Seminário 5

Uso do conversor A/D do μC LP2129 para a realização de um filtro numérico.

Sumário

- Especificações do filtro.
- Projeto do filtro.
- Configuração do conversor A/D.
- Configuração das interrupções.
- Desenvolvimento e teste do código no simulador.

Referências

1. Datasheet do microcontrolador Philips LPC2129.
2. Manual de uso do microcontrolador Philips LPC2129.
3. Simulador do processador ARM7.

Filtro digital

Pede-se projetar e realizar com o μC LP2129 um filtro digital do tipo passa-baixas com as seguintes especificações:

1. banda passante de 5 kHz;
2. atenuação de 20 ± 0.5 dB a 50 kHz;
3. ganho unitário na banda passante;
4. sinal de entrada proveniente de um transdutor com *range* de +/- 5V;
5. *clock* do processador de 12 MHz.

O código deve ser escrito em linguagem assembly e adequadamente comentado.

Projeto do filtro digital

Da análise das especificações fica claro que é suficiente um filtro IIR de primeira ordem.

Vimos que uma resposta equivalente se pode também realizar no modo FIR.

Porém a quantidade de cálculos requerida para uma realização FIR é sempre superior àquela necessária para um filtro IIR.

Por outro lado, o filtro FIR não apresenta problemas de estabilidade numérica, que podem se verificar com o filtro IIR, por causa do feedback interno.

Projeto do filtro digital

A realização do tipo FIR poderia ser adotada caso se estivesse trabalhando com um DSP, ou ainda se houvessem especificações sobre o defasamento da resposta em freqüência do filtro (os filtros FIR podem ter fase linear).

O uso de um μC, ainda que muito potente como aquele indicado, e o fato que não existem especificações sobre o defasamento, levam a considerar a realização do tipo IIR.

Trata-se então de realizar o algoritmo:

$$y(k) = b \cdot x(k) + a \cdot y(k-1)$$

Aquisição dos dados

Uma vez decidido o algoritmo, é necessário escolher como organizar a aquisição dos dados. O μC possui um conversor A/D interno, o que simplifica muito as coisas.

Como todos os periféricos do LP2129, também este é mapeado na memória.

É preciso aprender como programá-la e decidir uma estratégia de gestão (*polling* vs interrupção).

É importante recordar que para as aplicações de processamento de sinais a temporização é essencial.

Aquisição dos dados

É necessário que as amostras sejam adquiridas com uma frequência constante e conhecida.

Além disso, é necessário que o processamento do dado adquirido se conclua dentro do período de amostragem pré-fixado.

Estes fatos implicam em vínculos para a escolha dos parâmetros relativos ao conversor A/D.

Em outras palavras, a escolha da frequência de amostragem não dependerá apenas das especificações do filtro, mas também da qualidade (número de instruções, i.e. tempo de cálculo) do algoritmo.

Aquisição dos dados

Do manual do LP2129 tem-se que o ADC é controlado através de dois registros (ADCR e ADDR):

Name	Description	Access	Reset Value	Address
ADCR	A/D Control Register. The ADCR register must be written to select the operating mode before A/D conversion can occur.	Read/Write	0x0000 0001	0xE003 4000
ADDR	A/D Data Register. This register contains the ADC's DONE bit and (when DONE is 1) the 10-bit result of the conversion.	Read/Write	NA	0xE003 4004

O primeiro contém o conjunto de parâmetros de configuração (modo de funcionamento, frequência, …), o segundo, o dado convertido em 10 bits ADDR[15:6] e o bit (ADDR[31]) de final de conversão.

Aquisição dos dados

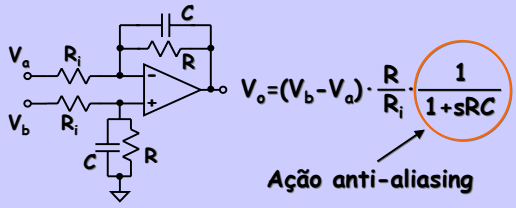
A tensão de alimentação do ADC, que representa o fundo de escala para os dados de entrada, é de 3.3V.

Devemos, porém, adquirir dados cujos valores podem estar entre –5 e +5 V.

Temos, necessidade, portanto, de um circuito de escalonamento do sinal de entrada, que o desloque e atenue até torná-lo compatível com o ADC.

O resultado da conversão representa a razão entre a tensão de entrada e o fundo de escala de 3.3V.

