SEMBscope - Osciloscópio Digital em Arduino Uno

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores
Sistemas Embarcados 2017/2018
António Pereira - up201404797@fe.up.pt
Carlos Neto - up201405711@fe.up.pt
Isabel Oliveira - up201403560@fe.up.pt

***Resumo*—Com recurso a um Arduino Uno (ATmega 328/P 16MHz) foi desenvolvido um osciloscópio digital de 2 canais e a respetiva interface gráfica em linguagem JAVA©, em que as operações em tempo real são controladas através de um μkernel cooperativo de prioridades fixas *tick based*. Foram implementadas várias funcionalidades encontradas em osciloscópios comerciais, como *trigger*, mudança das escalas vertical e horizontal e medidas de tensão máxima, mínima e média, para além de um modo de alta resolução (HRM - *High Resolution Mode*), que permite ver uma janela temporal do sinal com um maior número de amostras.**

***Palavras-chave*—Tempo real, Osciloscópio, Arduino, Sistemas Embarcados**

I. Introdução

No âmbito da unidade curricular de Sistemas Embarcados foi desenvolvido um osciloscópio digital recorrendo a um Arduino e a uma aplicação JAVA a correr num computador, onde a amostragem do sinal analógico é realizada pela placa e os seus componentes integrantes, para além de todo o conjunto de operações de comunicação entre o computador e o μ-controlador.

O projeto tem como objetivo identificar as tarefas executadas pelo sistema embarcado e dimensionar um escalonamento destas de tal forma a que o sistema seja capaz de realizar a transformação das entradas em saídas de forma a interagir com o meio em tempo real.

Este relatório tem como finalidade apresentar a metodologia de desenvolvimento do projeto e os resultados experimentais obtidos.

II. Funcionalidades Implementadas

Foram implementadas as seguintes funcionalidades, existentes nos oscilosópios convencionais:

- Modo 1 canal;
- Modo 2 canais;
- Escala temporal;
- Escala de amplitude;
- Deslocamento vertical de ambos os canais;
- *Trigger*: ascendente, descendente e possibilidade de escolher a *source* do mesmo;
- Medidas de tensão máxima, mínima e média de ambos os canais (só quando a onda está "*triggada*").

Para além destas funcionalidades, foi ainda implementada uma funcionalidade designada por modo de alta resolução, detalhada nas secções seguintes.

III. *Hardware*

A. Arduino

Foi utilizado um Arduino Uno com processador ATmega 328/P de *clock* de $16MHz$. Este Arduino possui uma ADC de 10 bits, no entanto, usaram-se apenas 8 bits para se poder aumentar o *clock* da ADC acima dos $200kHz$, uma vez que o ENOB (*Effective Number of Bits*) diminiu com o aumento da frequência e para 8 bits é possível aumentar esta frequência para $1MHz$, que foi o valor que se utilizou. A ADC operou sempre em modo *Single Conversion*, isto é, as conversões são forçadas pela escrita do bit *Start Conversion* a "1"do registo *ADC Control and Status Register*, o que permite um controlo preciso dos instantes de amostragem, requisito essencial num osciloscópio.

B. Porta Série

A comunicação entre o Arduino e o computador foi feita por USART. Esta comunicação é *full-duplex* e foi usada uma *baudrate* de 230400 baud/s. Isto significa que uma amostra de 8 bits é enviada a uma cadência de 23040 samples/s, tendo em conta que a cada 8 bits são, de facto, enviados 10 por causa do acresento de um *start* e *stop* bit. Assim, o tempo de envio de uma amostra é de $43.4\mu s$.

IV. Requisitos e Limitações Temporais

A grande limitação temporal existente corresponde ao envio de uma amostra pela porta série. Como já apresentado anteriormente, o envio de um *byte* demora $43,4\mu s$. De forma a implementar um protocolo de comunicação com a aplicação JAVA, o envio de dados requer um *byte* adicional para simbolizar o fim da comunicação, tendo sido utilizado a função *Serial.println()* para enviar a amostra seguido do referido *byte*. Como tal, o envio completo de uma amostra demora então $86,4\mu s$, sendo este tempo o fator limitante para o escalonamento.

Existe ainda outra limitação temporal na conversão da ADC. Após o início forçado da conversão é necessário esperar 13 *clocks* (tempo de conversão da ADC) para se obter um valor correto. Por esta razão, todas as outras tarefas têm um *offset* de forma a implementar a espera necessária.

