Desenvolvimento de um Sistema Computacional baseado em Processador RISC em FPGA

Eduardo Furtado Sá Corrêa
Departamento de Ciência da Computação
Universidade de Brasília
Brasília, Brasil
eduardoxfurtado@gmail.com

Ricado Pezzoul Jacobi
Departamento de Ciência da Computação
Universidade de Brasília
Brasília, Brasil
rjacobi@cic.unb.br

***Abstract*—This paper presents the process involved in the development of a basic architecture using the Verilog HDL to synthesize the BA22-DE processor on a FPGA starting from a simulation ready code and the processor's documentation. With the use of a base simulation model to drive the synthesis development and a code profiling approach, it was possible to develop a system architecture to provide a minimalist environment for the processor to work. All designs have been synthesized and tested running a simple piece of software with FPGA's and the results insures the correctness of the design and sets the path for future development in order to run more complex software on the BA22-DE processor.**

I. INTRODUÇÃO

Os processadores da família BA2x são fáceis de programar e tecnicamente capazes de competir com processadores 32-bit tradicionais. Possuí um ISA inovador, que conta com código extremamente denso que reduz o uso de memória e cache e ultimamente reduz o consumo de energia, e de acordo com [1], o BA22 apresenta performance melhor do que outras ISAs como a dos processadores MIPS e ARM.

Simulação e síntese são processos complementares durante o desenvolvimento de um sistema integrado, entretanto ambos modelos podem produzir resultados muito diferentes. A síntese gera um circuito, através da tradução de HDL a uma *netlist* composta por portas lógicas, enquanto a simulação simula o comportamento do circuito. A descrição de um modelo para simulação tende a ser mais simples por não levar em consideração tantos elementos físicos como na síntese, além de possuir mais recursos da linguagem que facilitam a descrição do modelo que por sua vez não são "sintetizáveis" e, diferente da síntese, possui uma semântica baseada em execução sequencial. Porém, a síntese é objetivo final de qualquer modelo, e requer código extremamente correto, o que faz com que a codificação e teste sejam tarefas bastante demoradas e que requerem tempo. Projetar para síntese partindo de uma simulação que descreve como se deve comportar o sistema possibilita o desenvolvimento rápido de implementações de arquiteturas de hardware. Junto da documentação de como funciona o processador, essa proposta praticamente elimina a fonte de erros do projeto.

Outra metodologia que acelera o tempo de projeto é a realização de um *profiling,* uma forma de análise dinâmica de programas para otimizar o desenvolvimento em que se mede diversos aspectos da execução como memória usada, complexidade ou, neste caso, quais componentes do hardware estavam sendo utilizados por um algoritmo de testes a ser executado pelo processador BA22 e selecionam-se apenas as partes essenciais do hardware para o funcionamento do sistema, com a finalidade de que apenas estas sejam descritas para a arquitetura. Os módulos restantes ficam desligados, o que economiza espaço no chip e diminui o consumo de energia. Isto é, se gasta tempo de projeto de hardware apenas com aqueles módulos que devem estar presentes para a execução de um software específico no processador.

Nesta pesquisa, buscou-se aproveitar um modelo pronto, feito para simulação, como base para implementar uma arquitetura sintetizável do processador BA22-DE em uma FPGA e um teste com um simples software.

Este esforço visa testar a performance de execução do algoritmo de decodificação de áudio AAC, desenvolvido pela equipe da UnB, no processador BA22 como uma solução embarcada.

Atualmente, a equipe da UnB possui um decodificador AAC que utiliza a abordagem de coprojeto de hardware e software, onde a parte de processamento crítica está implementada em módulos de hardware dedicados e o restante funciona em software no processador Nios II da Altera. No futuro, espera-se utilizar o processador BA22 com o barramento AMBA para substituir o processador Nios II e o barramento Avalon.

Este artigo é organizado da seguinte forma: a Seção 2 apresenta a metodologia utilizada; a Seção 3 explica o desenvolvimento do módulo inicializador de memória; a Seção 4 explica o desenvolvimento do módulo gerador de *clocks*; a Seção 5 explica o desenvolvimento do módulo monitor; a Seção 6 apresenta os resultados obtidos; e, por fim, a Seção 7 apresenta as conclusões do trabalho.

II. METODOLOGIA

O primeiro passo para o desenvolvimento deste projeto foi realizar a simulação do funcionamento do processador para observar o comportamento dos sinais gerados pelo sistema, a leitura da documentação e o estudo do código não sintetizável já existente. Foi possível realizar um *profiling* de que módulos estavam sendo ativados com a execução de um algorítimo simples, através do estudo dos sinais gerados durante a simulação do sistema, assim identificando os módulos indispensáveis ao projeto e que deveriam ser implementados para síntese. A Figura 1 mostra o diagrama de blocos do ambiente de simulação.

