Desenvolvendo Sistemas de Tempo Real em Processadores de Aplicação

Por **Guilherme Fernandes** - 23/03/2017

A utilização de processadores está em expansão contínua. SoCs (System-on-Chip) baseados em arquitetura ARM estão dominando aplicações tradicionalmente desenvolvidas com microcontroladores pequenos, como, por exemplo, os ARM Cortex-M. Essa tendência é motivada por diversos fatores como:

- Os intensos requisitos por conectividade, relacionados a IoT e, não somente do ponto de vista de hardware, mas também relacionados a software, à segurança e a protocolos de comunicação;
- A crescente necessidade por interatividade com o usuário por meio de telas sensíveis ao toque e de alta resolução;
- A produção em escala dos processadores vem derrubando consideravelmente o preço dos SoCs. Isso os torna cada vez mais viáveis no momento da escolha da arquitetura adotada (em projetos) no projeto.

O caso típico que observamos, originados das tendências descritas acima, são os diversos clientes que entram em contato todos os dias, pois estão iniciando a modernização de seus produtos fazendo o upgrade de um projeto baseado em microcontrolador para um baseado em microprocessador. Esse upgrade apresenta novos desafios, uma vez que o projeto eletrônico é mais complexo e a camada de sistema operacional é muito mais abstrata.

As dificuldades de um projeto de hardware com microprocessadores (ou SoCs) é facilmente superada por meio da utilização de projetos de referência ou alternativas de prateleira como computadores em módulo ou single board computers.

Já na camada de sistema operacional, a utilização de distribuições de Linux Embarcado está consolidada e amplamente utilizada na indústria. Além do mais, uma infinidade de ferramentas Open Source está disponível para simplificar o desenvolvimento de sistemas embarcados complexos e ricos em recursos. Sistemas similares, se desenvolvidos em microcontroladores, seriam extremamente complicados, consumindo, assim, muitos recursos, como tempo de engenharia para desenvolvimento, teste e manutenção. Apesar de todos os benefícios, o uso de sistemas operacionais, como o Linux, levanta diversas questões e suspeitas relacionadas a determinismo e outros conceitos de tempo-real.

Uma abordagem tradicionalmente utilizada por desenvolvedores é a estratégia de separar as tarefas críticas (em tempo) em diferentes processadores. Desta maneira, um processador Cortex-A ou similar é utilizado para os recursos de multimídia e conectividade enquanto um microcontrolador é ainda empregado para as tarefas de tempo-real e que precisam de determinismo. O objetivo deste artigo é apresentar algumas opções que os desenvolvedores podem considerar ao desenvolver sistemas de tempo-real utilizando processadores. Nós apresentaremos três possíveis soluções que habilitam a capacidade de tempo-real a projetos

Testando o desempenho de tempo-real

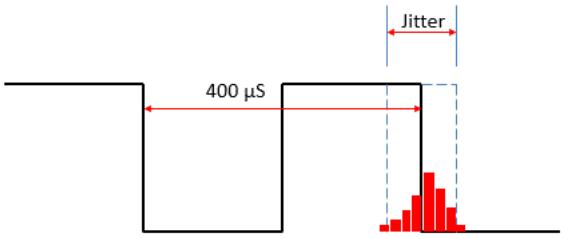

Figura 1: Medida do Jitter

Existem diversas ferramentas de benchmarking para avaliar o desempenho de um sistema de tempo-real, entretanto, neste artigo gostaríamos de fazer um teste rápido (e descomplicado) que proporcione uma percepção sobre qual o desempenho de cada abordagem e o efeito que ela produz nos resultados medidos. Assim, optamos por fazer a medida do "jitter" em uma onda quadrada gerada a partir de um GPIO sendo acionado em um programa desenvolvido em cada abordagem apresentada. Com isso, podemos facilmente construir uma percepção preliminar e rapidamente analisar as mudanças de desempenho obtidas de uma abordagem em relação à outra. Desenvolvemos, portanto, uma pequena aplicação que altera o estado de uma saída digital a uma frequência de 2,5KHz (200µs High / 200µs Low). A saída digital foi conectada a um analisador lógico onde as medidas precisas da onda quadrada foram capturadas e analisadas.

