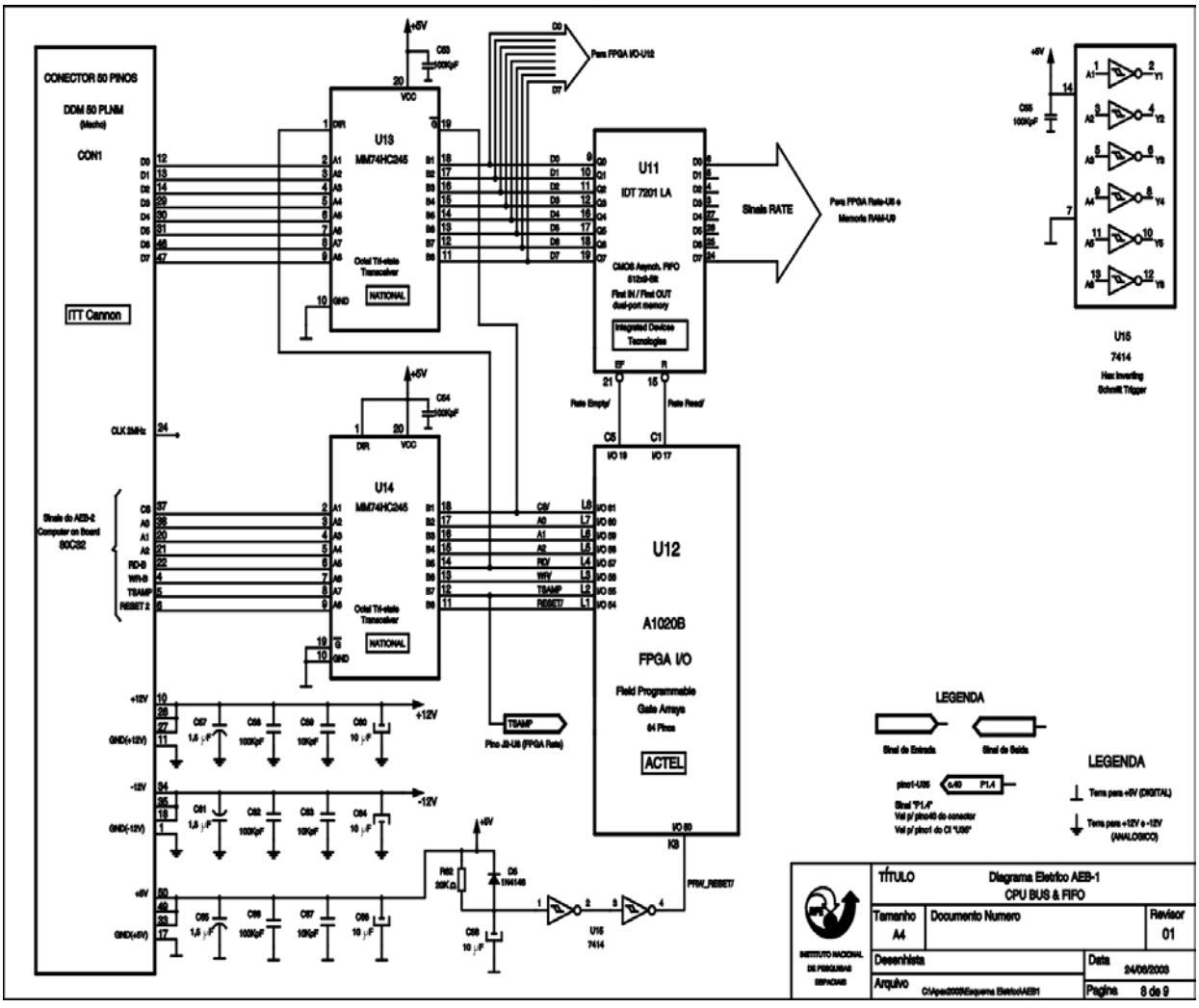
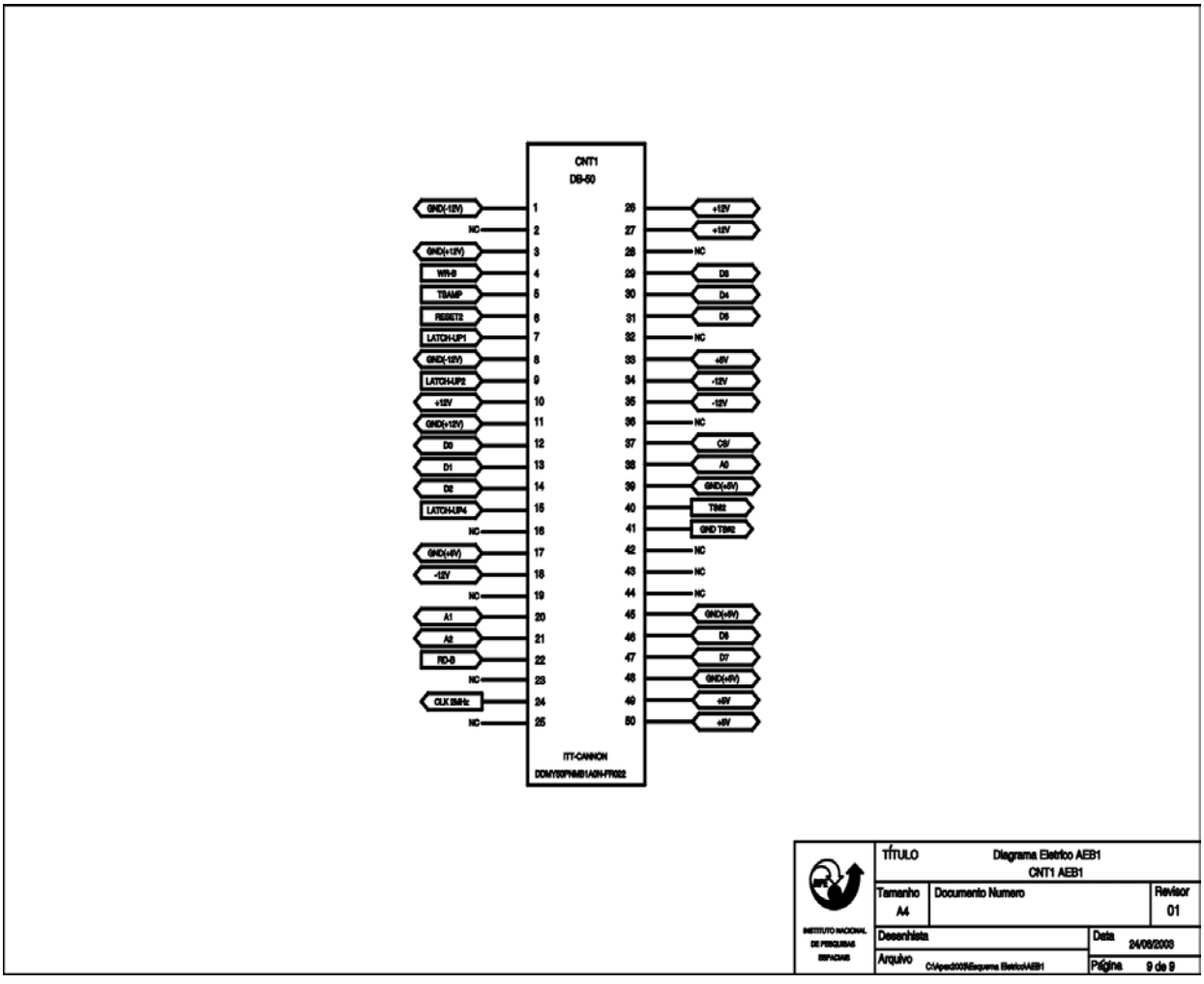


Figura 09 – Conector AEB1



APÊNDICE B – LISTAGEM DOS PROGRAMAS VHDL DESENVOLVIDOS

```
-- IOInterface.vhd – Definição da Entity e da Architecture do FPGA I/O

library ieee;
use ieee.std_logic_1164.all;

entity IOInterface is

    port (
          -- data and address buses
          D: inout std_logic_vector(7 downto 0); -- data bus
          A: in std_logic_vector(2 downto 0);    -- address

          -- CPU control signals
          RESET: in std_logic;                   -- reset (inv)
          TSAMP: in std_logic;                   -- TSAMP (edge rising)
          WR: in std_logic;                      -- write (inv)
          RD: in std_logic;                      -- read (inv)
          CS: in std_logic;                      -- chip select (inv)


          -- latches control signals
          PRE_LDD_HI_OFF: out std_logic;
          PRE_LDD_HI_ON1: out std_logic;
          PRE_LDD_HI_ON2: out std_logic;
          PLLD: in std_logic;                     -- PLLD status

          PRE_TRG_MODE_ELECT: out std_logic;
          PRE_TRG_MODE_PROT: out std_logic;
          PTRG: in std_logic;                     -- PTRG status

          -- FIFO control signals
          RATE_EMPTY: in std_logic;                -- (inv)
          RATE_READ: out std_logic;                 -- (inv)

