ISEL | DEETC

Sistemas Embebidos 2

Automato

Semestre de Verão de 2010/2011

Autores:
31401 – Nuno Cancelo

33595 – Nuno Sousa

Indicie

Introdução

Este relatório segue a mesma estrutura elaborada no relatório da unidade curricular “Sistemas Embebidos I”.

Neste semestre foi retomado o estudo da arquitetura do módulo LPC-2106[1], sendo analisados profundamente alguns periféricos que não tinham sido na unidade curricular anterior.

Após o seu estudo, o capitulo seguinte abrange a comunicação com um módulo de hardware (enc28j60[2]) que implementa o protocolo ethernet permitindo comunicação com interfaces de rede ethernet.

Em seguida é revisto o capitulo das interrupções, dando a conhecer mais profundamente o seu modo de funcionamento.

Com o final desta breve revisão, é dado a conhecer o uIP[3] Uma plataforma que implementa o essencial do protocolo IP, permitindo realizar simples implementações de vários serviços utilizados em grandes aplicações.

Por fim, foi dado a conhecer o sistema operativo eCos[4]. Este sistema operativo introduz novos conceitos e potencialidades, numa arquitetura que até agora não era possível.

Ao longo do relatório verificar-se-á estes tópico e de que forma os mesmos serão ligados para realizar o projeto proposto. Será possível verificar igualmente as alterações que foram efetuadas de forma a migrar a nossa API para ser compatível e integrada na API do aCos.

1 http://www.olimex.com/dev/lpc-h40.html
2 http://www.olimex.com/dev/enc28j60-h.html
3 http://www.sics.se/~adam/old-uip/
4 http://ecos.sourceware.org/

Novos Periféricos Internos

A implementação de novos periféricos existentes no kit didático, são incluídos na camada de ligação que foi descrita no relatório anterior.

O primeiro novo periférico surge como alternativa à utilização de outro periférico.

SPI

Este periférico implementa o protocolo com o mesmo nome, protocolo o qual apesar de cobrir a mesma área de funcionamento do protocolo I2C[5], tem a desvantagem de gastar mais portos de GPIO, no entanto tem mais algumas vantagens como suportar ritmos superiores, mais periféricos e permitir comunicações em full-duplex.

A implementação deste periférico introduz uma nova alteração à forma como são declarados os ficheiros, criando uma separação sobre o que são funções para uso privado ao módulo e as funções que são de interface de uso publico.

SPI.c, SPI.h

Neste módulo foi definido uma estrutura que define um periférico SPI no LPC-2106, identificando devidamente os seu campos. Define igualmente a sua estrutura para relacionar as interrupções.

As macros definidas neste ficheiro header são as necessárias para o devido funcionamento deste módulo.

SPI.c, SPI_Public.h

Esta é a definição utilizada na interface publica. Este ficheiro header é utilizado para implementação interna como implementação externa por outros módulos.

Após a compilação deste módulo, o ficheiro SPI_Public.h é copiado para área onde estão os ficheiros headers públicos sob o nome de SPI.h.

Esta aplicação declara os seguintes enumerados para simplificação de interpretação e utilização pelo programador:

- SPI_ERRORS:
  - Definição dos erros de retorno das funções
- SPI_MODE
  - Modo de funcionamento do SPI
- SPI_ROLE
  - Indicação qual o papel que o este periférico irá ter.
    - Nota: Assume-se que o papel master, significa que o LPC é gestor da comunicação e que o papel de slave, significa que o

5 http://en.wikipedia.org/wiki/I%C2%B2C

LPC é gerido pelo periférico.

- SPI_STATUS
  - Possível status devolvidos pelo registo do SPI
- SPI_BYTE_SHIFT
  - Indicação o modo como os bits são enviados.