Os resultados dos experimentos mencionados acima, em um Linux Standard, rodando em um processador ARM NXP i.MX6 (Colibri iMX6 da Toradex) estão apresentados na figura abaixo. A tarefa responsável por executar as alterações de estado do output digital foi configurada com uma tarefa RT (SCHED_RR) e a configuração de Kernel adotada foi a Voluntary Kernel Preemption (CONFIG_PREEMPT_VOLUNTARY).

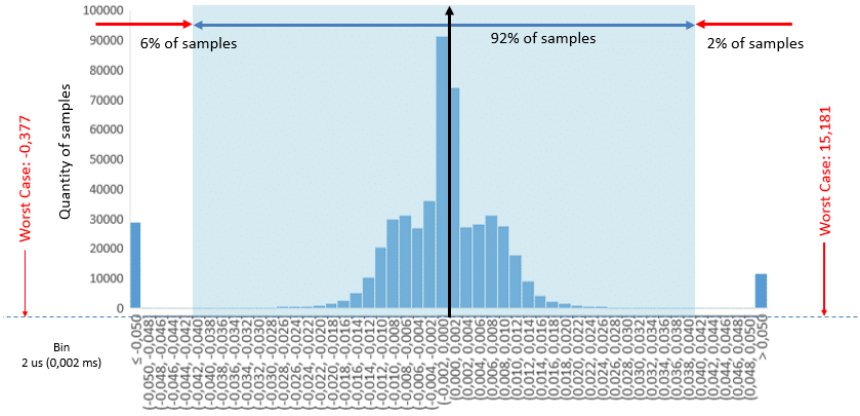

Figura 2: Histograma de medidas de erro dos testes realizados com o Kernel Linux padrão

A distribuição das medidas revela que apenas 92% das amostras coletadas foram de fato realizadas em um intervalo com erro de ±10% do período (400us ± 40us). Além disso, o pior caso medido apresentou um atraso de 15ms, ou seja, um erro de 3700% do período desejado.

Real-Time

A primeira estratégia apresentada neste artigo é baseada puramente em software. O Linux não é um Sistema Operacional de Tempo Real, entretanto existem iniciativas capazes de melhorar substancialmente o desempenho dele em tempo real. Um desses esforços é o Real-Time Linux project. O Real-Time Linux é composto por um conjunto de patches (PREEMPT_RT) destinados a adicionar novas estratégias de preempção ao Kernel do Linux em conjunto com novos recursos e ferramentas que melhoraram a usabilidade dele para tarefas de tempo real. Você pode encontrar a documentação de como aplicar o PREEMPT_RT patch ao Kernel do Linux e a como desenvolver aplicações no site oficial Real-Time Linux Wiki ou aqui.

Executamos o teste de desempenho utilizando o PREEMPT_RT patches em nosso Colibri iMX6DL para exemplificar a melhoria no desempenho do resultado do Jitter. Toda a documentação de como preparar a imagem de Linux da Toradex aplicando o PREEMPT_RT patch está disponível neste link. O histograma abaixo mostra os resultados no sistema configurado com o PREEMPT_RT e configurado para Preempção de tempo real (Real-Time Preemtion). A análise dos resultados mostra que apenas 0,002% das amostras apresentam erro superior a ±10% do período. Adicionalmente, o pior caso (0,106us) é “apenas” 25% do período (uma melhoria significativa se comparado ao Linux padrão).

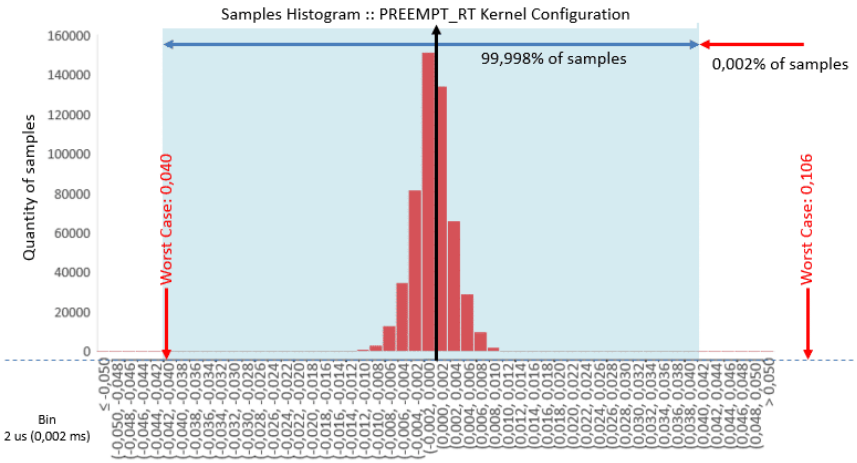

Figura 3: Histograma de medidas realizadas com o Linux Preempt-RT kernel.

