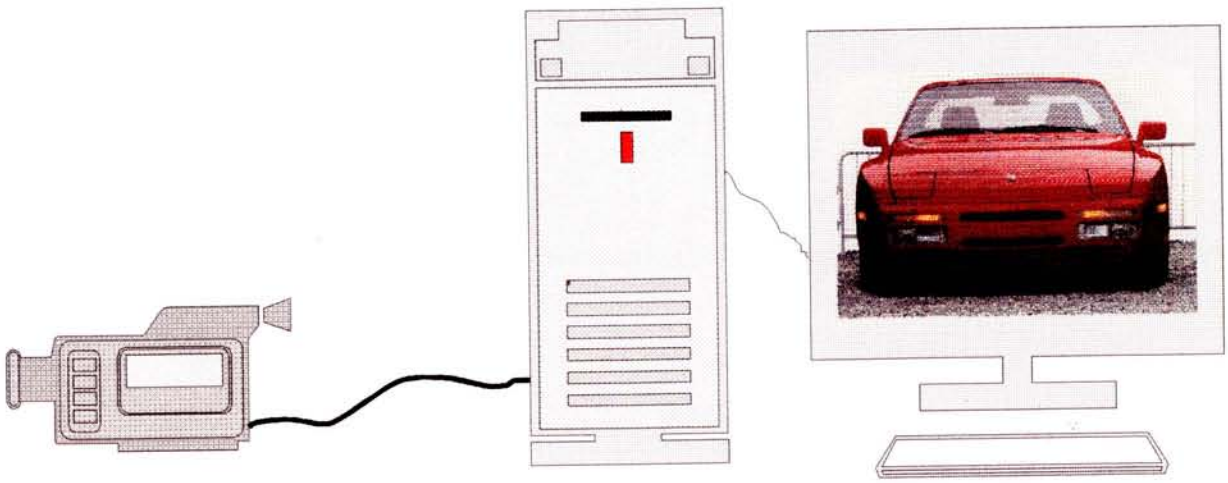
ALGORITMOS PARA CODIFICAÇÃO DIGITAL DE TELEVISÃO



Rogério Paulo Ferreira de Sousa Nunes

INESC Instituto de Engenharia
de Sistemas e Computadores

PARECER

Confirmo o empenho revelado no estágio pelo *Rogério Nunes*, bem como a qualidade do trabalho realizado e intitulado "*Algoritmos para Codificação Digital de Televisão*".

Considero assim o candidato merecedor da bolsa **PRODEP**.

Porto, 23 de Julho de 1993.

MÁRIO JORGE LEITÃO
Director

Largo Mompiller 22 • Apartado 4433 • 4007 Porto Codex • Portugal • Tel. 351 2 321006 • Fax. 351 2 318692 • Tlx. 23023 INESC P
Pessoa Colectiva n.º 501107169 • Associação Privada sem fins lucrativos, declarada de Utilidade Pública • C.R.C. de Lisboa n.º 72

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO
Departamento de Engenharia Electrotécnica e de Computadores
Rua dos Bragas, 4099 Porto Codex, PORTUGAL
Telef. 351-2-317105/107/412/457 · Telex 27323 FEUP P · Telefax 351-2-319280

PARECER

Estágio PRODEP de
Rogério Paulo Ferreira de Sousa Nunes

O Rogério Nunes desenvolveu um trabalho intitulado "Algoritmos para Codificação Digital de Televisão" que se desenrolou associado ao estágio do Nuno Arantes visou o desenvolvimento de rotinas para a interface na placa de aquisição de vídeo. O trabalho desenrolou-se de forma muito boa e é por isso completamente merecedor da bolsa PRODEP.

Porto, 23 de Julho de 1993

Artur Pimenta Alves
(Prof. Associado, do DEEC)

Introdução

O porquê da realização deste trabalho/projecto ?

Este trabalho/projecto teve como finalidade contribuir para criar um sistema de codificação e aquisição Digital de Vídeo. Insere-se assim na concepção e desenvolvimento de um sistema de aquisição de vídeo vocacionado para a implementação de codecs de vídeo de baixa cadência, através de ferramentas de software. Este trabalho foca essencialmente os algoritmos obtidos, e portanto necessários, para a realização do objectivo atrás descrito.

Surge-nos assim a questão - **porque forma foi implementada a solução pretendida ?**

Num passado não muito distante o desenho de circuitos ou sistemas digitais era concebido unicamente através de componentes discretos, de que faziam parte, contadores, inversores, ORs, ANDs,... Enquadravam-se assim no desenho de Sistemas Digitais por componentes discretos. Hoje em dia com o advento dos computadores, cada vez mais poderosos, começaram a surgir as primeiras ferramentas que permitem fazer a implementação deste tipo de circuito, mas utilizando o computador como auxiliar do desenvolvimento do nosso objectivo. Surgiram assim os programas de CAD para electrónica(ex. "ORCAD"), que além de conterem os elementos que nós precisamos, permite-nos desenhar um ou outro componente que nas livrarias de componentes não existe. Permite-nos ainda, através do desenho, dispormos os componentes da melhor forma e ligá-los da forma que pensarmos mais adequada. Por fim permite-nos ainda fazer o teste funcional ao circuito por nós concebido, detectando possíveis erros. Ultrapassou-se assim o problema de implementar os circuitos utilizando, em parte a base tentativa/erro. As coisas estão mais organizadas e permitem um trabalho de melhor qualidade.

A ferramenta de trabalho por nós utilizada foi a chamada EPLD(Erasable Program Logic Device), que é um circuito que permite incluir circuitos combinacionais e sequenciais. Outro factor importante relacionado com estes componentes é o facto de serem extremamente rápidos, podendo-se obter atrasos na ordem dos 12 ns, importantes quando se trata com um período de amostragem, na resolução máxima da placa de vídeo, de 72ns por pixel. Estas EPLDs são

programadas por um programa que corre em PC, sobre o WINDOWS, chamado "ALTERA - MAX-PLUS II".

Neste programa encontra-se uma ferramenta de desenho que permite desenharmos o nosso circuito digital em termos gráficos, através dos componentes existentes em livrarias, bem como conceber as suas ligações. Mas se fosse só esta a única possibilidade, não traria grande vantagem perante aquilo que existe no mercado, e que falei atrás(CAD). Assim este programa MAX-PLUS II permite que através de uma linguagem de programação que ele possui, se possa conceber os nossos próprios componentes, concebendo-os à medida das nossas necessidades. Não havendo desperdício de recursos. A grande novidade surge assim na concepção dos blocos através de linhas de código de programação(ver à frente rotinas de implementação dos blocos), e não através da combinação de circuitos(quer combinacionais quer sequenciais).

Depois de criados os nossos próprios elementos, e compilados, são criados os símbolos gráficos associados a cada um desses elementos. Convém salientar que estes símbolos podem ser conjugados, no mesmo desenho do circuito com componentes existentes em livrarias. Daqui resulta uma extrema flexibilidade.

Se a tudo o que dissemos acrescentarmos a possibilidade de podermos simular o nosso circuito em tempo real(se for possível), bastando para isso gerar quer por texto quer através de uma interacção gráfica os sinais de entrada, verificamos que estamos perante um programa vanguardista, que serve os nossos intentos, e que nos permite poupar imenso tempo em termos de implementação. O único senão da simulação, já bastante falado, é o facto de por vezes para se simular todas as situações de interesse, o tempo de simulação ter que ser demasiado grande implicando horas e horas de espera.

Além dos módulos de edição/compilação dos nossos elementos e da simulação, ainda existem outros módulos dentro do MAX-PUS II. Vou-me referir a um outro de especial interesse, o analisador de tempos/atrasos("TIMING ANALYSER"), que dada uma qualquer entrada permite saber qual o atraso que vai ser estabelecido entre qualquer uma das saídas. Esta ferramenta para frequências muito elevadas é importantíssima.

Quais os sinais existentes e o seu significado, para uma melhor compreensão do trabalho?

Este ponto visa essencialmente prover o leitor deste trabalho, de algum conhecimento, embora superficial, sobre os sinais que irão ser incluídos nos blocos dos elementos por mim definidos no capítulo da implementação. Neste ponto só serão

focados os sinais de entrada, pois os sinais de saída dos elementos, serão descritos na análise individual de cada bloco.

Assim temos diversos sinais de entrada, nos nossos blocos, que são provenientes, e em seguida extraídos por circuitos analógicos e digitais, do sinal de sincronismo composto da câmara de vídeo.

Temos o sinal de Sincronismo Horizontal que permite sincronizar o canhão do televisor com o início de uma nova linha de informação. No nosso caso vai sincronizar o nosso sistema. Quanto ao sinal que surge no final do Sincronismo Horizontal, o Back-Porch, faz parte do sinal de blanking horizontal, correspondendo á parte final deste. Este sinal de blanking horizontal tem como função sincronizar o início da imagem e também fazer o retrocesso do canhão para se percorrer mais uma linha. Assim no final do sinal de Back-Porch inicia-se uma nova linha de informação válida.

No que concerne agora ao sinal Sincronismo Vertical, ele tem como papel sincronizar o nosso canhão com um novo campo que se vai iniciar. Cada imagem é composta por dois campos. Um par e um ímpar. Na nossa representação também temos um sinal que indica qual o campo que está a ser tratado.