Aquisição dos dados



Fazendo V_a = -5 V e escolhendo a razão R/R_i igual a 3.3/10, obtemos o escalonamento necessário. Importante: a saída do OPAMP deve estar fisicamente muito próxima à entrada do ADC.

Aquisição dos dados

Após o circuito de aquisição dispomos de um sinal entre 0 e 3.3V.

Os valores menores que 1.65V serão correspondentes a tensões negativas no transdutor. Aquelas maiores, correspondem a tensões positivas.

Sendo importante manter a informação sobre o sinal, devemos prever um procedimento no programa desenvolvido.

Os dados de saída do ADC terão sempre valores positivos.

Conversão A/D

Ao término da conversão, o ADC escreve o dado no registro ADDR e coloca o bit 31 em 1.

Uma eventual rotina de *polling* pode aproveitar este fato para identificar o final da conversão.

A leitura do registro, com a conversão concluída, corresponde à leitura de um número < 0 (o bit de sinal é alto).

Se a conversão ainda estiver em curso, o número lido será positivo (ou nulo). A supervisão do periférico em *polling* é, assim, imediata.

Conversão A/D

Na realidade, ao se completar a conversão, o ADC envia sempre ao VIC uma solicitação de interrupção.

Se o correspondente bit do registro VICIntEnable for 1, esta solicitação é enviada ao processador, segundo a modalidade prevista pelo programador, ou seja, segundo a configuração do próprio VIC.

Caso contrário, a solicitação é ignorada.

É pois, possível gestir o periférico também por meio das interrupções.

Conversão A/D

Neste exemplo, o filtro é a única atividade prevista para o μC. Esta é uma situação completamente irrealística.

Na verdade, o filtro deverá fazer parte de uma série mais ampla de atividades do μC, que deverá estar livre para realizar também outras funções.

Ainda que seja possível para nós, a gestão em *polling* não é aquela que seria adotada em uma aplicação real, na qual o uso das interrupções seria a escolha obrigatória.

Conversão A/D

Programa principal

```
Inicialização
loop:  nop
       jmp loop
```

O programa, depois de uma fase de inicialização de variáveis, registros, periféricos, etc., entra em um ciclo infinito de espera.

ISR A/D

```
A/D:
       ⋮
       ret
```

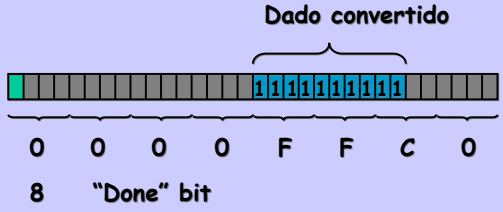
Cada vez que conversor A/D gera uma requisição de interrupção, realiza-se o algoritmo do filtro.

Organização do programa Filtro_PB

Conversão A/D

O dado convertido pelo ADC é escrito com 10 bits no ADDR[15:6]. O valor de fundo de escala será memorizado como:

ADDR = 0x0000FFC0



Projeto do filtro digital

O primeiro parâmetro que deve ser fixado no projeto de um filtro é a frequência de amostragem (f_C).

Usualmente esta é fixada pelas especificações, ou ainda é vinculada a outros aspectos do projeto.

No nosso caso, ao contrário, é um grau de liberdade do projeto.

Uma primeira estimativa pode se basear na banda passante do filtro. Sabemos que as técnicas de discretização fornecem bons resultados até 1/10 da frequência de amostragem.

Projeto do filtro digital

No nosso caso, estabelece-se um limite inferior da ordem de 50 kHz, valor abaixo do qual convém não descer.

O limite superior é dado pela potência de cálculo do μC. De fato, devemos ser capazes de completar os cálculos dentro de um período de amostragem.

Isto significa que devemos ser capazes de completar os cálculos e produzir um novo valor de y(k) no máximo em 20 μs.

Um outro limite superior para f_C poderia vir dos efeitos de quantização.

Projeto do filtro digital

- Four channel 10-bit A/D converter with conversion time as low as 2.44 µs.
- Multiple serial interfaces including two UARTs (16C550), Fast I²C (400 kbits/s) and two SPIs™.
- 60 MHz maximum CPU clock available from programmable on-chip Phase-Locked Loop.

Da análise do datasheet vê-se que o ADC do µC é capaz de adquirir dados com uma frequência máxima de cerca de 400 kHz.