Um exemplo de software que confia no PREEMPT_RT patch é a solução de CLP por Software Codesys Solutions. Eles se apoiam no Real-Time Linux kernel em conjunto com a OSADL para executar a solução de CLP por software. Essa solução está espalhada pela indústria em milhares de CLPs, CNCs e outros comandos de máquinas. O vídeo abaixo apresenta a solução da Codesys rodando em um Colibri iMX6DL 512MB. Você pode encontrar mais informações sobre o Codesys nos módulos Toradex, incluindo um demo executável, neste link.

Xenomai

Xenomai é outro framework popular para transformar o Linux em um Sistema de tempo real. O Xenomai atinge seu objetivo adicionando um co-kernel ao kernel do Linux. O co-kernel gerenciará as tarefas críticas com restrições temporais e terá prioridade maior com relação ao Kernel padrão. Você pode obter informações adicionais aqui. É importante comentar que para utilizar as capacidades do Xenomai é mandatório o uso das suas APIs (conhecidas como libcobalt) para interface entre as aplicações do usuário e o Cobal core, que é responsável por garantir o determinismo de tempo real.

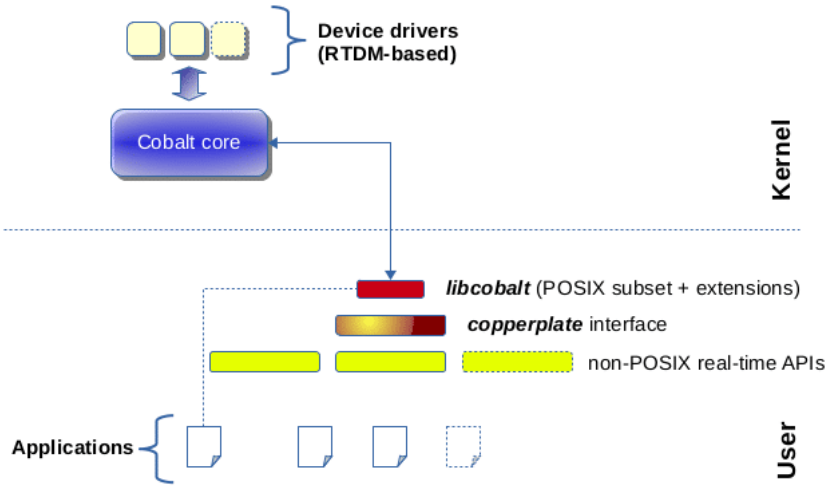

Figure 4: Configuração Dual Core Xenomai. Fonte: https://xenomai.org/start-here/

A documentação sobre como instalar o Xenomai em seu target pode ser encontrado no website. Adicionalmente, existe uma lista de placas embarcadas que já foram testadas, veja aqui a hardware reference list (que inclui a série de processadores NXP i.MX).

Para validar o uso do Xenomai no SoC i.MX6, executamos mais uma vez nosso experimento. O target foi um Colibri iMX6 da Toradex. Foram executados os testes com a mesma abordagem descrita para os PREEMPT_RT. Algumas partes da aplicação desenvolvida são apresentadas abaixo para ilustrar o uso das APIs do Xenomai

```
1  void blink(void *arg __attribute__((__unused__)))
2  {
3    int iomask = 0;
4    rt_task_set_periodic(NULL, TM_NOW, TIMESLEEP);
5    while(1)
6    {
7      rt_task_wait_period(NULL);
8      if(iomask) SET_G35;
9      else CLR_G35;
10     iomask = 1 - iomask;
11   }
12 }
13
14 int main(void)
15 {
16   /* Task Creation */
17   rt_task_create(&blink_task, "blinkLed", 0, 99, 0);
18   rt_task_start(&blink_task, &blink, NULL);
19
20   getchar();
21   rt_task_delete(&blink_task);
22
23   return 0;
24 }
```

Os resultados dos testes utilizando Xenomai são apresentadas no gráfico abaixo. Mais uma vez, a solução apresenta uma perceptível vantagem com relação ao Linux padrão, repare que neste caso, o pior caso está contido em uma área de erro de apenas ±10%.