          CLK: in std_logic                       -- clock, 2MHz
          );
end IOInterface;

library ieee;
use ieee.std_logic_1164.all;


architecture v0 of IOInterface is

signal RATE_READY: std_logic;

begin
     process (WR,RD, RATE_EMPTY)
     begin
         RATE_READY <= '1';

         if (WR = '0' and CS='0') then -- write begin
            case A is
                when "000" =>
```

```
                    PRE_LDD_HI_OFF <= not D(0);
                    PRE_LDD_HI_ON1 <= D(0);
                    PRE_LDD_HI_ON2 <= D(0);
                when "001" =>
                    PRE_TRG_MODE_ELECT <= D(0);
                    PRE_TRG_MODE_PROT  <= not D(0);

                when others => null;
            end case;
        end if;


        if (RD = '0' and CS='0') then
            case A is
                when "000" => RATE_READ <= '0';
                when "001" => D <= RATE_READY &"00000" & PTRG & PLLD;

                when others => null;
             end case;
        end if;

        if (RD = '1') then
           D<="ZZZZZZZZ";
        end if;

        if (RATE_EMPTY = '0') then
           RATE_READ <='1';
        end if;

    end process;
end v0;
```

```
-- Diagrama de temporização para testes
-- Gerado automaticamente pelo ambiente integrado de desenvolvimento
-- Generated by WaveFormer Lite Version 9.0u at 17:31:28 on 5/11/2005