Além disso, o período de clock do processador é igual a cerca de 83 ns, o que comporta a disponibilidade de cerca de 240 ciclos de clock no tempo máximo à disposição. Tem-se, pois, tempo para muitas instruções (mas não muitíssimas!).

Projeto do filtro digital

Esta rápida análise mostra que é possível resolver o problema dado pois tanto a velocidade de conversão do ADC quanto a frequência de trabalho da CPU são adequadas à realização do filtro.

Neste ponto podemos escrever um código hipotético e avaliar o tempo de execução.

Isto permitirá entender qual é a máxima frequência de amostragem f_C a qual podemos efetivamente pensar em atingir, sem perder dados.

Código do filtro digital

```
        LDR     r9, [r2]                /* Leio dado, endereço em r2  */
        BIC     r9,r9,#0x80000000       /* Limpo bit fim de conversão */
        LDR     r8, =PWMMR0             /* Endereço para o resultado  */
Filtro:                                 /* O valor adquirido está em r9*/
        LDR     r10, =b_1               /* b_1 em r10                 */
        MUL     r7, r9, r10
        MOV     r9, r7, LSR #16         /* b_1*x(k) em r9             */
        LDR     r10, =a_1               /* a_1 em r10                 */
        MUL     r7, r5, r10             /* r5 = y(k-1)                */
        MOV     r7, r7, LSR #16         /* a_1*y(k-1) em r7           */
        ADD     r5,r9,r7                /* r5 = y(k)                  */
```

O algoritmo resulta muito simples: apenas 10 instruções.

Código do filtro digital

A análise do nosso código hipotético mostra que o número de instruções para o filtro passa-baixas é muito pequeno.

No pior caso, podemos contar 3 ciclos de *clock* para as instruções não MUL e 5 para as MUL (na realidade serão menos) para um total inferior a 35 ciclos de *clock*, ou ainda, com o nosso valor de f_{clk} (12 MHz), a 3 µs.

Se fosse apenas pelo algoritmo, a frequência f_C poderia ser aumentada até mais de 300 kHz.

Na prática, convém limitar-se a um valor menor, para não aumentar os efeitos de quantização.

Projeto do filtro digital

Uma primeira escolha de f_C pode ser:

$$f_C = 100 \text{ kHz}.$$

É suficientemente maior que a banda exigida para o filtro e ao mesmo tempo compatível com o tempo de cálculo estimado e a velocidade de conversão do ADC.

Pode-se agora encontrar os valores dos coeficientes do filtro, para obter a banda desejada. Um modo elementar de resolver o problema é através da discretização de um filtro analógico equivalente.

Projeto do filtro digital

O filtro a discretizar é:

$$F(s) = \frac{1}{1 + sT_p} \qquad T_p = 3.183 \cdot 10^{-5}$$

Com a Z-form:

$$s = \frac{1 - z^{-1}}{T_C}$$

Integração de Euler, $T_C = 1/f_C$

Encontramos:

$$a\_1 = \frac{T_p}{T_p + T_C} \qquad b\_1 = \frac{T_C}{T_p + T_C}$$

Projeto do filtro digital

Substituindo os valores obtemos:

a_1 = 0.760943 b_1 = 0.239057

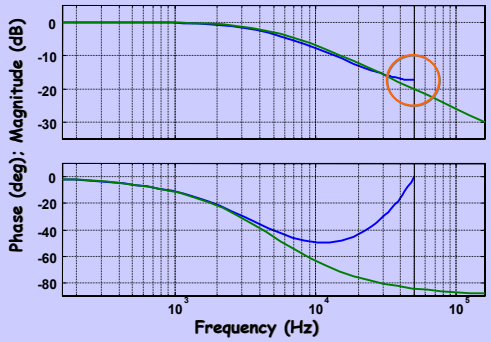
N.B: obviamente a_1 + b_1 = 1.

Devemos agora verificar se o projeto está correto.

Para tanto podemos recorrer a um programa como o Matlab, para obter a resposta em frequência e ao degrau do filtro.

Por enquanto, deixemos de lado todos os efeitos de quantização.