É definida uma estrutura para identificar o device SPI, denominada SPI_Device com a seguinte estrutura:

```
const void (*irqHandler)(void);      /*função de tratamento de interrupções*/
U32  clock;                  /*ritmo do sinal de relógio*/
U32  chipSelect;             /*define a identificação do periférico*/
U32  nbrbits:4;              /*número de bits de uma palavra*/
U32  mode:2;                 /*modo SPI (CPHA, CPOL). do tipo SPI_MODE*/
U32  role:1;                 /*qual o papel do periferico: 1- Master; 0- Slave SPI_ROLE*/
U32  started:1;       /*indicação se o periférico foi préviamente iniciada. Por
                      omissão o mesmo deve estar a 0. Não se garante o comportamento
                      caso na construção se coloque a 1.*/
U32  byteShift:1;            /*SPI_BYTE_SHIFT*/
U32   activeHigh;            /*Mode como o device é activado*/
```

Desta forma é possível ligar os enumerados descritos anteriormente com a sua aplicação na estrutura.

As funções de interface pública são as seguinte, sendo o nome evidente a seu funcionamento:

```
U8 SPI_init( pSPI_Device devices, U32 nbrDevices);
U8 SPI_start_device(pSPI_Device device);
void SPI_stop_device(pSPI_Device device);
U8 SPI_transfer(pSPI_Device device, U32 size, const U8 *tx_data, U8 *rx_buffer);
```

UART

A implementação deste periférico foi motivada pela utilização de uma biblioteca fornecida para permitir enviar mensagens para uma consola. Para a visualização das mensagens foram utilizados dois tipos de terminais: o gtkterm[6] e o minicom[7].

Foram utilizados ambos, um pela facilidade de configuração outro pela interface simples de utilização. A facilidade de visualização de informação enviada pelo Kit Didático numa janela de consola no PC, tornou mais interessante a aplicação assim como a efetuar mais algum debug às aplicações desenvolvidas.

UART.c, UART.h

Esta interface somente configura um conjunto de macros que permite o correto funcionamento do módulo.

6 http://sourceforge.net/projects/gtkterm/
7 http://en.wikipedia.org/wiki/Minicom

UART.c, UART_Public.h

A implementação da interface pública teve algumas escolhas para generalizar o seu uso. Essa generalização foi obtida após a analise dos campos e dos requisitos das definições da UART.

Antes de descrever as estruturas faz todo o sentido indicar todas os enumerados e macros utilizadas para sintetizar a sua utilização, dando algum suporte ao utilizador.

Os enumerados implementados são os seguintes:

- UART_INTERRUPT_IDENTIFICATION
  - Indica o tipo de interrupção
- UART_RX_TRIGGER_LEVEL
  - Indicação do tipo de analise de sinal
- UART_WORD_LENGTH_SELECT
  - Indicação do tamanho da palavra
- UART_PARITY_SELECT
  - Indicação do tipo de paridade
- UART_ERRORS
  - Indica o tipo de erros
- UART_STOP_BITS
  - Indica o numero de bits de paragem

Após esta descrição é fácil associar estes enumerados às estruturas definidas:

```
typedef struct{
 pLPC_UART  uartAddr;
 U32     baudrate;
 U32     bits:2;   /* Should use UART_WORD_LENGTH_SELECT enum */
 U32     parity:2;  /* Should use UART_PARITY_SELECT enum */
 U32     stopbits:1; /* Should use UART_STOP_BITS enum */
 U32     started:1; /* should be initiaded with 0 */
}Uart,*pUart;
```

Existe ainda a estrutura de LPC_UART, que tornou-se genérica de forma a poder ser utilizada por ambas as uarts que constam no módulo LPC. Dada a extensão da definição da estrutura e da sua apresentação não ser relevante para o contexto a mesma não é exposta neste artigo, podendo ser analisada ao detalhe no referido ficheiro header.