Por outro lado, outro requisito temporal existente reside na necessidade de se estabelecer uma amostragem em instantes bem definidos, para permitir a correta representação da onda.

Devido à natureza do problema, as *deadlines* envolvidas são todas do tipo *soft deadlines*, dado que os valores obtidos

após uma *deadline miss* são ainda válidos para a aplicação, no entanto, degrada-se a qualidade da onda representada.

V. Arquitetura de *Software*

A. μ*Kernel*

Usou-se um *Kernel* cooperativo de prioridades fixas *tick based*, fornecido pelos docentes da Unidade Curricular, acrescido de uma funcionalidade que permite trocar a função que uma tarefa executa sem a remover – algo importante para tornar o modo HRM mais eficiente. Foi utilizado um *Kernel* cooperativo em detrimento de um preemptivo, visto que não existe necessidade de adquirir amostras a uma taxa muito superior à taxa de envio pois iria resultar na perda de amostras. Devido à limitação temporal do envio de uma amostra, o período de um *tick* corresponde a $100\mu s$. Este *tick* foi definido com um *timer* de 8 bits (*Timer* 2).

B. Escalonamento - Overview

O escalonamento varia consoante os comandos recebidos da aplicação JAVA. Sempre que se inicia a aplicação, o escalonamento é o do Modo Normal e o canal escolhido é o número 1. O utilizador pode, posteriormente, escolher o outro canal ou mesmo os dois canais, não sendo o escalonamento alterado nestes casos. Contudo, quando é feito o pedido de entrada em modo de alta resolução e vice versa, o escalonamento é alterado, como explicado na subsecção E.

C. Modo Normal - 1 Canal

O Modo Normal para 1 canal executa 4 tarefas, apresentadas por ordem decrescente de prioridade:

- **Tarefa 1: Aquisição** [T = 3[1]] – A cada execução é iniciada uma nova conversão na ADC.
- **Tarefa 2: Leitura** [T = 3] – A cada execução o registo da ADC é lido para armazenar a amostra.
- **Tarefa 3: Envio** [T = 3] – A cada execução é enviada a amostra recolhida.
- **Tarefa 4: *Polling*** [T = 3] – A cada execução é realizada a verificação se foi recebido algum comando por parte do utilizador. O comando recebido serve para alterar os modos entre a representação do canal 1, canal 2, dois canais e modo de alta resolução.

D. Modo Normal - 2 Canais

Tal como no Modo Normal para 1 Canal, este modo executa as 4 tarefas referidas, com ligeiras alterações.

Na tarefa de Aquisição, são adquiridas 512 amostras referentes a um canal e posteriormente 512 relativas ao segundo canal. Esta mudança é conseguida através da escrita de um bit para o multiplexador da ADC, que seleciona qual o pino da placa que serve de entrada para a ADC. Para sincronizar a leitura dos dois canais em simultâneo com a aplicação JAVA, é definida uma *flag* a ser transmitida, na tarefa de Envio, antes do início da recolha das 512 amostras referentes a um canal. Para não interferir com a gama dinâmica da conversão, o valor das *flags* escolhidas foram 256 e 257 para simbolizar o canal 1 e 2, respetivamente.

[1]Períodos em *ticks*

E. High Resolution Mode - HRM

Ao entrar em modo de alta resolução, pretende-se que se seja capaz de amostrar o sinal com uma frequência superior à utilizada no modo normal. Assim, reduz-se o intervalo entre ativações da tarefa de aquisição do sinal para o mínimo, isto é, ativa-se esta tarefa a cada *tick*.

Com isto, consegue-se que, para o mesmo intervalo de tempo, sejam apresentadas o dobro das amostras neste modo. Porém, tal redução do *deadline* desta tarefa de alta prioridade não permite manter um escalonamento *feasible* enviando a amostra obtida de forma sequencial. A solução passa então pela existência de um *buffer* onde se guarda a amostra antes de dar início a uma nova aquisição. O envio ocorre, então, quando se enche totalmente o *buffer*. Este *buffer* foi sobredimensionado para 1536 amostras.

Este funcionamento implica que, quando se dá a passagem para este modo, se modifique o escalonamento. Por isso, são removidas todas as tarefas do μ*kernel* e adicionadas novas. De forma dual, quando se volta ao modo normal, é reposto o escalonamento supracitado no ponto anterior.