Por fim o nosso relógio do sistema(clock) está sincronizado pelo bordo descendente do sinal de Sincronismo Horizontal(activo baixo).

Também enquadrado neste tema dos sinais, temos as resoluções adoptadas para o nosso sistema de aquisição. Assim foram escolhidas três tipos de resoluções que passo a descrever: **Resolução Alta** - 576 x 720(linhas x colunas) a que corresponde um período de clock de 72 ns ; **Resolução Média** - 288 x 360 com um período de amostragem de 144 ns ; e por fim **Resoluçaõ Baixa** - 144 x 180 a que corresponde um período de amostragem de 288 ns.

Implementação

Algumas considerações sobre a implementação.

Nas rotinas que são apresentada neste capítulo, referentes à programação dos diversos blocos ou elementos, que irão constituir o nosso sistema de aquisição, sempre que o caracter / aparecer antes de um sinal de entrada ou de saída, quero-me referir a um sinal activo baixo. Ou seja esse sinal está activo se a sua tensão for de 0 Volt.

Existe ainda outra consideração, que é o facto de se no final da designação de um sinal aparecer o caracter _ , querer indicar que esse sinal é o sinal de saída desse bloco. Foi necessária esta convenção pelo facto de haver sinais de entrada e de saída com o mesma designação. Em seguida seram feitos os comentários sobre a concepção de cada uma das rotinas.

ROTINA CONF_ADC

CONFIGURAÇÃO DO ADC

Este bloco foi projectado e criado com dois objectivos específicos.

O primeiro deles é o de gerar um **sinal de escrita(/wr_adc)** para **configuração** de um conversor analógico/digital(ADC) , ou seja, para **aceitação de uma palavra de comando** para se **seleccionar o modo de funcionamento**. Este sinal de escrita activo baixo, segundo as especificações técnicas do fabricante, necessita de estar no mínimo cerca de 50ns activo(neste caso, activo baixo refere-se ao nível lógico 0, que é o mesmo que os 0 Volt).

O segundo objectivo que orientou a concepção deste bloco foi a necessidade de criar um sinal de reset(inicialização) do sistema(constituído por todos os blocos) e que vai derivar deste bloco. Este sinal de **/reset**, que designaremos por **reset-saída**, será activado(sinal activo ao nível baixo - 0 Volt) por um outro sinal - **reset**, que da mesma forma designaremos por **reset-entrada** (sinal activo ao nível alto - 5Volt). Este sinal de **reset** poderá ser actuado por exemplo pelo utilizador ao premir um simples botão. A sua função(**/reset**) no sistema, é a de permitir inicializar FLIP-FLOPs e todos os elementos constituintes dos blocos, que por sua vez fazem parte deste projecto. Convém referir que sempre que o utilizador actue nesse botão, irá ser

também gerado o sinal de configuração do periférico(ADC), durante um ciclo de clock(mínimo de 72ns). Por ultimo, antes de entrar na parte mais específica do que foi feito, convém referir que o sistema ao arrancar se encontra na situação de reset permanente, deixando de o estar quando é actuado o sinal de **reset** pela primeira vez. Depois de iniciado o funcionamento, sempre que o sinal de **reset** for activo será gerado o sinal de **/reset** bem como o sinal **/wr_adc**.

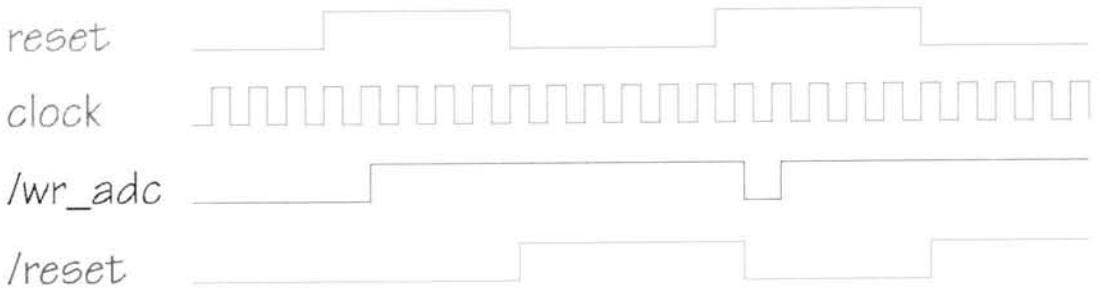
Quanto à maneira de implementar o que foi dito atrás, desenvolveu-se a rotina cujo nome é **CONF_ADC**, e que se apresenta a seguir. Nesta temos dois sinais de entrada, **reset** e **clock**, e dois sinais de saída, **/reset** e **/wr_adc**. Neste como em qualquer outro bloco será sempre utilizado um FLIP-FLOP para atribuir o nível de tensão à saída, portanto fixo, à excepção das transições do ciclo de clock, por forma a evitar que o ruído ou alterações na alimentação possam alterar o nível lógico da saída.

No arranque do sistema ambas as saídas estão ao nível lógico 0, ou seja estão activadas. Assim está-se a gerar o sinal para configurar o periférico, bem como permanentemente a fazer reset a todos os restantes blocos do sistema. Quando surgir o sinal de **reset** o sistema está no estado S1 e continua a activar os dois sinais, garantindo-se que o sinal de /wr_adc está activo mais um ciclo. Em seguida o sistema é passado para o estado S2. Aqui o sinal de /wr_adc é colocado a 1(desactivado), e o sinal de **/reset** contínua activo até o sinal de entrada **reset** deixar de ser actuado. Nesta ultima situação ambos os sinais são colocados a 1(desactivados) e o sistema regressa ao estado de partida S1. Aqui como não temos o sinal reset activo, não se procede a alterações em ambos os sinais de saída, e a máquina de estados permanece em S1, até se voltar a fazer **reset** ao sistema. O papel da máquina de estados é permitir actuar as saídas nas transições do sinal reset. Quando nos referimos às saídas estamos na realidade a referimo-nos aos FLIP-FLOPs. Mas como o valor destes é atribuído às saídas, a menos de um ciclo de clock, optamos, por ser mais simples, por nos referirmos antes às saídas. Este princípio será o adoptado, a menos que se verifique que não é o adequado.

O que atrás foi dito pode ser visto na figura seguinte. Nesta figura bem como nas próximas fica estabelecido que os sinais acima do clock são os de entrada, ou contribuem como sinais de entrada, ao passo que abaixo do sinal de clock temos os sinais de saída ou que contribuem como tal.

```
SUBDESIGN CONF_ADC
(
        reset                   : INPUT;        % entrada de sinal de um botão de reset %
        clock                   : INPUT;        % relógio de trabalho deste bloco %

        /reset                  : OUTPUT;       % saída de sinal de reset para os restantes blocos %
        /wr_adc                 : OUTPUT;       % saída de sinal para configuração do ADC %
)

VARIABLE

        /wr_AD                  : DFF;          % definiçao de FLIP-FLOPs %
        /reset_flip             : DFF;

        maq_adc                 : MACHINE OF BITS (q1)      % definição de máquina de estados %
                                  WITH STATES ( s1 = B"0",
                                                s2 = B"1"
                                                );

BEGIN

        /wr_AD.clk = clock;                     % relógio de trabalho dos elementos %
        /reset_flip.clk = clock;                % atrás definidos %
        maq_adc.clk = clock;

        /reset = /reset_flip.q;                 % sinal de saída é associado à saída de FLIP-FLOP %
        /wr_adc = /wr_AD.q;                     % mesmo que o anterior %
```

```
    CASE maq_adc IS
        WHEN S1 =>                                  % máquina de estados em S1 %
            IF reset THEN                           % no caso de reset estar activo, então %
                /wr_AD.d = GND;                     % activa sinais de que os FLIP-FLOPs %
                /reset_flip.d = GND;                % são portadores %
                maq_adc = S2;                       % faz máquina estados saltar para o estado S2 %
            ELSE                                    % caso de reset não estar activo %
                /wr_AD.d = /wr_AD.q;                % valor dos FLIP-FLOPs é mantido %
                /reset_flip.d = /reset_flip.q;      % e espera neste estado S1, até que %
                maq_adc = S1;                       % o sinal de reset fique activo %
            END IF;
        WHEN S2 =>                                  % máquinas de estados em S2 %
            IF reset THEN                           % no caso de reset activo %
                /wr_AD.d = VCC;                     % o ciclo de escrita no ADC terminou(1 ciclo) %
                /reset_flip.d = GND;                % mas o reset para os outros blocos mantém-se %
                maq_adc = S2;                       % espera em S2 até que sinal de reset termine %
            ELSE                                    % no caso de reset ficar desactivado %
                /wr_AD.d = VCC;                     % ambos os sinais de que os FLIP-FLOPs %
                /reset_flip.d = VCC;                % são portadores são desactivados %
                maq_adc = S1;                       % e salta máquina de estados para S1, até que %
            END IF;                                 % volte a surgir o sinal de reset %
    END CASE;
END;
```



ROTINA CONT_BAC

GERA UM SINAL EQUIVALENTE À SOMA DO SINCRONISMO HORI- ZONTAL COM O BACK-PORCH, INDICADOR DE INÍCIO DE UMA LINHA

A criação deste bloco surgiu como necessidade de gerar um sinal que indique em que ponto se inicia a amostragem de uma linha. Como se sabe, o primeiro pixel aparece logo a seguir ao sinal de **BACK-PORCH(/back_)**. Este sinal é por sua vez assíncrono com o clock, pois no nosso caso o clock é sintetizado numa PLL, síncrono com o bordo descendente do sinal de **Sincronismo Horizontal(/hsync)**. Logo dado ser um sinal assíncrono, e como se sabe a utilização de FLIP-FLOPs atrasar um ciclo, o nosso sinal de saída que indicaria o começo de uma nova linha, não podia ser gerado através do sinal de BACK-PORCH. Outra razão que levou a não utilizar o sinal de BACK-PORCH como indicador de início de linha foi o de implicar a utilização de mais um pino de entrada, quando se poderia gerar um sinal equivalente, síncrono, a partir do aparecimento do sinal de Sinc. Horizontal. Este sinal equivalente é igual à soma dos sinais de Sinc. Horizontal com o BACK-PORCH. No nosso caso o que nos interessa é substituir este valor de tempo por ciclos de clock, de forma a o sistema estar todo síncrono com o clock. Este valor de ciclos de clock, depende da resolução em causa(**resol[1..0] = B"00" - resolução alta ; resol[1..0] = B"01" - resolução média ; resol[1..0] = B"10" - resolução baixa**), colocado num contador.