Identificou-se que o Gerador de *Clock*, o inicializador de memória, o monitor e o barramento AMBA eram módulos que deveriam ser totalmente reescritos para a síntese. Utilizando a linguagem de descrição de hardware Verilog, primeiramente foi reescrito o módulo inicializador de memória, que tem a função de copiar desde uma ROM, feita utilizando VHDL, contendo o código binário executável. Em seguida foi implementado o módulo gerador de *Clock* e o resto do projeto foi adaptado para as novas entradas. Após, optou-se por não utilizar um barramento completo, e por meio de simplificações em como seria o acesso à memória, foi escrita uma pequena porção de código que atua como um barramento, dando permissão total ao processador, único elemento que acessa a memória. Finalmente o módulo monitor foi adaptado para as novas entradas e saídas. Por fim, essa arquitetura foi prototipada e testada em um FPGA.

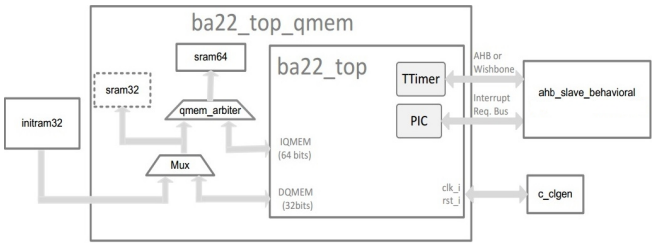

Figura 1. Diagrama de Bloco do ambiente de simulação, base para a arquitetura da síntese

III. DESENVOLVIMENTO DO MÓDULO INITRAM

O Inicializador de memória basicamente copia de uma memória a outra, neste caso da ROM para a RAM. O módulo faz isso de 4 em 4 endereços de memória, e ao terminar, é responsável por ligar o processador, que aguarda o fim do carregamento para começar a trabalhar. Tomou-se o cuidado de gravar na posição X enquanto se estava lendo da posição X+1. Trata-se de uma RAM de 16kbs. A figura 2 mostra o código da máquina de estados deste módulo.

```
always @(posedge clk)
begin
    if (rst) begin
        addr_rom <= 0;
        addr_ram <= 0;
        byte_en <= 4'b1111; //estamos copiando de 4 em 4,
        state <= s0;        //então usa-se as 4 partes do
        load_done <= 0;     //endereçamento da ram
    end
    else begin
        case (state)
            s0: begin
                addr_ram <= 0;
                addr_ram[13:0] <= {addr_rom, 2'b00}; //concatena
                addr_rom <= addr_rom + 1;   //com o shift para
                state <= s1;                //passar de 4 em 4.
            end
            s1: begin
                addr_ram <= 0;
                addr_ram[13:0] <= {addr_rom, 2'b00};
                addr_rom <= addr_rom + 1;
                if ( addr_rom >= (2**12)-1 ) begin
                    state <= s_done;
                end
            end
            s_done: begin
                load_done <= 1;
                state <= s_done;
                byte_en <= 0;
                addr_ram <= 0;
            end
            default:
                state <= s0;
        endcase
```

Figura 2. Código da máquina de estados do inicializador de memória

IV. DESENVOLVIMENTO DO MÓDULO CLGEN

O módulo do gerenciamento de energia do BA22-DE usa um *clock* diferente do restante do projeto. Além disso, a execução do software na placa é muito rápido e por isso foi feito um módulo auxiliar ao gerador de *clock* para atrasar o mesmo, possibilitando acompanhar a execução do programa de teste passo a passo. A Figura 3 mostra o código do módulo clgen. A figura 4 mostra o código do atrasador de *clock*.

```
always @(posedge clk)
begin
    if ( rst == 0 ) begin
        contador_load_done  = 6'b000000;
        contador            = 2'b00;
        pm_clk_o            = 1;
        rst_o               = 0;
        `ifdef BA22_PM_IMPLEMENTED
            pm_stall_o      = 1'b0;
        `endif
    end
    else begin
        if(contador == 2'b01) begin
            pm_clk_o <= ~pm_clk_o;
            contador = 2'b00;
        end
        else begin
            contador <= contador +1;
        end
        if ( load_done == 1 ) begin //delay=60 para rst ativo
            contador_load_done = contador_load_done + 1;
            if ( contador_load_done == 59) begin
                rst_o = 1;
            end
        end
    end
end //end always
```

Figura 3. Código do gerador de *clock*

```
always @(posedge CLOCK_50)
begin
    if(KEY[0] == 0) begin
        CLOCK_LENTO     = 1'b0;
        clock_slower    = 18'b000000000000000000;
    end
    else begin
        if ( SW == 0 ) begin
            CLOCK_LENTO <= ~CLOCK_LENTO;
        end
        if ( SW == 1 ) begin
            if( clock_slower == 18'b111101000010010000 ) begin
                CLOCK_LENTO <= ~CLOCK_LENTO;
                clock_slower = 0;
            end
                clock_slower = clock_slower + 1;
        end
    end
end
```

Figura 4. Código do atrasador de *clock*

V. DESENVOLVIMENTO DO MÓDULO MONITOR

O módulo monitor foi projetado com o objetivo de mostrar na placa por meio de LEDs o estado em que se encontra o sistema, inicializando a memória ou após a inicialização, o que estava sendo executado no processador e aviso de fim da execução.