Projeto do filtro digital



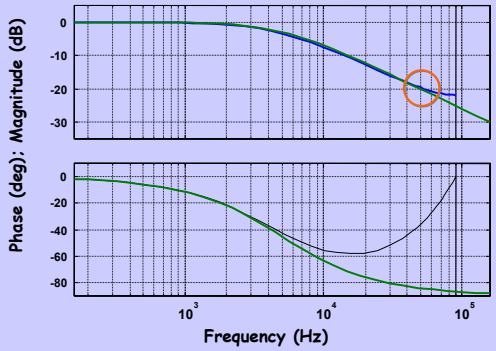
Projeto do filtro digital

A escolha de f_C leva a um erro na atenuação pedida em 50 kHz. Para reduzir este erro, mantendo a estrutura do filtro, é necessário aumentar a frequência de amostragem.

Repetimos o projeto diversas vezes aumentando sucessivamente o valor de f_C.

Para f_C = 180 kHz o erro se reduz a menos de 0.5 dB, que pode ser considerado aceitável.

Projeto do filtro digital



Projeto do filtro digital

Deve-se agora discutir a representação interna dos coeficientes do filtro.

O nosso μC dispõe de aritmética a 32 bits. No entanto, a execução de multiplicação a 32 bits requereria um acumulador de 64 bit, que não é disponível.

Portanto, a representação dos dados de entrada e dos coeficientes do filtro deve ser no máximo em 16 bits.

Pode-se aproveitar do fato que serão processados apenas números positivos.

Projeto do filtro digital

Os valores dos coeficientes (para f_C = 180 kHz) são:

a_1 = 0.851402 b_1 = 0.148598

Para representar em 16 bits sem sinal, aproveitando assim ao máximo a resolução disponível, podemos usar a relação:

$$\boxed{\text{1-LSB}} = 2^{16}-1 = 65535 = \boxed{\text{FFFF}}$$

Obtendo:

a_1_16 = 0xD9F5 b_1_16 = 0x260A

Projeto do filtro digital

Se também o conteúdo da parte baixa do registro ADDR for interpretado como dado inteiro a 16 bits (sem sinal), o valor em Volts do LSB é:

$$3.3V/2^{16} \cong 50.35\ \mu V.$$

A resolução efetiva do nosso filtro será, porém, muito mais imprecisa, por causa da quantização de entrada a 10 bits, que permite distinguir, no máximo, variações de

$$3.3V/2^{10} \cong 3.2\ mV.$$

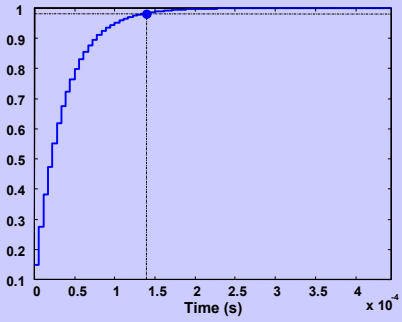
Projeto do filtro digital

Dado que usamos 16 bits para representar os valores dos coeficientes na sua representação binária, não há efeitos visíveis devido à quantização. A nossa resolução numérica é igual a $1/2^{16} = 1.53 \cdot 10^{-5}$.

Os diagramas de Bode (ideal/quantizado) resultam praticamente sobrepostos.

Qualquer efeito poderia haver devido ao truncamento implícito nas multiplicações com vírgula fixa, mas a sua avaliação pode ser feita diretamente com o sistema de desenvolvimento.

Projeto do filtro digital



Resposta ao degrau esperada para o filtro

Multiplicação

A escolha da representação dos dados tem efeito também na execução das multiplicações.

Ao final de cada multiplicação, é preciso transformar o resultado, que está em 32 bits, para 16 bits, tornando-o homogêneo com os outros dados. Isto é feito de modo a garantir que os bits tenham sempre o mesmo "peso", para salvaguardar a exatidão das eventuais somas calculadas na sequência.

No nosso caso, isto se obtém deslocando o resultado 16 bits à direita.

Multiplicação

Por exemplo: multipliquemos 0x4000 (que representa a fração 0.25, em 16 bits sem sinal) por 0xA000 (que representa a fração 0.625 sempre na mesmo modalidade).

A multiplicação dos dois números (que o processador trata como inteiros com sinal mas a 32 bits) é igual a 0x28000000 (obviamente >0).

Na nossa notação o resultado esperado será:

$0.25 \cdot 0.625 = 0.1625$ ou seja 0x2800.

O valor correto, coerente com a normalização, se obtém com um deslocamento do resultado em 16 posições à direita.

Multiplicação

Em geral, este tipo de normalização é, em efeito, uma pré-multiplicação dos operandos por um fator de escala do tipo 2^p.