-- Stimulator for stimulus

library ieee, std;
use ieee.std_logic_1164.all;
-- Libraries used by Model Under Test.
use IEEE.std_logic_1164.all;
-- End Libraries used by Model Under Test.

entity stimulus is
  port (
    D : inout std_logic_vector(7 downto 0) := "ZZZZZZZZ";
    A : out std_logic_vector(2 downto 0) := "ZZZ";
    RESET : out std_logic := 'Z';
    TSAMP : out std_logic := 'Z';
    WR : out std_logic := 'Z';
    RD : out std_logic := 'Z';
    CS : out std_logic := 'Z';
    PLLD : out std_logic := 'Z';
    PTRG : out std_logic := 'Z';
    RATE_EMPTY : out std_logic := 'Z';
    CLK : out std_logic := 'Z');

  signal D_driver : std_logic_vector(7 downto 0);
  signal A_driver : std_logic_vector(2 downto 0);
  signal RESET_driver : std_logic;
  signal TSAMP_driver : std_logic;
  signal WR_driver : std_logic;
  signal RD_driver : std_logic;
  signal CS_driver : std_logic;
  signal PLLD_driver : std_logic;
  signal PTRG_driver : std_logic;
  signal RATE_EMPTY_driver : std_logic;
  signal CLK_driver : std_logic;
end stimulus;

architecture STIMULATOR of stimulus is

  -- Control Signal Declarations
  type TStatus is
(TB_INIT,TB_ABORT,TB_ONCE,TB_LOOPING,TB_DONE,TB_TIMEOUT,TB_RESTART);
  signal tb_status : TStatus;
  signal tb_ParameterInitFlag : boolean := false;

  -- Assignments from drivers to output and inout ports.