VI. RESULTADOS

O sistema foi prototipado na placa de desenvolvimento DE2-115, que possui uma FPGA Altera Cyclone IV E, modelo EP4CE115F29C, apresentada na figura 5.

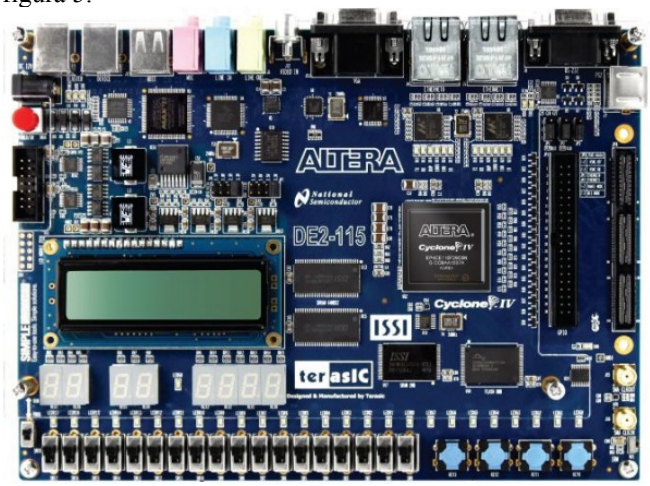

Figura 5. Placa de prototipação DE2-115

O *clock* da placa teve sua frequência dividida por 500000, fazendo com que o tempo de execução total do teste fosse de cerca de dois minutos. Dessa forma foi possível acompanhar o andamento do sistema, desde que recebido o sinal de *reset*, carregamento da RAM e execução do código C apresentado abaixo.

```
const char *s = "Hello, World!\n";
while (*s) {
        REG32( CAST_CHAR ) = *s;
        ++;
}
int i = 0;
for (i=0; i <= 0x99; i++) {
         REG32( CAST_NUM ) = i;
}
while (i--) {
         REG32( CAST_NUM ) = i;
}
s = hello;
while (*s) {
        REG32( CAST_CHAR ) = *s;
        s++;
}
REG32( CAST_ENDSIM ) = 1;
while (1);
```

O resultado da síntese mostrou que o sistema desenvolvido utiliza 17.413 elementos lógicos, 3758 registros, 18 pinos, 1.361.984 bits de memória interna, 8 multiplicadores de 9 bits e nenhum PLL.

VII. CONCLUSÃO

Neste trabalho desenvolveu-se uma arquitetura em hardware para a síntese do processador de embarcados BA22-DE. Optou-se por uma metodologia com utilização de um projeto para simulação para realizar um *profiling* do que seria necessário para o funcionamento do processador. O sistema desenvolvido foi prototipado em uma FPGA. Os resultados mostram que o objetivo foi alcançado.

Cabe mencionar que este é apenas o primeiro passo para viabilizar a decodificação de áudio AAC em tempo real no processador BA22-DE e após isso, substituir o processador NIOS II pelo BA22 para utilizar o processador BA22 como uma solução de coprojeto com o decodificador de áudio AAC.

VIII. REFERÊNCIAS

A estrutura discutida acima foi baseada nas fontes abaixo, acrescida da opinião dos autores e colaboradores.

- CAST; BA22 32-bit RISC Processor Integration Manual*, 2012. BA22-INM-4.0P06-103*, IP Product Version 4.0p06.
- CAST; BA22 32-bit RISC Processor Hardware Specification*, 2012. BA22-DES-4.0P06-104*, IP Product Version 4.0p06.
- Thomas & Moorby's; The Verilog Hardware Description Languague; Fifth Edition; ISBN 1-4020-7089-6; 2002 Kluwer Academic Publishers.
- Stuart Sutherland and Don Mills; Verilog and System Verilog Gotchas – 101 Common Coding Errors and How to Avoid Them; ISBN 978-0-387-71714-2; 2007 Springer.

[1] CAST, Inc. **BA2 Instruction Set Architecture. Disponível** em:<www.cast-inc.com/ip-cores/processors32bit/index.html> Acesso em: 12 Jul. 2013.