Esta implementação possui a versatilidade de como se tem ainda outros blocos seguintes com FLIP-FLOPs, se poder actuar no valor com que se carrega os contadores, por forma a se simular este sinal equivalente como um sinal sem atrasos nenhuns, ao ser visto pelo final do sistema, ou então ser fácil corrigir estes atrasos.

No que se refere á forma de implementar o que atrás foi dito, pode-se ver a rotina **CONT_BAC.** Como sinais de entrada temos a resolução(**resol[1..0]**), o sinal de Sincronismo Horizontal(**/hsync)** que é activo baixo, e o sinal de reset(**/reset**) para

```
SUBDESIGN CONT_BAC
(
        clock                   : INPUT;                % entradas do bloco %
        /hsync                  : INPUT;
        resol[1..0]             : INPUT;
        /reset                  : INPUT;

        /back                   : OUTPUT;               % única saída do bloco %
)
VARIABLE

        count[7..0]             : DFF;                  % definição de elementos %
        back_flip               : DFF;                  % constantes só neste bloco, %
                                                        % e que por isso não podem ser %
        maq_back                : MACHINE OF BITS (q1)  % acedidos do exterior %
                                  WITH STATES (  S1 = B"0",
                                                 S2 = B"1"
                                                 );

BEGIN

        back_flip.prn = /reset;                         % FLIP-FLOP inicializado a 1 %

        maq_back.clk = clock;                           % clock de trabalho destes elementos %
        count[].clk = clock;
        back_flip.clk = clock;
```

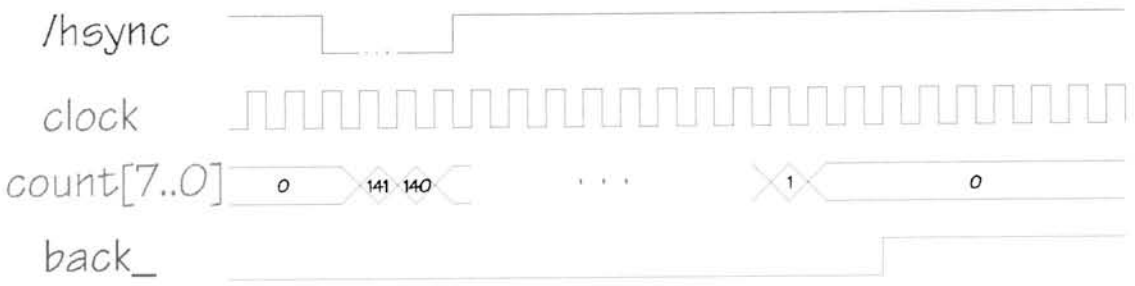
```
/back = back_flip.q;                          % à saída está associada um FLIP-FLOP %

CASE maq_back IS
        WHEN S1 =>                            % máquina de estados em S1, por defeito %
                IF !/hsync THEN               % sinal de Sinc. Horizontal activo %
                        IF resol[] == B"00" THEN      % resolução Alta %
                                count[].d = B"10001110";
                        ELSIF resol[] == B"01" THEN   % resolução Média %
                                count[].d = B"01000110";
                        ELSE
                                count[].d = B"00100010";   % resolução Baixa %
                        END IF;
                        back_flip.d = GND;    % activo sinal equivalente de que o FLIP-FLOP é portador %
                        maq_back = S2;        % máquina de estados salta para S2 %
                ELSE                          % no caso de esperar que Sinc. Horizontal fique activo %
                        back_flip.d = VCC;    % sinal contido no FLIP-FLOP é desactivado %
                        maq_back = S1;        % máquina permanece no estado S1 %
                END IF;
        WHEN S2 =>                            % máquina de estados está em S2 %
                IF count[].q == B"00000000" THEN   % se já passou o tempo de duração do sinal equivalente %
                        back_flip.d = VCC;    % desactiva sinal de que o FLIP-FLOP é portador %
                        maq_back = S1;        %máquina salta para S1 onde espera que Sinc.Hor fique activo%
                ELSE                          % no caso de o sinal equivalente nao ter chegado ao fim %
```

```
                    count[].d = count[].q - 1;          % decrementa o contador que contém a duração %
                    back_flip.d = GND;                  % sinal de saída contínua activado %
                    maq_back = S2;                      % espera neste estado que o contador fique igual a zero %
                END IF;
    END CASE;
END;
```

permitir que o sistema ao se premir o botão de reset arranque com o sinal equivalente inactivo, ou seja a 1. Como sinal de saída temos **/back_**. No que se refere agora à implementação, quando surge **/hsync** procede-se ao carregamento no contador do valor do sinal equivalente, também activo baixo, que é dado pela soma dos sinais de Sinc. Horizontal com o BACK-PORCH em termos de ciclos de clock. Atendendo à norma do CCITT para sistemas de televisão PAL, a soma dos tempos em que estes dois sinais estão activos é de 10.5μs. Logo na resolução máxima e que temos um período de clock de 72ns, o sinal equivalente corresponde a 145.8333 ciclos de clock, na resolução média temos um período de 144ns e 72.9166 ciclos de clock e por fim na resolução baixa, em que o período é de 288ns temos 36.4583 ciclos de clock. Como nos interessa por questões de segurança não perdermos o primeiro pixel, então arredondamos o valor de ciclos de clock para o inteiro imediatamente abaixo. Ou seja apanhamos na pior das hipóteses um pixel na região de BACK-PORCH. Depois do valor carregado no contador, no **ciclo seguinte** é actuada a saída (.q) do FLIP-FLOP, ou seja colocado nos 0 VOLT, correspondente ao sinal de saída(ao sinal de saída está associado um FLIP-FLOP por questões já tratadas). Em seguida a máquina passa para o estado S2. Aqui o contador é decrementado até se atingir o valor 0. Ou seja até se atingir 0 no contador, o número de ciclos contados foi de mais um que o desejado. Isto pois os FLIP-FLOPs do contador demoram um ciclo a colocar na saída(.q) o que é colocado na entrada(.d). Depois de se verificar que é zero o FLIP-FLOP associado à saída é actuado, só também colocando na saída(.q) o que se colocou na entrada(.d) no ciclo seguinte. Ou seja ao valor calculado atrás é necessário retirar 3 ciclos de clock. Depois de analisado o conjunto dos blocos chegou-se à conclusão que era ainda preciso retirar aqui mais um devido ao atraso introduzido por um FLIP-FLOP(**linha_flip**) no bloco **MAQ_LIN**(ver rotina à frente). Passamos assim a ter os seguintes valores para carregar o contador: Resol. Alta - 141 = Binario"10001101" ; Resol. Média - 68 = B"01000100" ; Resol. Baixa - 32 = B"00100000".

Depois de atingir o valor 0 e actuar a saída para o valor lógico 1(não actuado), a máquina de estados passa para S1, em que o sistema espera que o sinal de Sinc. Horizontal volte a estar activo(nível 0), para se voltar a repetir o ciclo atrás descrito.

O esquema dos sinais seguintes permite dar uma visão dos objectivos deste bloco. Neste bloco foi assumido que estavamos na resolução máxima de trabalho.



ROTINA CONT_H

GERA SINAL INDICADOR DE FINAL DE UMA LINHA

A criação deste bloco surge no intuito de gerar um sinal que é activo ao nível baixo, determinando se a aquisição de uma linha é para se processar ou se já terminou. Dito de uma outra forma, indica se os pixeis de uma linha já foram todos adquiridos.

Como já foi descrito no sistema de televisão por nós adoptado, o início de uma linha com informação surge no final do sinal de BACK-PORCH. Depois consoante a resolução por nós pretendida, são amostrados os pixeis, atingindo-se o final de uma linha em termos do número de pixeis amostrados até então.