begin

  D <= D_driver;
  A <= A_driver;
  RESET <= RESET_driver;
  TSAMP <= TSAMP_driver;
  WR <= WR_driver;
  RD <= RD_driver;
  CS <= CS_driver;
```

```
PLLD <= PLLD_driver;
PTRG <= PTRG_driver;
RATE_EMPTY <= RATE_EMPTY_driver;
CLK <= CLK_driver;

-- Status Control block.
process
  variable good : boolean;
begin
  wait until tb_ParameterInitFlag;
  tb_status <= TB_ONCE;
  wait for 394.0 ns;
  tb_status <= TB_DONE;
  wait;
end process;

-- Parm Assignment Block
AssignParms : process
begin
  tb_ParameterInitFlag <= true;
  wait;
end process;

-- Clocked Sequences

-- Sequence: Unclocked
Unclocked : process
begin
  D_driver <= "ZZZZZZZZ";
  A_driver <= "XXX";
  RESET_driver <= '1';
  TSAMP_driver <= '1';
  WR_driver <= '1';
  RD_driver <= '1';
  CS_driver <= '1';
  PLLD_driver <= '1';
  PTRG_driver <= '0';
  RATE_EMPTY_driver <= '1';
  CLK_driver <= 'Z';
  wait for 10.0 ns;
  PTRG_driver <= '1';
  wait for 3.0 ns;
  PLLD_driver <= '0';
  wait for 4.0 ns;
  PTRG_driver <= '0';
  wait for 2.0 ns;
  A_driver <= "001";
  wait for 6.0 ns;
  CS_driver <= '0';
  wait for 1.0 ns;
  PLLD_driver <= '1';
  PTRG_driver <= '1';
  wait for 3.0 ns;
  RD_driver <= '0';
  wait for 5.0 ns;
  PTRG_driver <= '0';
  wait for 2.0 ns;
```

```
PLLD_driver <= '0';
wait for 6.0 ns;
PTRG_driver <= '1';
wait for 7.0 ns;
PLLD_driver <= '1';
wait for 2.0 ns;
PTRG_driver <= '0';
wait for 1.0 ns;
RD_driver <= '1';
wait for 4.0 ns;
PLLD_driver <= '0';
wait for 1.0 ns;
CS_driver <= '1';
wait for 7.0 ns;
A_driver <= "XXX";
wait for 1.0 ns;
PLLD_driver <= '1';
wait for 1.0 ns;
PTRG_driver <= '1';
wait for 7.0 ns;
PLLD_driver <= '0';
wait for 1.0 ns;
PTRG_driver <= '0';
wait for 7.0 ns;
PTRG_driver <= '1';
wait for 1.0 ns;
PLLD_driver <= '1';
wait for 5.0 ns;
PTRG_driver <= '0';
wait for 5.0 ns;
A_driver <= "000";
PLLD_driver <= '0';
PTRG_driver <= '1';
wait for 7.0 ns;
CS_driver <= '0';
wait for 1.0 ns;
PLLD_driver <= '1';
wait for 2.0 ns;
PTRG_driver <= '0';
wait for 6.0 ns;
RD_driver <= '0';
wait for 1.0 ns;
PLLD_driver <= '0';
wait for 4.0 ns;
PTRG_driver <= '1';
wait for 3.0 ns;
PLLD_driver <= '1';
wait for 4.0 ns;
PTRG_driver <= '0';
wait for 2.0 ns;
RATE_EMPTY_driver <= '0';
wait for 4.0 ns;
PLLD_driver <= '0';
wait for 1.0 ns;
PTRG_driver <= '1';
wait for 5.0 ns;
PLLD_driver <= '1';
```

```
wait for 2.0 ns;
PTRG_driver <= '0';
wait for 5.0 ns;
RD_driver <= '1';
wait for 4.0 ns;
PTRG_driver <= '1';
wait for 1.0 ns;
CS_driver <= '1';
wait for 1.0 ns;
PLLD_driver <= '0';
wait for 3.0 ns;
PLLD_driver <= '1';
wait for 4.0 ns;
A_driver <= "XXX";
wait for 1.0 ns;
PTRG_driver <= '0';
wait for 2.0 ns;
PLLD_driver <= '0';
wait for 6.0 ns;
PTRG_driver <= '1';
wait for 1.0 ns;
PLLD_driver <= '1';
wait for 7.0 ns;
PTRG_driver <= '0';
wait for 1.0 ns;
PLLD_driver <= '0';
wait for 4.0 ns;
PTRG_driver <= '1';
wait for 3.0 ns;
PLLD_driver <= '1';
wait for 3.0 ns;
PTRG_driver <= '0';
wait for 54.0 ns;
D_driver <= x"01";
wait for 2.0 ns;
A_driver <= "000";
wait for 8.0 ns;
CS_driver <= '0';
wait for 3.0 ns;
WR_driver <= '0';
wait for 26.0 ns;
WR_driver <= '1';
wait for 6.0 ns;
CS_driver <= '1';
wait for 7.0 ns;
A_driver <= "XXX";
wait for 4.0 ns;
D_driver <= "ZZZZZZZZ";
wait for 25.0 ns;
D_driver <= x"01";
wait for 4.0 ns;
A_driver <= "001";
wait for 8.0 ns;
CS_driver <= '0';
wait for 5.0 ns;
WR_driver <= '0';
wait for 23.0 ns;
```

```
    WR_driver <= '1';
    wait for 3.0 ns;
    CS_driver <= '1';
    wait for 7.0 ns;
    A_driver <= "XXX";
    wait for 6.0 ns;
    D_driver <= "ZZZZZZZZ";
    wait for 23.0 ns;
    wait;
  end process;
end STIMULATOR;

-- Test Bench wrapper for stimulus and Model Under Test
library ieee, std;
use ieee.std_logic_1164.all;
-- Libraries used by Model Under Test.