Assim sendo fica a faltar indicar quais são as funções que pertencem à API pública:

```
U32 UART_init(pUart uart);
U32 UART_write(pUart uart, const U8 *block, U32 size);
U32 UART_read(pUart uart,U8 *block, U32 size);
```

```
U32 UART_receiver_data_ready(pUart uart);
```

Interrupções

A implementação das interrupções vetorizadas, assim como as interrupções externa haviam sido efetuadas no semestre passado e como tal usufruímos da nossa implementação prévia para avançarmos no conteúdo programático.

Novos Periféricos Externos

O periférico que vamos descrever recai no âmbito do camada de periférico, uma vez que é um periférico externo que utiliza a API para comunicar com o kit didático.

Enc28J60

A implementação deste periférico é dividida em duas partes, uma que se dedica em utilizar o protocolo SPI para a comunicação entre o kit didático e o periférico externo e ainda uma parte que se dedica a enviar e receber informação sendo considerado a API pública.

ENC28J60.c, ENC28J60.h

A definição deste módulo é demasiado extensa para demonstração neste relatório, no entanto referimos que este módulo implementa toda a interface do Microchip Enc28j60[8] (facto pelo qual é extenso) tendo um conjunto de funções para escrever e ler registos. Estas funções utilização o periférico interno SPI do kit didático como plataforma de comunicação por entre os módulos de hardware.

Este módulo é utilizado pelo módulo Ethernet que vamos analisar de seguida.

Ethernet.c, Ethernet.h

Este módulo é implementado de forma bastante simples, sendo as funções fornecidas quase uma cópia do que foi implementado no SPI.

Este módulo é caracterizado por ter dois enumeradores, uma estrutura e três funções.

Como se pode verificar nos seguintes enumerados:

- ETHERNET_ERRORS
  - Indicação do tipo de erros de retorno possíveis
- ETHERNET_COM_SYSTEM
  - Indicação se a comunicação é full-duplex ou half-duplex

Tem a seguinte estrutura de dados:

```
typedef struct{
```

8 http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en022889

```
  SPI_Device ethernetDevice;
  ETHERNET_COM_SYSTEM duplex;
  U8 mac[MAC_NBR_BYTES] ;
}ETHERNET_Device,*pETHERNET_Device;
```

E como não podia deixa de ser, seguem as funções para utilização pública:

```
U8  Ethernet_init(pETHERNET_Device ethernetDevice);
U32 Ethernet_receive(U8* buffer, U32 buffer_size);
U8  Ethernet_send(U8* packet, U16 packet_size);
```

eCos (Parte 1)

Antes de passarmos à utilização do uIP, vamos falar do eCos de forma a não tornarmos o relatório monótono ao repetir muita informação.

O eCos é um sistema operativo(S.O) para sistemas embebidos de baixa capacidade que introduz ao sistema embebido conceitos como threads, sincronismo, escalonador, programação concorrente, etc.

Tal como qualquer S.O. o eCos funciona numa camada de abstração entre uma camada de hardware e a uma camada de aplicação, e como tal não faz sentido haver "forma" de uma aplicação aceder diretamente ao hardware sem que o S.O. a controle.

Desta forma houve a necessidade de reformular a nossa API de forma a ser compatível com o eCos.

De forma otimizar implementações, optámos por utilizar as funcionalidades fornecidas pelo S.O. e ainda permitir que algumas das nossa implementações/sintaxes fossem utilizadas através de funções do eCos, permitindo utilizar periféricos internos que o S.O. por omissão despreza.

Alterações à API

As alterações necessárias foram separadas em vários grupos, de forma a separar as responsabilidades e evitando repetição de código.

Esta conclusão foi chegada após analisar a documentação do eCos e verificar que potencialidades poderíamos acrescer ao sistema prejudicar o S.O. e por outro lado tirar partido do que já tinha sido implementa no S.O..

LPC

A primeira alteração que foi efetuada foi no sistema de tipos. Uma vez que o eCos usa uma nomenclatura diferente, houve a necessidade de criar uma conversão de tipos. Essa alteração verifica-se no ficheiro de header TYPES.h.