O que este bloco faz é contar o número de pixeis amostrados à frequência de clock, e activar um sinal **fim_linha**, que é activo alto, indicando que os pixeis pretendidos de uma linha já foram todos amostrados. Portanto dá em seguida a informação que o fim da linha foi atingido. Quando voltar a surgir de novo o sinal de BACK-PORCH, o seu final, a linha volta a poder ser amostrada.

No que se refere à implementação, temos como sinais de entrada, o sinal **/load** que permite carregar o número de pixeis a serem amostrados consoante a resolução introduzida através do sinal de entrada **dq[1..0]**. Esta resolução pode ser de três níveis. Temos também o sinal activo baixo **/reset** que permite ao o sistema arrancar, ou seja quando é feito o reset geral ao sistema, a saída deste bloco estar activa. Indicando assim que o fim da linha foi atingido, esperando pelo próximo **BACK-PORCH equivalente**.

Quanto à estruturação do código, tudo o que vai ser dito pode ser constatado na rotina que se apresenta a seguir. Quando o sinal de **/load**, que é o nosso sinal atrás descrito BACK-PORCH equivalente, ficar activo é carregado no contador o número de pixeis a amostrar numa linha e o sinal indicador de fim de linha é colocado a 0,

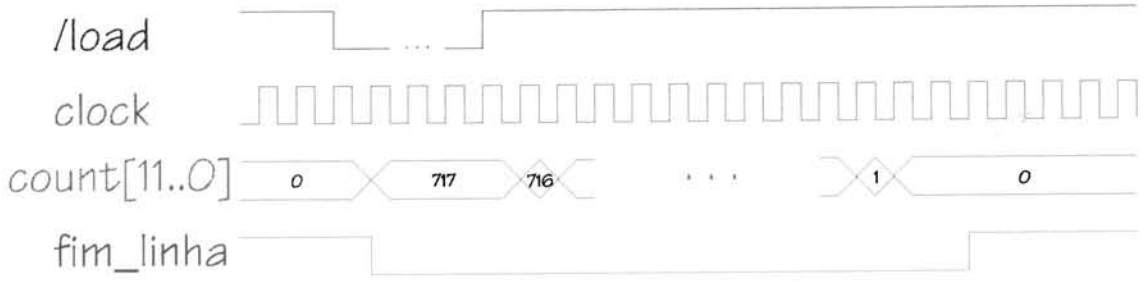
```
SUBDESIGN CONT_H
(
        clock                   : INPUT;        % entrada dos diversos sinais %
        /load                   : INPUT;
        dq[1..0]                : INPUT;
        /reset                  : INPUT;

        fim_linha               : OUTPUT;       % saída do uníco sinal %
)
VARIABLE

        count[9..0]             : DFF;          % definição dos elementos internos %
        fim_flip                : DFF;

BEGIN

        fim_flip.prn = /reset;                  % este FLIP-FLOP é inicializado a 1 %

        count[].clk = clock;                    % clock de trabalho dos elementos %
        fim_flip.clk = clock;

        fim_linha = fim_flip.q;                 % sinal de saída associado a um FLIP-FLOP %

        IF !/load THEN                          % no caso de surgir ordem para carregar valor nos contadores %
                IF dq[] == B"00" THEN           % valor para a resolução Alta %
                        count[].d = B"1011001101";
```

```
                fim_flip.d = GND;                    % sinal de fim de uma linha de aquisição é desactivado %
            ELSIF dq[] == B"01" THEN                 % valor para a resolução Média %
                count[].d = B"0101100101";
                fim_flip.d = GND;                    % sinal é desactivado %
            ELSE                                     % valor para a resolução Baixa %
                count[].d = B"0010110001";
                fim_flip.d = GND;                    % sinal é desactivado %
            END IF;
        ELSE                                         % senão é para carregar valor, e então %
            count[].d = count[].q -1;                % decrementa contador e %
            fim_flip.d = fim_flip.q;                 % mantém valor lógico do sinal de saída %
        END IF;

        IF count[].q == 0 THEN                       % se contador chegou a zero, então %
            fim_flip.d = VCC;                        % sinal de fim de aquisição desta linha é activado %
        END IF;
END;
```

indicando que a linha está pronta para ser amostrada. Convém referir que seria bom retirar um ciclo ao sinal BACK-PORCH equivalente por forma a este sinal poder ser actuado na altura certa e não com um ciclo de atraso(este procedimento já foi realizado e explicado na rotina Cont_Bac). Nos outros ciclos de clock em que **/load** não está activo o contador é decrementado até que se atinja o valor 0. Aí como já se adquiriram todos os pixeis duma linha, o sinal de indicação de fim de linha é activado. O processo repete-se em todas as linhas.

Quanto ao valor a carregar no contador, este depende da resolução em causa. Assim temos que na resolução Alta queremos amostrar 720 pixeis, na resolução Média 360 pixeis e na Baixa 180 pixeis. Como a comparação é feita com o valor 0 temos que retirar 1 ao contador, e também como o sinal de saída está associado a um FLIP-FLOP temos que retirar outro ciclo pelo atraso daí resultante. Finalmente ainda se retira um ciclo pelo atraso do FLIP-FLOP quando surge o sinal **/load**, sinal esse que advém do bloco CONT_BAC. Convém referir que este sinal surge um ciclo em atraso relativamente ao que seria de esperar, pelo facto de o sinal de **/load** do bloco CONT_BAC surgir um ciclo mais cedo, como já foi referido. Refira-se que este sinal tem interesse em estar atrasado pelo facto de, tal como no bloco anterior, no bloco **MAQ_LIN** um FLIP-FLOP(**linha_flip**) introduzir um ciclo de atraso indesejável. Assim os valores a introduzir no contador são: Resol. Alta 717 - B"1011001101" ; Resol. Média - 357 - B"0101100101" ; Resol. Baixa - 177 - 0010110001".

Mais uma vez se apresenta a figura seguinte por forma a uma melhor compreensão do que até aqui foi dito. A resolução adoptada neste exemplo foi a mais elevada.



ROTINA RES_VER

GERA SINAL INDICADOR DE QUAIS AS LINHAS QUE DEVEM SER ADQUIRIDAS

A finalidade deste bloco é a de ao se passar de uma resolução mais elevada para outra, como já foi referido a frequência de amostragem vê-se reduzida para metade ou para um quarto. Mas não nos podemos esquecer que também o número de linhas tem que ser reduzido na mesma proporção. Pois nos ao reduzirmos a frequência estamos a reduzir somente o número de colunas.

Assim na resolução mais elevada todas as linhas de qualquer um dos campos, par ou ímpar, são para adquirir, ao passo que na resolução média são para adquirir todas as linhas dentro de um mesmo campo. Na resolução mais baixa são para adquirir dentro de um mesmo campo linha sim - linha não. É o que se faz neste bloco. Atribui ao sinal de saída **capt_linha** o valor do FLIP-FLOP saida_flip indicando assim se a linha presente, consoante a resolução, se é para adquirir ou não.

A forma prática de implementar o que foi atrás referido está expresso na rotina a seguir apresentada. As entradas deste bloco referem-se ao sinal indicador de início de uma nova linha **/back_porch**, à resolução em questão(já foi abordado atrás) **resol[1..0]** e ao sinal **/par_impar** que se refere ao campo da imagem(0 - campo par ; 1 - campo ímpar).