use IEEE.std_logic_1164.all;
-- End Libraries used by Model Under Test.

entity testbench is
end testbench;
architecture tbGeneratedCode of testbench is
  signal D : std_logic_vector(7 downto 0);
  signal A : std_logic_vector(2 downto 0);
  signal RESET : std_logic;
  signal TSAMP : std_logic;
  signal WR : std_logic;
  signal RD : std_logic;
  signal CS : std_logic;
  signal PLLD : std_logic;
  signal PTRG : std_logic;
  signal RATE_EMPTY : std_logic;
  signal CLK : std_logic;
  signal PRE_LDD_HI_OFF : std_logic;
  signal PRE_LDD_HI_ON1 : std_logic;
  signal PRE_LDD_HI_ON2 : std_logic;
  signal PRE_TRG_MODE_ELECT : std_logic;
  signal PRE_TRG_MODE_PROT : std_logic;
  signal RATE_READ : std_logic;

  -- Stimulator instance

begin

  stimulus_0 : entity work.stimulus
    port map (D => D,
              A => A,
              RESET => RESET,
              TSAMP => TSAMP,
              WR => WR,
              RD => RD,
              CS => CS,
              PLLD => PLLD,
              PTRG => PTRG,
              RATE_EMPTY => RATE_EMPTY,
              CLK => CLK);
```

```
  -- Instantiation of Model Under Test.
  IOInterface_0 : entity work.IOInterface
    port map (D => D,
              A => A,
              RESET => RESET,
              TSAMP => TSAMP,
              WR => WR,
              RD => RD,
              CS => CS,
              PRE_LDD_HI_OFF => PRE_LDD_HI_OFF,
              PRE_LDD_HI_ON1 => PRE_LDD_HI_ON1,
              PRE_LDD_HI_ON2 => PRE_LDD_HI_ON2,
              PLLD => PLLD,
              PRE_TRG_MODE_ELECT => PRE_TRG_MODE_ELECT,
              PRE_TRG_MODE_PROT => PRE_TRG_MODE_PROT,
              PTRG => PTRG,
              RATE_EMPTY => RATE_EMPTY,
              RATE_READ => RATE_READ,
              CLK => CLK);
end tbGeneratedCode;
```

```
-- Descricao da arquitetura final pos-síntese
-- Gerado automaticamente pelo ambiente integrado de desenvolvimento
-- Version: 6.0 Production 6.0.0.133

library ieee;
use ieee.std_logic_1164.all;
library APA;

entity IOInterface is

    port(D : inout std_logic_vector(7 downto 0); A :
        in std_logic_vector(2 downto 0); RESET, TSAMP, WR, RD, CS :
        in std_logic; PRE_LDD_HI_OFF, PRE_LDD_HI_ON1,
        PRE_LDD_HI_ON2 : out std_logic; PLLD : in std_logic;
        PRE_TRG_MODE_ELECT, PRE_TRG_MODE_PROT : out std_logic;
        PTRG, RATE_EMPTY : in std_logic; RATE_READ : out
         std_logic; CLK : in std_logic);

end IOInterface;

architecture DEF_ARCH of IOInterface is

  component AND2
    port(A, B : in std_logic := 'U'; Y : out std_logic);
  end component;

  component OR2
    port(A, B : in std_logic := 'U'; Y : out std_logic);
  end component;

  component LD