A segunda alteração foi no sentido de mapear as nossas estruturas nos endereços definidos pelo S.O., criando desta forma mais uma camada de abstração permitindo extensibilidade por entre modelos semelhantes de hardware, assim como upgrade/alterações do S.O.. Esta alteração pode ser verificada no ficheiro de header LPC2106.h.

De forma a criar uma camada de abstração, independente do hardware, foi criada um ficheiro de header LPC21XX.h que incluí o ficheiro de header respetivo para a sua aplicação (no caso LPC2106.h).

Por fim, verificou-se que não haveria necessidade de alterações ao módulo SCB.h, uma vez que já se havia alterado o sistema de tipos e os mapeamentos de estruturas.

GPIO

Uma vez que existe um plataforma de comunicação no eCos para falar com camada de hardware, este módulo deixaria de fazer sentido se não fosse a forma "intragável" como a sintaxe está definida.

Desta forma, optámos por mapear as nossas funções com a sintaxe utilizada pela API do eCos. Como forma de não criar latência em tempo de execução, optámos por ter um pouco mais de pré-processamento em tempo de compilação e para o efeito utilizamos macros para realizar o dito mapeamento.

As macros definidas têm a seguinte assinatura (respeitando a nomenclatura utilizada previamente como função):

```
#define GPIO_CLEAR(clear_pin)  (HAL_WRITE_UINT32((pGPIO)->IOCLR, 1 << clear_pin))
#define GPIO_SET(set_pin)      (HAL_WRITE_UINT32((pGPIO)->IOSET, 1 << set_pin))
#define GPIO_WRITE(pin_mask,value)    (HAL_WRITE_UINT32( pin_mask, value ))
#define GPIO_READ(pin_mask,value)     (HAL_READ_UINT32( pin_mask, value ))
#define GPIO_SET_DIRECTION(value,direction)

       {cyg_uint32 iodir;\
       HAL_READ_UINT32((pGPIO)->IODIR, iodir); \
       HAL_WRITE_UINT32(pGPIO)->IODIR, (direction)?(iodir |value):(iodir & (~value)))}
#define  GPIO_INIT_PINSEL0(mask)

       {cyg_uint32 pinsel;    \
       HAL_READ_UINT32(PINSEL0, pinsel);\
       HAL_WRITE_UINT32(PINSEL0, pinsel & mask);}
#define  GPIO_INIT_PINSEL1(mask)

       {cyg_uint32 pinsel;    \
       HAL_READ_UINT32(PINSEL1, pinsel);\
       HAL_WRITE_UINT32(PINSEL1, pinsel & mask);}
```

Desta forma é possível utilizar a nossa API para as aplicações como para outro periféricos.

SPI

Apesar de utilizarmos utilizarmos implementação da interface programática do SPI no eCos, as estruturas que criamos permite alguma versatilidade na implementação de uma solução que utilize o SPI.

TIMER_LPC.c, TIMER_LPC.h

A decisão de incluir este módulo deveu-se ao facto do eCos só utilizar o Timer0 do LPC e dessa forma deixa-nos utilizar o Timer1 da forma como entendermos, dando-nos espaço de manobra.

A implementação é igual à do semestre passado, sendo que a unica

diferença foi corrigir as assinaturas das funções utilizadas do módulo GPIO. Como tal, não vamos tornar o relatório monotono repetindo a api novamente.

RTC

Tal como o módulo anterior, este modulo não sofreu qualquer alteração, pois os módulos que incluí já haviam sido alterados e trás a potencialidade de utilizar o RTC que o eCos não utiliza e que o kit trás por omissão.