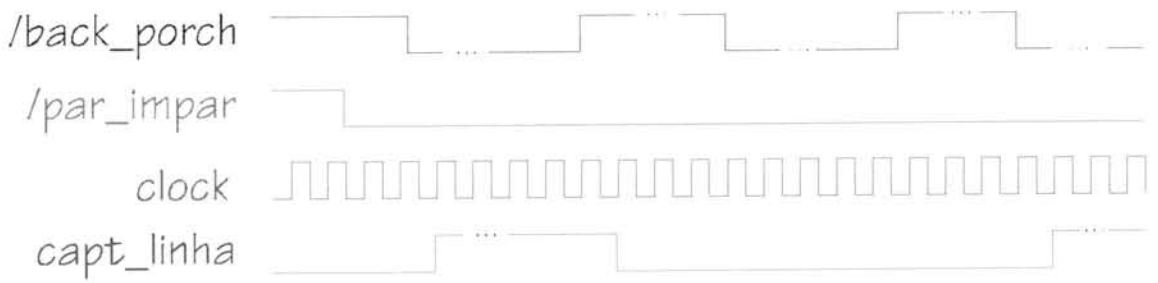
Na resolução Elevada todas as linhas são para adquirir. Daí se a resol[1..0] for igual a B"00" a saída está sempre activa(capt_linha). Se a resolução for a Média então nós adoptamos que são para adquirir todas as linhas dentro do campo par(/par_impar = GND). A situação mais complicada surge da resolução Baixa. As linhas a adquirir nesta resolução são as do campo par, também, embora linha sim - linha não. O procedimento adoptado foi o de se verificar se no fim de cada linha(início do **BACK-PORCH equivalente**), se essa linha foi adquirida ou descartada. Se foi descartada(**saida_flip** = GND) então a próxima é adquirida(**saida_flip** = VCC), ao passo que se foi adquirida(**saida_flip** = VCC) então a próxima é descartada(**saida_flip** = GND). Esta verificação é feita através do teste do valor que apresenta o FLIP-FLOP de saída - **saida_flip**(no fim de uma linha - **BACK-PORCH equivalente activo**). Em qualquer destas situações a máquina de estados é passada para S2. Ai í o valor do FLIP-FLOP de saída é mantido constante e espera-se que o valor do sinal de **/back_porch** deixe de estar activo. Quando deixar de o estar então a máquina passa para o estado S1, em que mantém os valores até que surja novamente o sinal de **/back_porch** do bloco Cont_Bac. O ciclo repete-se continuamente. Também aqui neste bloco era de toda a conveniência que o sinal de /back_porch surgi-se um ciclo mais cedo por forma a evitar o atraso do FLIP-FLOP de saída, que demora a actuar um ciclo de clock.

```
SUBDESIGN RES_VER
(
        /back_porch             : INPUT;                % definição dos sinais de entrada %
        resol[1..0]             : INPUT;
        /par_impar              : INPUT;
        clock                   : INPUT;

        capt_linha              : OUTPUT;               % definição do sinal de saída %
)
VARIABLE
        saida_flip              : DFF;                  % definição dos elementos internos ao bloco %

        maq_ver                 : MACHINE OF BITS (q1)
                                  WITH STATES (   S1 = B"0",
                                                  S2 = B"1"
                                                  );

BEGIN

        maq_ver.clk = clock;                            % clock de trabalho dos diversos elementos %
        saida_flip.clk = clock;

        capt_linha = saida_flip.q;                      % sinal de saída associado à saída(.q) de um FLIP-FLOP %

        IF ( resol[] == B"00" ) THEN                    % resolução Alta - todas as linhas são para adquirir %
                saida_flip.d = VCC;
```

```
END IF;
IF ( resol[] == B"01" & /par_impar == GND ) THEN          % resolução Média e campo par - todas as linhas são para %
        saida_flip.d= VCC;                                % adquirir %
ELSIF ( resol[] == B"11" ) THEN                           % se resolução é esta então houve erro e não adquire nada %
        saida_flip.d = GND;
END IF;
CASE maq_ver IS
        WHEN S1 =>
                IF ( resol[] == B"10" & /par_impar == GND ) THEN   % resolução Baixa e campo Par %
                        IF ( !/back_porch & !saida_flip.q ) THEN   % início de uma linha e a anterior não foi adquirida %
                                saida_flip.d = VCC ;               % então esta linha é para adquirir %
                                maq_ver = S2;                      % máquina de estados salta para S2 %
                        ELSIF ( !/back_porch & saida_flip.q ) THEN % se não, a linha anterior foi adquirida %
                                saida_flip.d = GND ;               % por isso a próxima não é para adquirir %
                                maq_ver = S2;                      % e salta a máquina de estados para S2 %
                        ELSE                                       % se não ainda, espera em S1 que ou comece %
                                saida_flip.d = saida_flip.q;       % uma linha ou então o campo seja o pretendido %
                                maq_ver = S1;                      % enquanto as saída permanecem inalteradas %
                        END IF;
                END IF;
        WHEN S2 =>                                                 % máquina de estados em S2 %
                IF ( resol[] == B"10" & /par_impar == GND ) THEN   % resolução Baixa e campo Par %
                        saida_flip.d = saida_flip.q;               % saida permanece inalterada %
                        IF ( /back_porch ) THEN                    % espera em S2 que o sinal de inicio de uma linha %
```

```
                        maq_ver = S1;                    % deixe de estar activo( sinal de /back_porch)%
                    ELSE
                        maq_ver = S2;
                    END IF;
                END IF;
        END CASE;
END;
```

A figura seguinte tenta ilustrar os objectivos pretendidos, através dos sinais gerados. Neste exemplo foi admitida a resolução baixa.



ROTINA MAQ_LIN

GERA SINAL INDICADOR DE SE A LINHA ESTÁ PRONTA PARA SER AMOSTRADA/ADQUIRIDA.

Este bloco tem como função activar um sinal - **para_linha**(activo alto) quando for detectado que já foram amostrados todos os pixeis referentes à resolução especificada para uma linha(sinal do **bloco CONT_H - fim_linha**), indicando que é para parar de adquirir pixeis ou de os amostrar. Quando surgir ou for detectado o sinal que indica que estamos perante uma nova linha(sinal do **bloco CONT_BAC - /back_porch equivalente**), o nosso sinal(**para_linha**) deve ser desactivado, permitindo que os pixeis possam ser outra vez adquiridos. Dito de uma outra forma permite gerar novo um sinal através da combinação dos sinais referidos atrás como **/back**_(Cont_Bac) e **fim_linha**(Cont_H). Essencialmente este sinal, bem como outros, vão permitir fazer o enable ou não do gerador de endereços das memórias, onde vão ser colocados os pixeis.

A implementação e o que dela vai ser dito pode ser acompanhado pela rotina MAQ_LIN.

Assim como entradas temos o sinal de **carry_out**(activo alto) que nos indica que o número de pontos dentro de determinada resolução já foram amostrados ou adquiridos. Outra entrada é o sinal de **/back_porch**(equivalente) que atrás já foi referido. O sinal de **/reset** permite ao inicializar o sistema, que o sinal de saída deste bloco esteja activo indicando que não é para adquirir nenhum pixel.

Quanto à forma de implementação temos quatro estados na nossa máquina de estados. Assim quando surge o sinal de paragem de amostragem(**carry_out** a 1) o sinal de parar a linha para amostragem - **para_linha** associado ao FLIP-FLOP

Implementação

Algumas considerações sobre a implementação.

Nas rotinas que são apresentada neste capítulo, referentes à programação dos diversos blocos ou elementos, que irão constituir o nosso sistema de aquisição, sempre que o caracter / aparecer antes de um sinal de entrada ou de saída, quero-me referir a um sinal activo baixo. Ou seja esse sinal está activo se a sua tensão for de 0 Volt.

Existe ainda outra consideração, que é o facto de se no final da designação de um sinal aparecer o caracter _ , querer indicar que esse sinal é o sinal de saída desse bloco. Foi necessária esta convenção pelo facto de haver sinais de entrada e de saída com o mesma designação. Em seguida seram feitos os comentários sobre a concepção de cada uma das rotinas.

ROTINA CONF_ADC

CONFIGURAÇÃO DO ADC

ROTINA CONT_BAC

GERA UM SINAL EQUIVALENTE À SOMA DO SINCRONISMO HORI- ZONTAL COM O BACK-PORCH, INDICADOR DE INÍCIO DE UMA LINHA

ROTINA CONT_H

GERA SINAL INDICADOR DE FINAL DE UMA LINHA

ROTINA RES_VER

GERA SINAL INDICADOR DE QUAIS AS LINHAS QUE DEVEM SER ADQUIRIDAS

ROTINA MAQ_LIN

GERA SINAL INDICADOR DE SE A LINHA ESTÁ PRONTA PARA SER AMOSTRADA/ADQUIRIDA.

ROTINA CONT_VSY

GERA SINAL EQUIVALENTE DO SINCRONISMO VERTICAL

ROTINA GER_END

GERA ENDEREÇOS PARA AS MEMÓRIAS

ROTINA IMA_DESC

GERA SINAL QUE DETERMINA QUAL A IMAGEM A ADQUIRIR

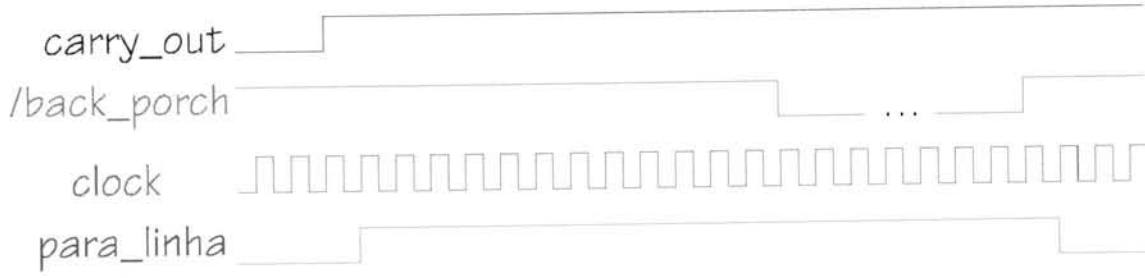
```
SUBDESIGN 'maq_lin'
(
        carry_out               : INPUT;                % definição dos sinais de entrada %
        /back_porch             : INPUT;
        clock                   : INPUT;
        /reset                  : INPUT;

        para_linha              : OUTPUT;               % definição do sinal de saída %
)

VARIABLE
        linha_flip              : DFF;                  % definição dos elementos internos %

        maq_linha               : MACHINE OF BITS (q1,q2)
                                  WITH STATES ( S1 = B"00",
                                                S2 = B"01",
                                                S3 = B"10",
                                                S4 = B"11"
                                                );
BEGIN

        linha_flip.prn = /reset;                        % FLIP-FLOP inicialiazdo a 1 %

        linha_flip.clk=clock;                           % clock de trabalho dos elementos %
        maq_linha.clk=clock;
```

```
para_linha=linha_flip.q;                    % sinal de saída associado a um FLIP-FLOP %

CASE maq_linha IS
        WHEN S1 =>                          % máquina de estados em S1 %
                IF ( carry_out == VCC) THEN % verificado o fim da amostragem de uma linha %
                        linha_flip.d = VCC; % sinal de parar a amostragem é activado %
                        maq_linha = S2;     % máquina de estado salta para S2 %
                ELSE                        % no caso de haver pixeis para adquirir, %
                        linha_flip.d = GND; % sinal de parar a aquisição continua desactivado %
                        maq_linha = S1;     % máquina espera em S1 pelo fim da amostragem %
                END IF;
        WHEN S2 =>                          % máquina de estados em S2 %
                IF ( carry_out == GND ) THEN % situação verificada quando surgir o sinal de Sinc. Horiz. %
                        linha_flip.d = VCC; % aquisição parada %
                        maq_linha = S3;     % máquina salta para o estado S3 %
                ELSE                        % se não surgiu ainda Sinc. Horiz. %
                        linha_flip.d = VCC; % mantém aquisição parada %
                        maq_linha = S2;     % e espera neste estado %
                END IF;
        WHEN S3 =>                          % máquina de estados em S3 %
                IF ( /back_porch == GND) THEN % quando surgir Sinc. Horiz. este sinal fica activo %
                        linha_flip.d = VCC; % e a aquisição ainda está parada %
                        maq_linha = S4;     % salta para estado S4 %
                ELSE
```

```
                    linha_flip.d = VCC;          % espera em S3 pela activação do sinal de Back-Porch %
                    maq_linha = S3;              % equivalente %
                END IF;
            WHEN S4 =>                           % máquina de estados em S4 %
                IF ( /back_porch == VCC) THEN    % se veio de S3 e sinal não activo, então %
                    linha_flip.d = GND;          % é para começar aquisição %
                    maq_linha = S1;              % e volta para estado S1 onde espera por sinal de fim de %
                ELSE                             % de aquisição, ou então espera em S4 pela activação %
                    linha_flip.d = VCC;          % do sinal de /back_porch %
                    maq_linha = S4;
                END IF;
        END CASE;
END;
```