    port(EN, D : in std_logic := 'U'; Q : out std_logic);
  end component;

  component IB33
    port(PAD : in std_logic := 'U'; Y : out std_logic);
  end component;

  component OB33PH
    port(PAD : out std_logic; A : in std_logic := 'U');
  end component;

  component NOR2
    port(A, B : in std_logic := 'U'; Y : out std_logic);
  end component;

  component NAND3FFT
    port(A, B, C : in std_logic := 'U'; Y : out std_logic);
  end component;

  component OTB33PH
    port(PAD : out std_logic; A, EN : in std_logic := 'U');
  end component;

  component INV
    port(A : in std_logic := 'U'; Y : out std_logic);
  end component;
```

```
  component LDB
    port(EN, D, CLR, SET : in std_logic := 'U'; Q : out std_logic);
  end component;

  component GND
    port(Y : out std_logic);
  end component;

  component LDS
    port(EN, D, SET : in std_logic := 'U'; Q : out std_logic);
  end component;

  component PWR
    port(Y : out std_logic);
  end component;

  component NOR3
    port(A, B, C : in std_logic := 'U'; Y : out std_logic);
  end component;

  component LDLC
    port(EN, D, CLR : in std_logic := 'U'; Q : out std_logic);
  end component;

  component IOB33PH
    port(PAD : inout std_logic := 'U'; A, EN : in
         std_logic := 'U'; Y : out std_logic);
  end component;

    signal N_13_i, un1_wr_3_0_a3_n, RATE_EMPTY_c_i_0,
        RATE_EMPTY_c, \D_in_i_0[0]\, \D_in[0]\, N_14, CS_c,
        \A_c_i_0_i[0]\, un1_wr_2_0_a3_n, WR_c, un1_wr_1_0_a3_n,
        N_15, un1_rd_0_a3_n, RD_c, rate_read_i_n, RATE_READ_c,
        \A_c[1]\, \A_c[2]\, PRE_LDD_HI_OFF_c, PRE_LDD_HI_ON1_c_c,
        PRE_TRG_MODE_ELECT_c, PRE_TRG_MODE_PROT_c, \d[0]_net_1\,
        PLLD_c, \d[1]_net_1\, PTRG_c, \d[7]_net_1\, \GND\, \VCC\
         : std_logic;

begin


    rate_read_i : AND2
      port map(A => N_15, B => RATE_READ_c, Y => rate_read_i_n);

    rate_read_i_a2 : OR2
      port map(A => \A_c_i_0_i[0]\, B => N_14, Y => N_15);

    \PRE_TRG_MODE_PROT\ : LD
      port map(EN => un1_wr_2_0_a3_n, D => \D_in_i_0[0]\, Q =>
        PRE_TRG_MODE_PROT_c);

    \PRE_LDD_HI_OFF\ : LD
      port map(EN => un1_wr_1_0_a3_n, D => \D_in_i_0[0]\, Q =>
        PRE_LDD_HI_OFF_c);

    PTRG_pad : IB33
```

```
  port map(PAD => PTRG, Y => PTRG_c);

un1_wr_2_0_a2 : OR2
  port map(A => \A_c[1]\, B => \A_c[2]\, Y => N_14);

PLLD_pad : IB33
  port map(PAD => PLLD, Y => PLLD_c);

RATE_READ_pad : OB33PH
  port map(PAD => RATE_READ, A => RATE_READ_c);

PRE_TRG_MODE_PROT_pad : OB33PH
  port map(PAD => PRE_TRG_MODE_PROT, A => PRE_TRG_MODE_PROT_c);

un1_wr_2_0_a3 : NOR2
  port map(A => un1_wr_3_0_a3_n, B => WR_c, Y =>
    un1_wr_2_0_a3_n);

RD_pad : IB33
  port map(PAD => RD, Y => RD_c);