Assim, definem-se apenas três tarefas: aquisição HRM, envio HRM e *polling*. A tarefa de *polling* é a mesma que no modo normal.

Na realidade apenas duas tarefas são escalonadas em cada momento: aquisição HRM + *polling* ou envio HRM + *polling*. Isto é conseguido mudando o apontador da função da 1ª tarefa de 1536 *ticks* em 1536 *ticks*. Tal abordagem é necessária, uma vez que não existe qualquer vantagem em adquirir uma amostra e enviá-la de seguida, pois o tempo necessário para a enviar é muito superior ao de aquisição e ambas as tarefas acedem ao mesmo *buffer*, pelo que após a tarefa conseguir enviar um conjunto completo de amostras, já o conjunto seguinte teria sido reescrito, ou seja, conseguir-se-ia adquirir vários *buffers* mas apenas enviar um.

As tarefas seguintes estão ordenadas por prioridade decrescente (uma vez que a tarefa 1 e 2 se substituem, ambas são a mais prioritária no seu momento de execução), tal como se pode verificar na figura 1:

- **Tarefa 1: Aquisição** [T = 1] – A cada execução, guarda-se uma amostra e inicia-se a aquisição da seguinte. Quando o *buffer* enche, esta tarefa é trocada pela tarefa 2 (desta forma, a troca ocorre a cada 1536 *ticks*).
- **Tarefa 2: Envio** [T = 1] – Envia-se uma amostra do *buffer* em cada execução da tarefa. Caso já se tenham enviado todas as amostras, repõe-se a tarefa 1.
- **Tarefa 3: *Polling* à porta série** [T = 1] – Tal como no modo normal, esta tarefa realiza a verificação de receção de qualquer *input* por parte da aplicação JAVA.

VI. Análise de Escalonabilidade

Como anteriormente apresentado, as tarefas em modo normal têm períodos de 3 *ticks*. Deste modo, da forma que o μ*Kernel* está construído, apesar do período ser de 3 *ticks*,

apenas ao 4º *tick* é que a tarefa é executada, o que faz com que a periodicidade destas tarefas seja de $400\mu s$. Desta forma, no modo normal para 1 canal, como na interface gráfica da aplicação JAVA a onda é representada após a receção de 512 amostras, a *refresh rate* é de $512 * 400\mu s = 204,8ms \rightarrow 4.88Hz$. No modo de 2 canais, como é necessário esperar por receber 512 amostras referentes a cada canal, num total de 1024, a *refresh rate* é de $1024*400\mu s = 409,6ms \rightarrow 2.44Hz$. No modo de alta resolução, são adquiridas 1536 amostras de $200\mu s$ em $200\mu s$, logo a *refresh rate* é de $1536 * 200\mu s = 307,2ms \rightarrow 3,26Hz$.

Verificou-se que o próprio *Kernel* introduz *overhead*, o que provoca algum *jitter*. Em particular, o tempo de execução de cada *tick* é de cerca de $12,6\mu s$. Esta foi também umas das razões para a escolha de um *Kernel* cooperativo, já que o preemptivo não era necessário e verificou-se que introduzia um *overhead* ainda maior.

Assim, com base nos resultados dos valores do tempo de execução medidos experimentalmente ligando uma saída digital quando se inicia a execução de uma tarefa e desligando-a quando a execução termina (figuras 3, 4, 5), apresentam-se os seguintes resultados:

Tarefas	T	D	WCET
τ_1	$400\mu_s$	$0\mu_s$	$5,2\mu_s$
τ_2	$400\mu_s$	$100\mu_s$	$4,4\mu_s$
τ_3	$400\mu_s$	$100\mu_s$	$214\mu_s$
τ_4	$400\mu_s$	$100\mu_s$	$6\mu_s$