linha_flip é colocado a 1, ou seja activado, saltando a máquina para o estado S2. Aí espera que o sinal de **carry_out** fique desactivado. Esta desactivação ocorre quando o sinal de **/back_porch** fica activo(0) - ver as rotinas atrás CONT_BAC e CONT_H. Neste estado a saída é colocada a VCC ou seja nível lógico 1. Quando o sinal **carry_out** fica então desactivado a nossa máquina salta para o estado S3. Garantiu-se assim que quando a máquina salta para S3 o sinal de **/back_porch** estava já activo, pois foi este sinal que desactivou o sinal de **carry_out**. Neste estado o valor da saída permanece activo, saltando para o estado S4, pois o sinal de **/back_porch** está activo. Em S4 espera-se que o sinal de **/back_porch** passe de activo para não activo, indicando uma nova linha, por forma a a saída ser desactivada. Ou seja começarem-se a adquirir pixeis novamente, agora referentes a uma nova linha. Depois de S4 a máquina salta para S1 em que espera novamente pela activação do sinal de **carry_out**. Dado este bloco e os sinais de entrada a ele associados (carry_out e /back_porch) há todo o interesse em que se tenha retirado um ciclo ao contador do bloco CONT_BAC devido aos atrasos verificados no FLIP-FLOP a actuar a saída.

Como tem sido habitual, em seguida apresenta-se o esquema com os sinais de entrada e de saída mais relevantes tratados neste bloco.

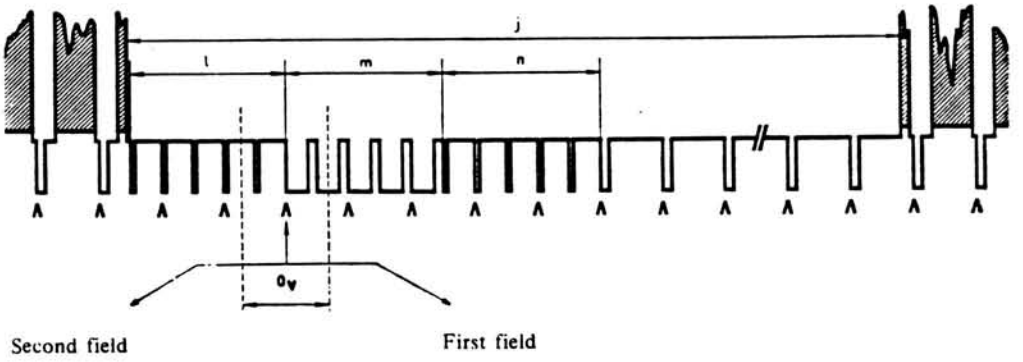


ROTINA CONT_VSY

GERA SINAL EQUIVALENTE DO SINCRONISMO VERTICAL

A criação deste bloco teve como função a síntese de um sinal, equivalente a um que já existe. É o caso do sinal de Sincronismo Vertical(activo baixo). A necessidade da criação de um sinal equivalente deste tipo, surge devido a ele não ser síncrono com o clock, quer o seu bordo descendente quer o bordo ascendente. Outro motivo, é a necessidade de nós não adquirirmos as linhas que se seguem a este pulso de Sincronismo Vertical - durante a segunda sequência de pulsos de equalização. Esta sequência surge logo a seguir ao final do pulso de sincronismo e dura 2,5 vezes o tempo de uma linha(64μs). O sinal de Sincronismo Vertical(duração **m**) bem como

estes pulsos(duração **n**) estão representados na figura abaixo. Seria para nós impossível e sem nenhum interesse adquirir estes pixeis, para além da forma dos sinais mudar muito(caso do Sincronismo Horizontal).



Criamos assim um sinal equivalente, que inclusivé nos é mais útil que o verdadeiro, pois além de conter a zona em que o Sincronismo Vertical fica activo, auxilia-nos a ultrapassar o problema atrás descrito.

A rotina de acompanhamento está a seguir descrita. Nesta rotina como sinais de entrada temos o referente ao Sincronismo Vertical - **/vsync** do sistema de televisão adoptado - PAL, a resolução em termos de nº colunas - **resol[1..0]** e o sinal de **/reset** para permitir ao inicializar o sistema que o sinal de saída deste bloco arranque no nível lógico 1, portanto desactivado.

Temos associada à entrada do sinal de Sincronismo Vertical um FLIP-FLOP por forma a tornar síncrono o sinal de Sincronismo Vertical que daí para a frente vai ser utilizado.

Ao sinal de saída **/vsync_** está associado um outro FLIP-FLOP, pelas razões de sempre enumeradas.

A descrição da síntese do sinal passa por quando surge o sinal de Sinc. Vertical "síncrono"(**vsync_in_flip**), é carregado num contador o valor da duração deste novo sinal equivalente. O valor a carregar, é o referente em ciclos de clock(consoante a resolução), à duração do Sinc.Vertical mais a duração da Segunda Sequência de Pulsos de Equalização(SSPE), que é igual a 2,5H + 2,5H, em que H é a duração de uma linha. Temos assim os seguintes valores para as diferentes resoluções: Resol. Alta - 4444.4444 ; Resol. Média - 2222.2222 ; Resol. Baixa - 1111.1111.

```
SUBDESIGN CONT_VSY
(
        clock                   : INPUT;          % definição dos sinais de entrada %
        /vsync                  : INPUT;
        resol[1..0]             : INPUT;
        /reset                  : INPUT;

        /vsync_                 : OUTPUT;         % definição do uníco sinal de saída %
)
VARIABLE

        count[12..0]            : DFF;            % definição dos elementos internos ao bloco %
        vsync_out_flip          : DFF;
        vsync_in_flip           : DFF;

        maq_vsync               : MACHINE OF BITS (q1)
                                  WITH STATES (   S1 = B"0",
                                                  S2 = B"1"
                                                  );

BEGIN

        vsync_out_flip.prn = /reset;              % inicialização destes FLIP-FLOPs ao nível lógico 1 %
        vsync_in_flip.prn = /reset;

        vsync_in_flip.clk = clock;                % clock de trabalho destes elementos %
```

```
maq_vsync.clk = clock;
count[].clk = clock;
vsync_out_flip.clk = clock;

vsync_in_flip.d = /vsync;                    % sinal de entrada é sincronizado pelo clock %

/vsync_ = vsync_out_flip.q;                  % ao sinal de saída está associado um FLIP-FLOP %

CASE maq_vsync IS
        WHEN S1 =>                           % máquina de estados em S1 %
                IF !vsync_in_flip.q THEN     % sinal de Sinc. Vertical "síncrono" está activo %
                        IF resol[] == B"00" THEN     % valor a carregar para a resolução Alta %
                                count[].d = B"1000101011101";
                        ELSIF resol[] == B"01" THEN  % para a resolução Média %
                                count[].d = B"0100010101111";
                        ELSE                         % e para a resolução Baixa %
                                count[].d = B"0010001010101";
                        END IF;
                        vsync_out_flip.d = GND;      % sinal de saída contido no FLIP-FLOP é activado %
                        maq_vsync = S2;              % máquina de estados salta para S2 %
                ELSE                                 % no caso de sinal Sinc. Vertical "síncrono" ausente %
                        vsync_out_flip.d = VCC;      % o sinal de saída contido no FLIP-FLOP é desactivado %
                        maq_vsync = S1;              % e fica à espera que o Sinc. Vertical "sinc." fique activo %
                END IF;
        WHEN S2 =>                           % máquina de estados em S2 %
```