un1_wr_3_0_a3 : NAND3FFT
  port map(A => N_14, B => CS_c, C => \A_c_i_0_i[0]\, Y =>
    un1_wr_3_0_a3_n);

\D_pad[1]\ : OTB33PH
  port map(PAD => D(1), A => \d[1]_net_1\, EN => \d[7]_net_1\);

\d_i[7]\ : INV
  port map(A => un1_wr_3_0_a3_n, Y => N_13_i);

\D_pad[6]\ : OTB33PH
  port map(PAD => D(6), A => \GND\, EN => \d[7]_net_1\);

WR_pad : IB33
  port map(PAD => WR, Y => WR_c);

\D_pad[7]\ : OTB33PH
  port map(PAD => D(7), A => \d[7]_net_1\, EN => \d[7]_net_1\);

PRE_LDD_HI_OFF_pad : OB33PH
  port map(PAD => PRE_LDD_HI_OFF, A => PRE_LDD_HI_OFF_c);

\D_pad[5]\ : OTB33PH
  port map(PAD => D(5), A => \GND\, EN => \d[7]_net_1\);

RATE_EMPTY_pad : IB33
  port map(PAD => RATE_EMPTY, Y => RATE_EMPTY_c);

\A_pad[0]\ : IB33
  port map(PAD => A(0), Y => \A_c_i_0_i[0]\);

\d[7]\ : LDB
  port map(EN => \GND\, D => \GND\, CLR => RD_c, SET =>
    N_13_i, Q => \d[7]_net_1\);

GND_i : GND
```

```
  port map(Y => \GND\);

PRE_LDD_HI_ON2_pad : OB33PH
  port map(PAD => PRE_LDD_HI_ON2, A => PRE_LDD_HI_ON1_c_c);

\D_pad[2]\ : OTB33PH
  port map(PAD => D(2), A => \GND\, EN => \d[7]_net_1\);

CS_pad : IB33
  port map(PAD => CS, Y => CS_c);

\pre_ldd_hi_on2\ : LD
  port map(EN => un1_wr_1_0_a3_n, D => \D_in[0]\, Q =>
    PRE_LDD_HI_ON1_c_c);

PRE_TRG_MODE_PROT_i : INV
  port map(A => \D_in[0]\, Y => \D_in_i_0[0]\);

\RATE_READ\ : LDS
  port map(EN => un1_rd_0_a3_n, D => rate_read_i_n, SET =>
    RATE_EMPTY_c_i_0, Q => RATE_READ_c);

PWR_i : PWR
  port map(Y => \VCC\);

\PRE_TRG_MODE_ELECT\ : LD
  port map(EN => un1_wr_2_0_a3_n, D => \D_in[0]\, Q =>
    PRE_TRG_MODE_ELECT_c);

un1_wr_1_0_a3 : NOR3
  port map(A => CS_c, B => WR_c, C => N_15, Y =>
    un1_wr_1_0_a3_n);

\d[1]\ : LDLC
  port map(EN => un1_wr_3_0_a3_n, D => PTRG_c, CLR => RD_c, Q
     => \d[1]_net_1\);

\A_pad[2]\ : IB33
  port map(PAD => A(2), Y => \A_c[2]\);

PRE_LDD_HI_ON1_pad : OB33PH
  port map(PAD => PRE_LDD_HI_ON1, A => PRE_LDD_HI_ON1_c_c);

\d[0]\ : LDLC
  port map(EN => un1_wr_3_0_a3_n, D => PLLD_c, CLR => RD_c, Q
     => \d[0]_net_1\);

RATE_EMPTY_c_i : INV
  port map(A => RATE_EMPTY_c, Y => RATE_EMPTY_c_i_0);

\D_pad[0]\ : IOB33PH
  port map(PAD => D(0), A => \d[0]_net_1\, EN => \d[7]_net_1\,
    Y => \D_in[0]\);

PRE_TRG_MODE_ELECT_pad : OB33PH
  port map(PAD => PRE_TRG_MODE_ELECT, A =>
    PRE_TRG_MODE_ELECT_c);
```

```
    un1_rd_0_a3 : NOR2
      port map(A => CS_c, B => RD_c, Y => un1_rd_0_a3_n);

    \A_pad[1]\ : IB33
      port map(PAD => A(1), Y => \A_c[1]\);

    \D_pad[4]\ : OTB33PH
      port map(PAD => D(4), A => \GND\, EN => \d[7]_net_1\);

    \D_pad[3]\ : OTB33PH
      port map(PAD => D(3), A => \GND\, EN => \d[7]_net_1\);


end DEF_ARCH;
```