Tabela I
TAREFAS DO MODO NORMAL

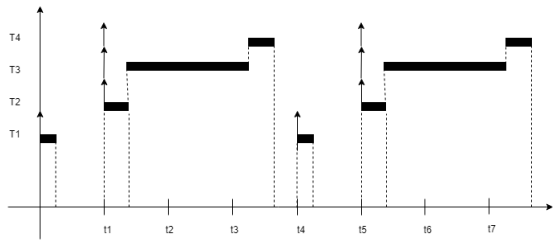

Figura 1. Gantt Chart das Tarefas em Modo Normal

$$Rwc_1 = I_1 + C_1 = 0 + 5,2\mu s = 5,2\mu s \quad (1)$$

$$Rwc_2 = I_2 + C_2 = 0 + 4,4\mu s = 4,4\mu s \quad (2)$$

$$Rwc_3 = I_3 + C_3 = Rwc_2 + C_3 = 4,4 + 214 = 218,4\mu s \quad (3)$$

$$Rwc_4 = I_4 + C_4 = Rwc_3 + C_4 = 218,4 + 6 = 224,4\mu s \quad (4)$$

Deste modo, somando o valor do *offset* das 3 tarefas ao WCRT da última tarefa a executar, obtemos $324,4\mu s$, o que garante a escalonabilidade das tarefas, o que se pode confirmar consultando a figura 4 em anexo.

Para o *High Resolution Mode*, como as tarefas têm períodos de 1 *tick*, estas são ativadas de $200\mu s$ em $200\mu s$. É importante voltar a referir que, neste modo, as tarefas τ_1 e τ_2 nunca executam simultaneamente, como explicado anteriormente e como representado na figura 2 e verificado experimentalmente (figura 5).

Tarefas	T	D	WCET
τ_1	$200\mu_s$	$0\mu_s$	$7,2\mu_s$
τ_2	$200\mu_s$	$0\mu_s$	$224\mu_s$
τ_3	$200\mu_s$	$0\mu_s$	$6\mu_s$

Tabela II
TAREFAS DO *High Resolution Mode*

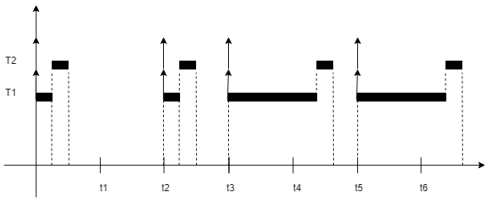

Figura 2. Gantt Chart das Tarefas em *High Resolution Mode*

Assim, quando τ_1 está em execução:

$$Rwc_1 = I_1 + C_1 = 0 + 7,2\mu s = 7,2\mu s \quad (5)$$

$$Rwc_2 = I_2 + C_2 = 7,2 + 6\mu s = 13,2\mu s \quad (6)$$

Quando τ_2 está em execução:

$$Rwc_1 = I_1 + C_1 = 0 + 224\mu s = 224\mu s \quad (7)$$

$$Rwc_2 = I_2 + C_2 = 224 + 6\mu s = 230\mu s \quad (8)$$

É importante referir que neste modo é fundamental que a aquisição das amostras seja feita com periodicidade de $200\mu s$, de forma a duplicar a qualidade da resolução. Para o envio não existe esta necessidade, uma vez que se as amostras chegarem com um pequeno atraso à aplicação JAVA, este não será percetível para o utilizador e a qualidade da onda não se degrada, pois as amostras foram adquiridas em instantes precisos. Quando a tarefa de envio acabar de enviar as 1536 amostras, a amostragem é retomada periodicamente, nunca perdendo a periodicidade bem definida.

Para o modo normal, visto que as amostras estão espaçadas de $400\mu s$, a frequência de amostragem será de $2,5kHz$. Segundo o teorema da amostragem de *Nyquist*, só é possível representar sinais até $1,25kHz$, contudo neste caso só são adquiridas duas amostras do sinal num período. Se se considerar que, para uma correta visualização do sinal, são necessárias, no mínimo, 10 amostras, então a frequência máxima a representar no modo normal é de $250Hz$. Verificou-se durante a demonstração e em testes anteriores que a máxima frequência atingida foi de cerca de 205Hz no modo normal.

Para o *High Resolution Mode*, visto que a frequência de amostragem é o dobro, com um raciocínio análogo, a frequência máxima a representar será de $500Hz$.

VII. Análise de Memória

O código ocupa 3682 bytes de 32256 bytes disponíveis, correspondendo assim a 11% de utilização da ROM.

As variáveis globais ocupam 1870 bytes de 2048 disponíveis, correspondendo a 91% de utilização da RAM. Este valor deve-se à existência de um *buffer* de 1536 posições, necessário para o HRM.

Não foi usado qualquer tipo de alocação dinâmica.

VIII. Conclusão

Foi gravado um vídeo de demonstração do projeto que pode ser visto no seguinte link: https://youtu.be/hvTrrLid0ZQ.