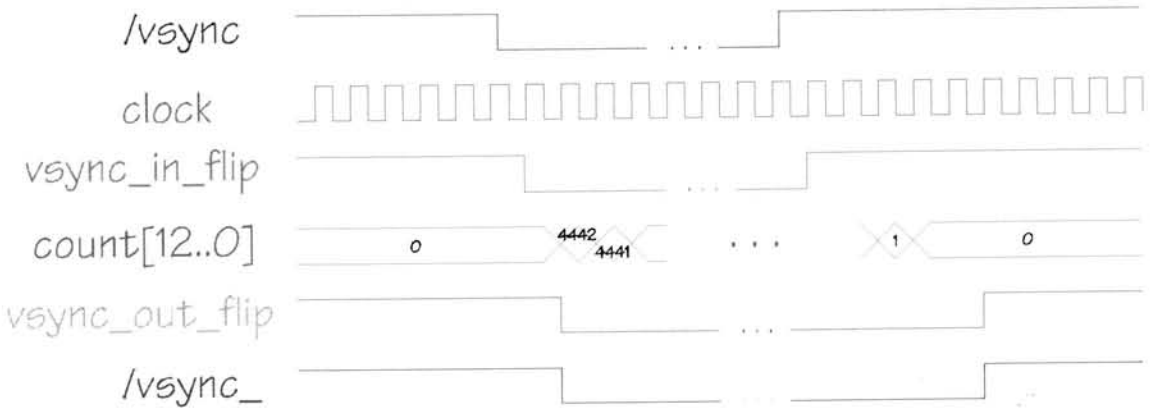
```
                IF count[].q == B"0000000000000" THEN     % no caso de ter terminada a duração do sinal de Sin. Vertical %
                        vsync_out_flip.d = VCC;           % equivalente, a saída é desactivada %
                        maq_vsync = S1;                   % máquina salta para S1,há espera que volte a surgir o S.Vert.%
                ELSE                                      % se não chegou a zero %
                        count[].d = count[].q - 1;        % contador é decremantado %
                        vsync_out_flip.d = GND;           % sinal de saída contínua activado %
                        maq_vsync = S2;                   % máquina de estados espera em S2 %
                END IF;                                   % pelo fim da duração %
        END CASE;
END;
```

Como nós não queremos adquirir nenhum pixel durante a SSPE, então arredondamos os valores atrás para o inteiro imediatamente acima.

Tal como num outro bloco em que existiam FLIP-FLOPs, é necessário fazer um ajuste no valor a carregar pelo contador. Assim temos que tirar 1 ciclo pelo atraso ao actuar a saída quando surge o sinal de Sinc. Vertical "síncrono"(**vsync_in_flip**). Outro ciclo pois o contador conta até zero, daí ser necessário fazer o reajuste, e outro ciclo ainda pois o FLIP-FLOP de saída demora um ciclo a actuar, também. Assim o valor a introduzir no contador é o seguinte consoante a resolução: Resol. Alta - 4442 ; Resol. Média - 2220 ; Resol. Baixa - 1109.

Depois de carregado o valor no contador o sinal de saída é actuado, sendo por isso colocado a 0. A máquina de estados salta para o estado S2. Aí o contador é decrementado até se atingir o valor 0(contaram-se N+1 ciclos), e a saída é colocada a 1, por forma a desactivar o nosso sinal de sincronismo equivalente. Nesta situação a máquina de estados é colocada no estado S1, esperando que volte a surgir o sinal de Sinc. Vertical "síncrono".

O esquema seguinte tenta mostrar de uma forma clara os resultados obtidos com este bloco. Neste exemplo foi admitida a resolução de trabalho mais elevada.



ROTINA GER_END

GERA ENDEREÇOS PARA AS MEMÓRIAS

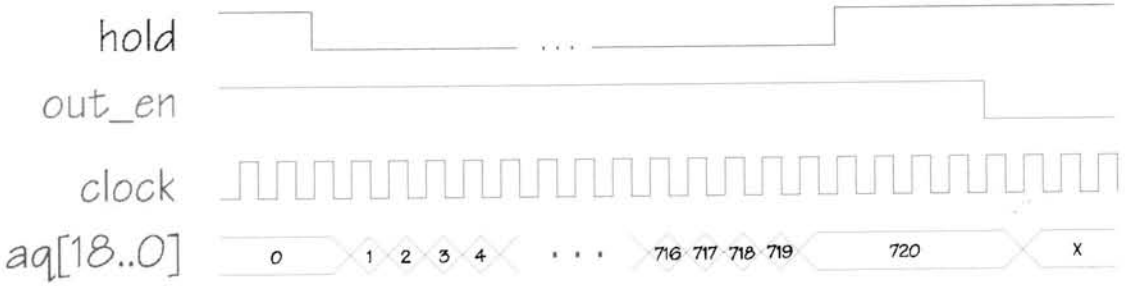
Este bloco tem como função gerar os endereços por forma a serem utilizados pelas memórias de imagem, para lá serem colocados os pixeis adquiridos. Estes endereços são gerados a partir de 19 linhas pelo facto de na resolução mais elevada termos que adquirir 720x588 pixeis, o que corresponde a 413.4375K posições de memória. Como também adquirimos linhas sem significado no início de cada

imagem(a seguir ao Sinc. Vertical equivalente) precisamos de gerar 512K posições de memória, o que corresponde às tais 19 linhas de endereço. As saídas do gerador de endereços necessitam de ser em Tri-State(alta-impedância) pois assim não é necessária a utilização de buffers de modo a permitir o acesso por qualquer outro periférico a essa mesma memória. É o caso do acesso á memória de imagem por parte de um computador para posterior visualização.

A rotina deste bloco encontra-se a seguir.

O reset do contador(tudo a zero) é feito quando surge na entrada **reset** um sinal ao nível lógico 1(este sinal é a combinação do Sinc. Vertical e do campo par ou ímpar). A entrada **hold** permite suspender a geração de endereços. Estamos numa situação de **hold** por exemplo durante o Blanking Horizontal. No que se refere à entrada de **out_en**, permite tirar estas linhas de alta impedância colocando nelas o valor do gerador de endereços.

Em termos de funcionamento no início de cada imagem é feito o **reset**(clear) do contador, ficando o gerador de endereços a apontar para a primeira posição da memória. Em seguida se o sinal de **hold** não estiver activo por cada ciclo de clock é gerado um novo endereço. Por fim se **out_en** estiver activo então nos pinos de saída deste gerador de endereços é colocado o valor actual do contador. Se não estes pinos de saída encontrar-se-ão em alta impedância. Mais uma vez a resolução adoptada neste exemplo foi a mais elevada.



ROTINA IMA_DESC

GERA SINAL QUE DETERMINA QUAL A IMAGEM A ADQUIRIR

Com este bloco pretende-se que na presença de máquinas com um poder de cálculo e velocidade dos seus circuitos internos razoável, seja possível o tratamento de imagens digitalizadas. Daí se recorrer ao intercalamento entre imagens a adquirir, de imagens a descartar, aumentando o tempo disponível para tratamento dessas

```
SUBDESIGN GER_END
(
        clock                   : INPUT;          % definição dos sinais de entrada %
        hold                    : INPUT;
        reset                   : INPUT;
        out_en          : INPUT;

        aq[18..0]               : OUTPUT;         % definição do sinal de saída %
)
VARIABLE

        open_col[18..0]         : TRI;            % Saídas do gerador de endereços em Tri-State %

        count[18..0]            : DFF;            % definição dos elementos internos ao bloco %

BEGIN

        count[].clrn = !reset;                    % contador colocado a zero quando sinal de reset activo %

        count[].clk = clock;                      % clock de trabalho do contador %

        open_col[].in = count[].q;                % valor do endereço colocado á entrada do "buffer" %
        open_col[].oe = out_en;                   % ordem para se en activo colocar na saída o que se encontra %
                                                  % na entrada, ou se não colocar a saída em Alta-Impedância %
        IF hold THEN                              % se sinal de hold activo, contador permanece parado, %
```

```
        count[].d = count[].q;
    ELSE                                        % se não contador é incrementado %
        count[].d = count[].q + 1;
    END IF;
    aq[] = open_col[].out;                      % saídas do "buffer" atribuidas aos sinais de saída(pinos) %
END;
```

imagens adquiridas. Isto tem interesse no processo de visualização, pois os computadores com os seus vulgares barramentos, precisam de algum tempo para adquirir uma imagem(colocar na sua memória).

Com este bloco e através de um valor binário de quatro bits é possivel descartar 14 imagens e adquirir uma, isto no desempenho mais baixo. Ao passo que no máximo se pode descartar uma e adquirir outra.

Convém salientar que como este bloco está implementado permite que, pelo facto de antes de uma imagem ser adquirida(a primeira) haja outras que foram descartadas, situações de instabilidade de sinais e circuitos nos momentos iniciais tendam a desaparecer quando for necessário adquirir mesmo a imagem.