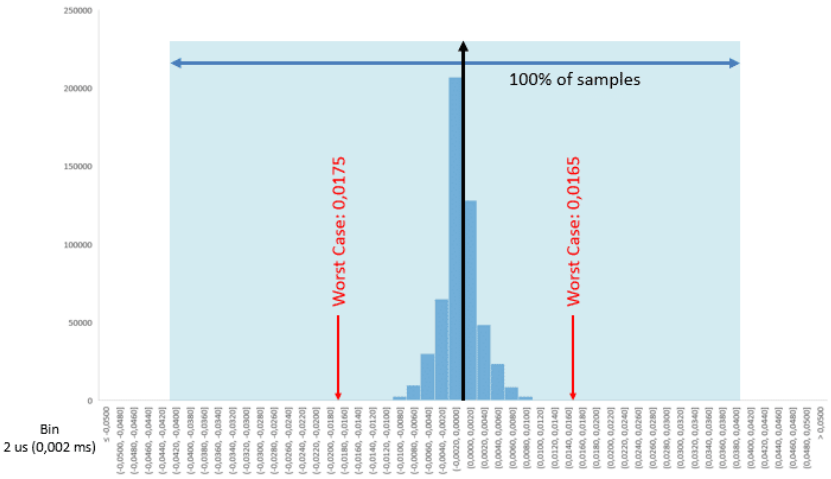

Figura 5: Histograma de amostras de erro medidas utilizando Xenomai.

Processamento Multicore Heterogêneo

O processamento Multicore Heterogêneo (ou processamento multicore assimétrico – HMP) é uma solução de hardware para o problema de tempo-real e determinismo no Linux. Processadores de aplicação como o NXP i.MX7 series, o NXP i.MX6SoloX e o futuro NXP i.MX8 series apresentam uma variedade de núcleos de processamento diferentes e com propósitos diferentes também. Se você considerar o i.MX7S, pode verificar que ele possui dois processadores, um Cortex-A7@800MHz trabalhando lado a lado de um Cortex-M4@200MHz.

A ideia por trás dessa arquitetura é a de que as partes de interfaces com usuário e a de conectividade de alta velocidade sejam implementadas em um sistema operacional de alta abstração, como o Linux, no núcleo Cortex-A. Enquanto isso, de forma independente e paralela, o Cortex-M executará um RTOS, como FreeRTOS. Os dois núcleos são capazes de compartilhar acesso à memória e a periféricos, permitindo flexibilidade e liberdade ao definir qual dos núcleos/SO será responsável por cada uma das interfaces de hardware existentes no chip. Veja a figura 6.

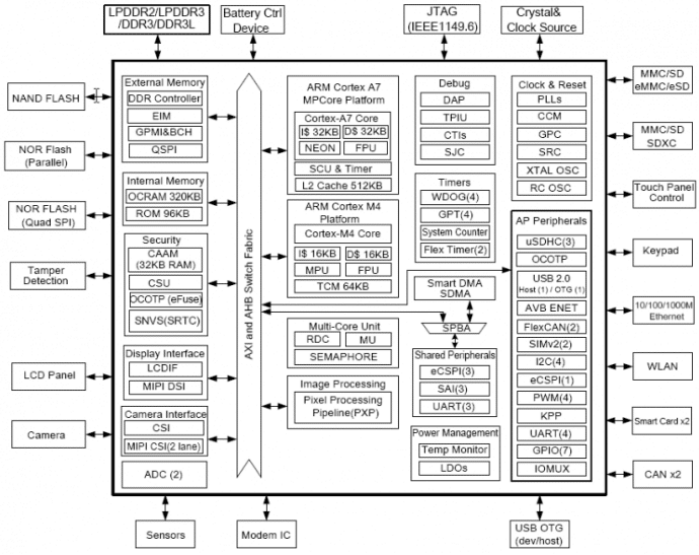

Figure 6: Diagrama de blocos do i.MX7. Fonte: http://cache.nxp.com/files/32bit/doc/data_sheet/IMX7SCEC.pdf?pspll=1

Algumas vantagens de usar a abordagem HMP são:

- Software Legado de microcontroladores, por ser reaproveitado;
- Update de firmware (Core M4) é simplificado uma vez que o firmware passa a ser um arquivo no file system do SO rodando no Cortex-A;
- Flexibilidade ao definir qual núcleo/SO gerencia cada interface. Uma vez que é uma definição de software, mudanças futuras são possíveis;