Ao invés de multiplicar x e y, de fato nós multiplicamos $x \cdot 2^p$ por $y \cdot 2^p$ obtendo

$$r = x \cdot y \cdot 2^{2p}.$$

Se deslocamos este número, à direita, de p bits, obtemos então

$$r' = x \cdot y \cdot 2^p,$$

que se encontra na mesma escala dos operandos iniciais.

Realização do filtro digital

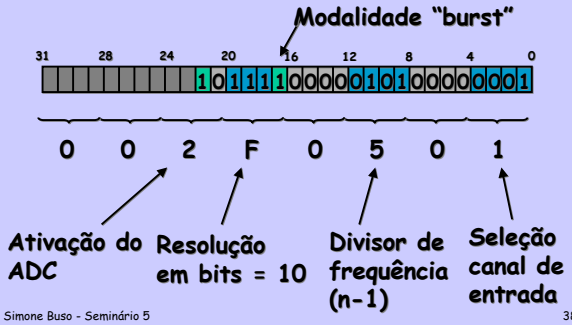
Inicialmente devemos configurar o sistema de aquisição de modo que trabalhe em 10 bits e a 180 kHz. As sequências de bits:

```
.equ  VPB_CFG,      0x00000001
.equ  AD_CFG,       0x002F0501
```

produzem este efeito. A primeira, uma vez escrita no registro VPBDIV, impõe uma frequência de *clock* para o bus dos periféricos (pclk) igual àquela da CPU (cclk). A segunda, uma vez copiada no registro ADCR, configura o ADC como se segue.

Realização do filtro digital

Configuração do registro ADCR:



Realização do filtro digital

Configuração do registro ADCR.

Dividindo a frequência do *clock* por 6 e lembrando que a conversão requer 11 ciclos da *clock*, tem-se que o conversor A/D, funcionando em modalidade "burst", terá os dados disponíveis para o processamento em uma frequência igual a:

$$f_C = f_{clk}/66 \cong 182 \text{ kHz}$$

que é um valor suficientemente próximo àquele de projeto.

Realização do filtro digital

Configuração do registro ADCR.

Teria sido possível fazer o conversor operar no modo "single shot", comandando o início da conversão na frequência requerida por meio do uso de um timer. A conversão deveria ser muito mais veloz, para deixar espaço ao processamento.

Isto é desvantajoso para a precisão da conversão e para o consumo de potência.

A solução adotada, além disso, utiliza menos recursos.

Realização do filtro digital

Configuração das interrupções.

É necessário sinalizar, na ocorrência de final de conversão, uma ISR, que deve ser alocada na posição correta do vetor de interrupções. Copiando a sequência de bits:

```
.equ  VICInt_CFG,   0x00040000
```

nos registros VICIntEnable e VICIntSelect obtém-se ativação da IR associada à ADC e a sua classificação como FIQ. às IR deste tipo é reservada a posição 0x0000001C

do vetor de interrupções

Realização do filtro digital

Configuração das interrupções.

Devemos alocar naquele endereço a nossa ISR ou ao menos uma instrução de salto à rotina.

```
Vectors:  LDR   PC, Reset_Addr
          LDR   PC, Undef_Addr
          LDR   PC, SWI_Addr
          LDR   PC, PAbt_Addr
          LDR   PC, DAbt_Addr
          NOP                     /* Reserved Vector */
          LDR   PC, IRQ_Addr
          LDR   PC, FIQ_Addr
```

Vetor das interrupções.
Vectors = 0x0000000, endereço 0 da memória.

Realização do filtro digital

Configuração das interrupções.

A alocação da rotina no endereço correto é obtida no módulo de configuração do nosso programa:

```
FIQ_Addr:      .word   FIQ_Handler
FIQ_Handler:   B       Isr_AD
```

uma vez resolvido pelo assemblador, produzirão o salto à nossa rotina (ISR_AD) a cada ocorrência da Fast Interrupt reQuest.

Realização do filtro digital

Programação da ISR.

A ISR do ADC contém o verdadeiro e próprio programa. O algoritmo que realiza o filtro é aquele apresentado antes.

A este se adiciona um segmento de código que permite registrar os primeiros 512 valores da sequência de saída do filtro na memória do μC, de modo a poder, visualmente, verificar o seu correto funcionamento.

Neste ponto, o código pode ser testado e.g. Através do simulador. Sinal de teste: onda quadrada (1.65, 1200).