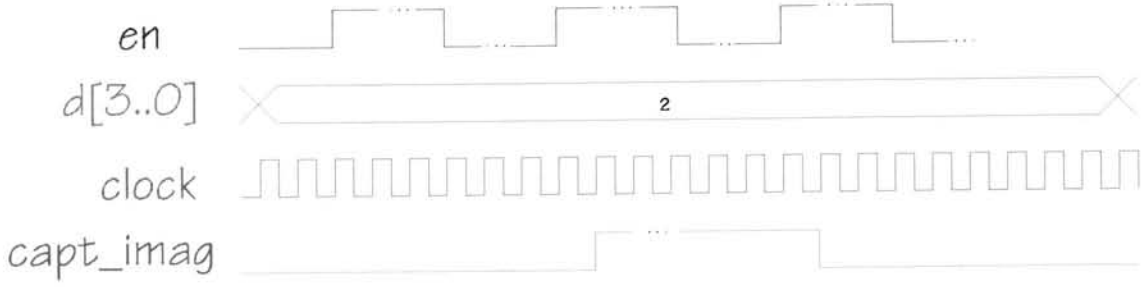
Novamente a rotina comentada deste bloco se encontra a seguir.

Como sinais de entrada temos o **en** que nos indica que estamos perante o começo de uma nova imagem, sobre a qual se tem que agir/actuar e os 4 bits de entrada para se definir o número de imagens a descartar até a próxima ser adquirida.

Associado ao pino de saída temos um FLIP-FLOP pelo motivo comum.

Quanto ao funcionamento, temos que quando surge uma nova imagem, no campo certo, o sinal de **en** fica activo. Nesta situação de **en** activo verifica-se se o valor do contador é 1. Se for é porque a imagem que vai começar é para adquirir, se não é porque ou se chegou a zero ou o valor do contador é superior a 1. No caso de ser 1 actua-se a saída por forma a infirmar que a imagem é para adquirir. Se for 0 é porque se acabou de adquirir uma imagem, e então o sinal de saída é colocado a 0 e o contador carregado com o nº de imagens a descartar. Se não se verificar nenhuma das hipóteses atrás descritas é porque ainda há imagens a descartar. Depois a máquina passa para S1, onde espera que o sinal de **en** passe a 0. Quando este passar a zero a máquina volta novamente ao estado S0 onde espera que uma nova imagem chegue para sobre ela agir(saída) ou contabilizar(contador).

Novamente será apresentado abaixo o esquema com os sinais de entrada e de saída mais importantes para a compreensão deste bloco.



```
SUBDESIGN IMA_DESC
(
        clock                   : INPUT;                % definição dos sinais de entrada %
        en                      : INPUT;
        d[3..0]                 : INPUT;

        capt_imag               : OUTPUT;               % definição do sinal de saída %
)

VARIABLE

        count[3..0]             : DFF;                  % definição dos elementos internos %
        capt_flip               : DFF;

        ss                      : MACHINE OF BITS ( q[0] )
                                  WITH STATES (   s0 = B"0",
                                                  s1 = B"1"
                                                  );

BEGIN

        ss.clk = clock;                                 % clock de trabalho dos diversos elementos %
        count[].clk = clock;
        capt_flip.clk = clock;

        capt_imag = capt_flip.q;                        % ao sinal de saída é atribuido o valor do FLIP-FLOP %
```

```
CASE ( ss ) IS
        WHEN s0 =>                                          % máquina de estados em S0 %
                IF ( en == VCC ) THEN                       % se en está activo %
                        IF ( count[].q == B"0001" ) THEN    % esta imagem é para adquirir, por isso %
                                capt_flip.d = VCC;          % coloca-se o FLIP-FLOp activo %
                                count[].d = count[].q - 1;  % e contador é decrementado %
                        ELSIF ( count[].q == B"0000" ) THEN % se não se o contador é zero, carrega-se %
                                count[].d = d[];            % o mesmo com o numero de imagens a descartar %
                                capt_flip.d = GND;          % e o sinal para capturar imagem é desactivado %
                        ELSE                                % se não ainda, %
                                count[].d = count[].q - 1;  % contador é decrementado de mais uma imagem a descartar %
                                capt_flip.d = capt_flip.q;  % sinal de saída contido no FLIP-FLOp é mantido %
                        END IF;
                        ss = s1;                            % máquina de estados salta para S1 %
                ELSE                                        % se en não está activo %
                        count[].d = count[].q;              % contador permanece com o valor que possui %
                        capt_flip.d = capt_flip.q;          % o mesmo acontecendo com o sinal de saída, %
                        ss = s0;                            % e espera que en fique activo no estado S0 %
                END IF;
        WHEN s1 =>                                          % máquina de estados em S1 %
                capt_flip.d = capt_flip.q;                  % valores inalterados %
                count[].d = count[].q;
                IF ( en == GND ) THEN                       % máquina de estados espera que o sinal en deixe de estar %
```

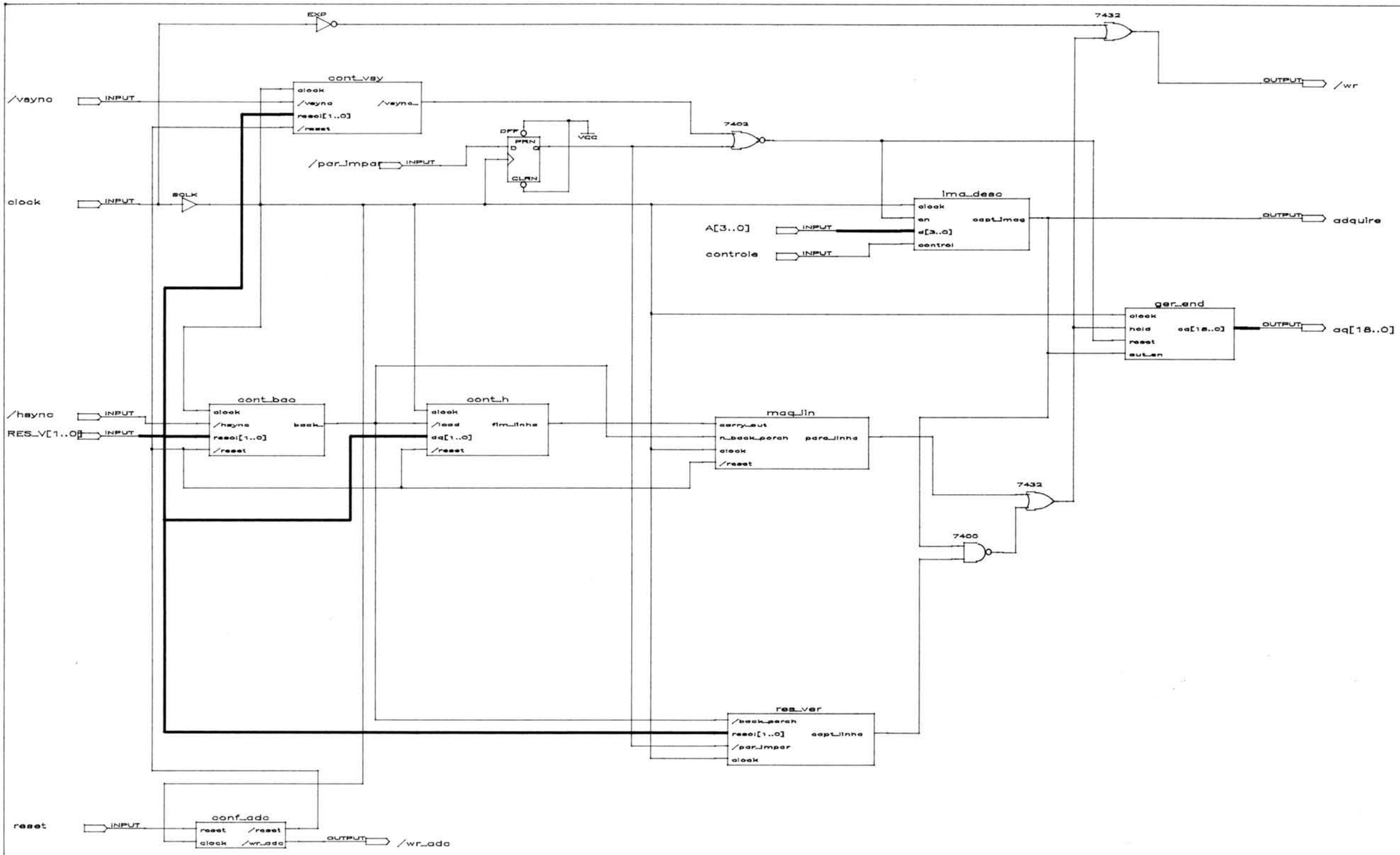
```
                    ss = s0;                    % activo para saltar para S0, e esperar que en fique outra vez %
                ELSE                            % activo %
                    ss = s1;
                END IF;
    END CASE;
END;
```

Como forma de concluir este trabalho, achei que seria de todo o interesse incluir o esquema gráfico com todos os elementos atrás descritos, bem como as suas ligações, para permitir uma visão mais clara das verdadeiras potencialidades deste programa - "ALTERA - MAX PLUS II", bem como demonstrar o que foi referido na introdução.

PRODEP

Porto, 15 de junho de 1993

Rogério Paulo Ferreira de Sousa Nunes.



TITLE	Codif. Digital de Video
COMPANY	Inesc Porto
DESIGNER	Rogerio Nunes
SIZE D	NUMBER 1.00 REV A
DATE JUNE 1993	SHEET 1 OF 1