uIP

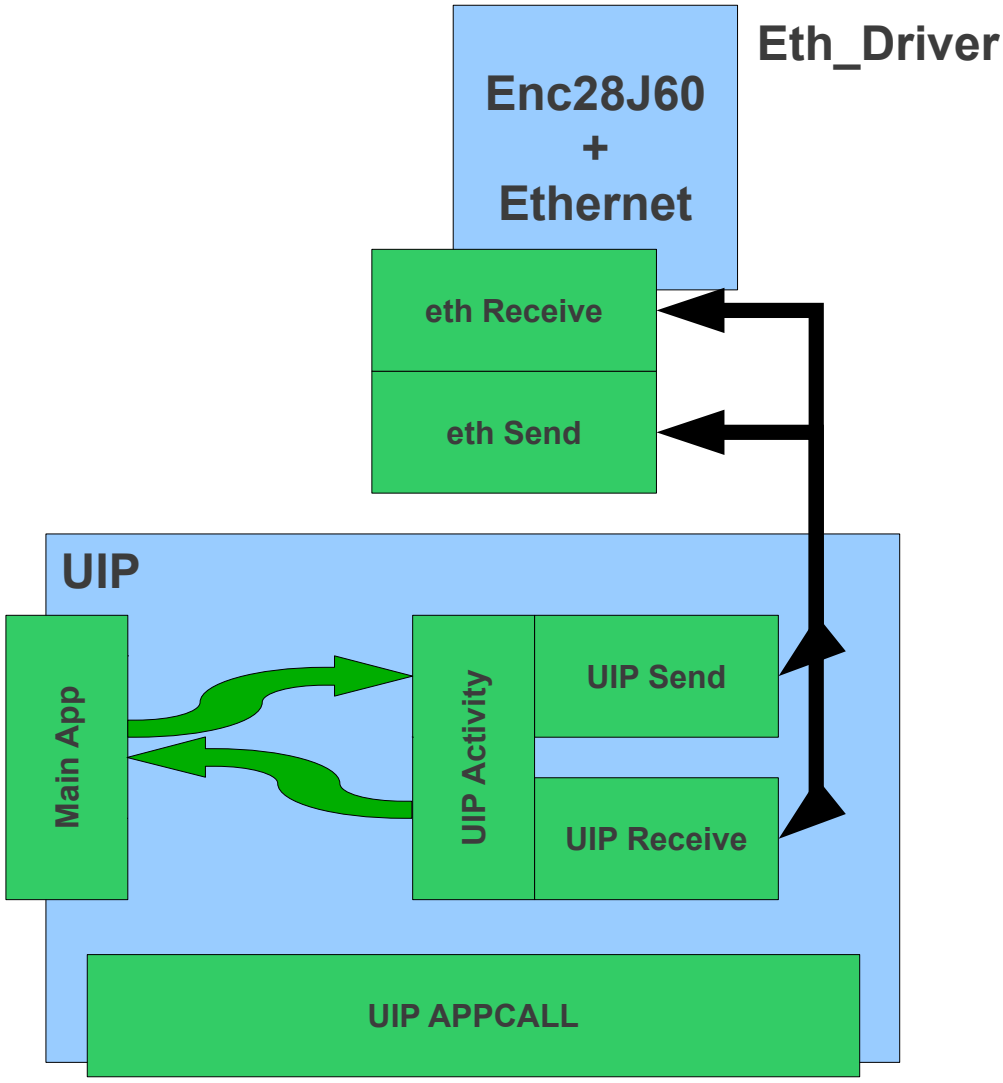
O uIP tráz a implementação do stack TCP/IP para sistemas embebidos a 8 bits e é uma implementação muito leve (em termos de tamanho) e com funcionalidades genéricas.

Esta biblioteca vem com diversos exemplos de utilização e um exemplo de implementação.

A partir do exemplo de implementação verificou-se o que já se sabia na teoria, ou seja, que era necessário ter quatro funções:

- Função de inicialização do periférico
- Função de Leitura do periférico
- Função de Escrita do periférico
- Função que devolve um tempo

Desta forma podemos esquematizar a interligação entre o uIP e o módulo de hardware Enc28j60:



Como se pode verificar as funções send e receive mapeam diretamente sobre as funções definidas sobre o ethernet para comunicação entre os módulos. Verifica-se igualmente que existem dois pontos de comunicação extra com o uIP.