Com o trabalho desenvolvido prova-se que é possível implementar um osciloscópio digital com um Arduino UNO, recorrendo a um *Kernel* cooperativo de prioridades fixas *tick based*.

Foram feitos dois escalonamentos distintos para os modos normal e de alta resolução, provando-se que ambos são *feasible*.

A maior limitação temporal deve-se à velocidade da porta série, o que influenciou a escolha dos *ticks* para $100\mu s$ e do período das tarefas para 3 *ticks*, limitando, assim, a frequência máxima teórica a 1.25kHz. Na prática, para uma boa visualização dos sinais, a frequência máxima obtida foi de cerca de 205Hz. O modo de alta resolução permite uma taxa de amostragem duas vezes maior, conseguindo-se, deste modo, frequências máximas da ordem dos 400Hz.

IX. Organização do Grupo

O grupo trabalhou sempre presencialmente com todos os elementos a contribuir de igual forma para todas as partes do projeto.

Assim, as percentagens atribuídas a cada elemento são:

- António Pereira: 33.33%
- Carlos Neto: 33.33%
- Isabel Oliveira: 33.33%

X. Referências

- ATmega 328/P Datasheet
  http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf
- http://www.instructables.com/id/Girino-Fast-Arduino-Oscilloscope/

XI. Anexos

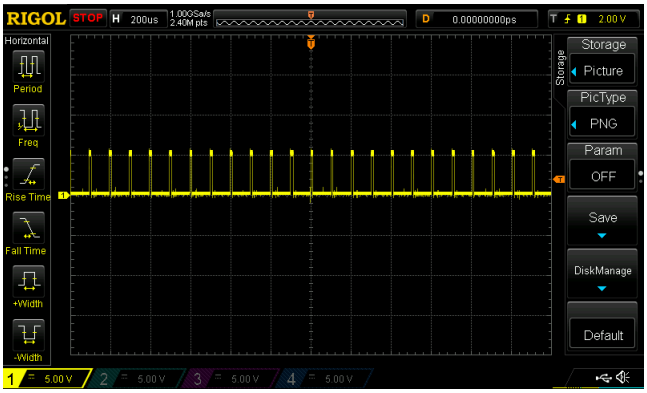

Figura 3. *Tick* (T = $100\mu s$)

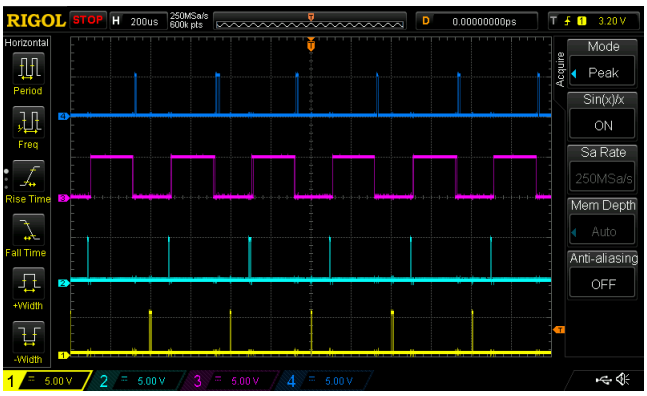

Figura 4. Gantt Chart das Tarefas em Modo Normal Obtido Experimentalmente

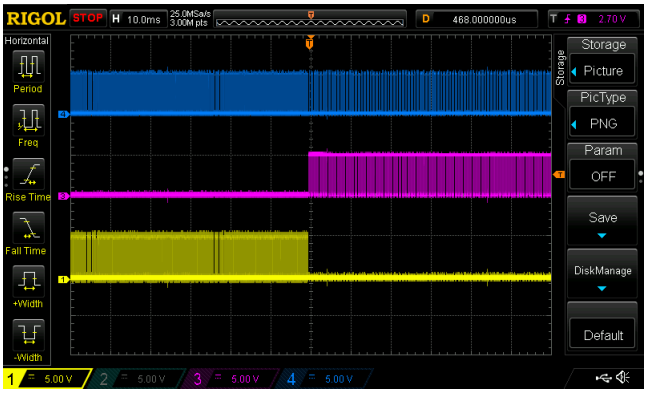

Figura 5. Gantt Chart das Tarefas em *High Resolution Mode* Obtido Experimentalmente