Mais informações sobre o Desenvolvimento de aplicações para processadores baseados em arquitetura HMP estão disponíveis nos artigos e webinar abaixo:

- A Balancing Robot Leveraging the Heterogeneous Asymmetric Architecture of i.MX 7 with FreeRTOS and Qt
- FreeRTOS on the Cortex-M4 of a Colibri iMX7.
- Desenvolvendo Sistemas Embarcados com o Colibri i.MX7, um CoM baseado no novo SoC i.MX7 da NXP

A Toradex, Antimicro e The Qt Company colaborativamente construíram um robô para demonstrar este conceito. O robô – apelidado de TAQ – é um pêndulo invertido e é controlado (interface e balanço) pelo computador em módulo Colibri iMX7. A interface de usuário é construída em Linux utilizando o framework QT, rodando no Cortex-A7 com Linux, e os motores, sensores e controle do balanço são executados no Cortex-M4. A comunicação entre os núcleos é usada para fazer o controle remoto do motor e disparar animações na interface do usuário quando o equilíbrio é desestabilizado. Veja o vídeo abaixo:

A Figura 7 apresenta os resultados do teste em um Colibri iMX7. Desta vez, a geração de onda quadrada é feita pelo FreeRTOS rodando no núcleo M4 do SoC i.MX7. Como esperado, os resultados do teste superaram todas as outras abordagens.

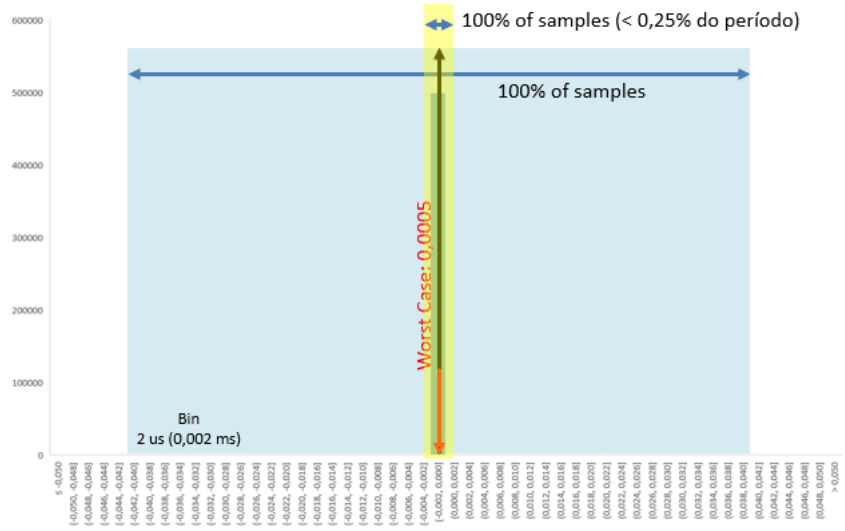

Figura 7: Histograma do erro medido utilizando processador HMP. A onda quadrada foi gerada no FreeRTOS que roda no núcleo M4.

Conclusão

Este artigo apresentou uma breve visão geral de algumas soluções disponíveis para desenvolver sistemas em tempo real em processadores de aplicação executando o Linux como o sistema operacional. Este é um ponto de partida para desenvolvedores que buscam usar microprocessadores e estão preocupados com o controle e determinismo e desempenho de controle de tempo real.

Apresentamos uma abordagem baseada em hardware usando os SoCs com processamento multicore heterogêneo (HMP) e duas abordagens baseadas em software, a saber: Linux-RT Patch e Xenomai. Os resultados apresentados não pretendem comparar sistemas operacionais ou técnicas em tempo real. Cada um deles tem pontos fortes e fracos e pode ser mais ou menos adequado dependendo do caso de uso.

A principal conclusão é que existem várias soluções viáveis para a utilização do Linux com processadores de aplicações confiáveis em tempo real.

Artigo escrito por: Guilherme Fernandes, Raul Muñoz, Leonardo Veiga e Brandon Shibley.

Guilherme Fernandes

http://www.toradex.com.br

Mestre em Engenharia Mecatrônica pela Escola de Engenharia de São Carlos (USP) atua como diretos da Toradex Brasil. trabalhando na implantação do escritório de vendas e suporte da empresa para o Brasil. Trabalhou 7 anos como gerente de engenharia de sistemas na área de automação industrial desenvolvendo mais de 300 projetos de máquinas para linhas de montagem e teste de produção para o setor de autopeças