O UIP APPCALL é utilizada para ligação estática com a aplicação que vai implementar, no nosso projecto, o servidor web.

O Main App é o módulo que irá gerir o fluxo de controlo do servidor web, ler, escrever, processar dados da página web.

Depois de se ter analisado o uIP, fica a faltar um último tema para ser relatado. Esse tema leva-nos de novo ao eCos onde poderemos falar já da integração entre o S.O., o uIP e a camada aplicacional.

eCos (Parte 2)

Na primeira parte do eCos foi referido alterações sobre que se teve que fazer para tornar compatível ambas as API's. Nesta secção vamos falar por alto algumas funcionalidades que iremos utilizar.

Como foi referido anteriormente o eCos trás novos conceitos para o sistema embebido, nomeadamente o conceito de Thread e seu controlo de fluxo.

Algo importante neste S.O. é a sequência de inicialização do sistema que é executado ainda antes do programa principal começar e imediatamente antes de o escalonador ser iniciado. Nesta sequencia de inicialização passa por alguns passos dos quais se destacam:

- Inicialização do hardware
- Invocação dos construtores dos modulos
- Inicialização das Threads, variáveis do programa, stack e controladores de fluxo das tarefas
- Inicialização do escalonador

A partir de agora o programa está apto para entrar em funcionamento.

Num ambiente concorrente, é necessário garantir que os dados estão estáveis na altura que são lidos ou exclusividade para quando os mesmos são escritos ninguém os está a ler.

Para controlar este fluxo de tarefas existem vários mecanismos de sincronismo conhecidos, tais como:

- Semáforo
- Mutex
- Conditional Variables
- Event Flags
- Message Box
- Interrupts

Não vamos estar a descrever todas as funcionalidades do S.O. ao pormenor, mas apenas referir os mecanismos utilizados.

Dada a estrutura da aplicação pretendida, vamos optar pela utilização das messages box, como forma de simplificação da solução, bem como otimizar a performance da solução.

Aplicação “Autómato”

Na interpretação do enunciado do projeto pode-se constatar trata-se de uma simples atividade que permite programar alguns portos para efetuar uma ação sobre outro porto.

Verifica-se igualmente um padrão nos requisitos solicitados nomeadamente na programação, pelo que conseguimos generalizar as seguintes propostas:

- Ativação Manual
  - Parâmetros: Saída
- Ativação Temporizada
  - Parâmetros: Saída, Data, Ciclo, Duração
- Ativação em Função da Entrada
  - Parâmetros: Entrada, Saída, Duração

Para a seguinte atividade

- Ativação Generalizada
  - Parâmetros: Entrada, Saída, Data, Ciclo, Duração

Pelo que podemos tirar as seguintes inferências:

- Ativação Manual
  - Entrada, Data, Ciclo, Duração estão a vazio
- Ativação em Função da Entrada
  - Data, Ciclo estão a vazio
- Ativação Temporizada
  - Entrada está a vazio
- Permite mais algumas combinações de Ativação, que devem ser contemplados no programa principal.

Os dois requisitos que faltam introduzem uma nova funcionalidade e uma particularidade que a utilização desta “API” neste sistema permite.

A utilização do RTC do LPC, permite-nos configurar o relógio do sistema e a utilização da generalização das Ativações como disposto anteriormente, resolve de forma imediata a consulta da programação

Desta forma podemos mostrar uma previsão da interface Web que o sistema terá e que pode ser analisada na Imagem 1: Interface Web

Trabalho Práctico de Sistemas Embebidos

Semestre de Verão de 2010/2011

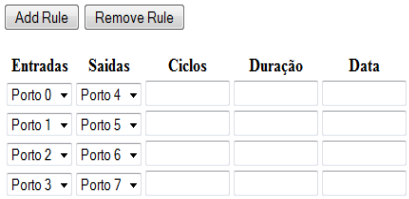
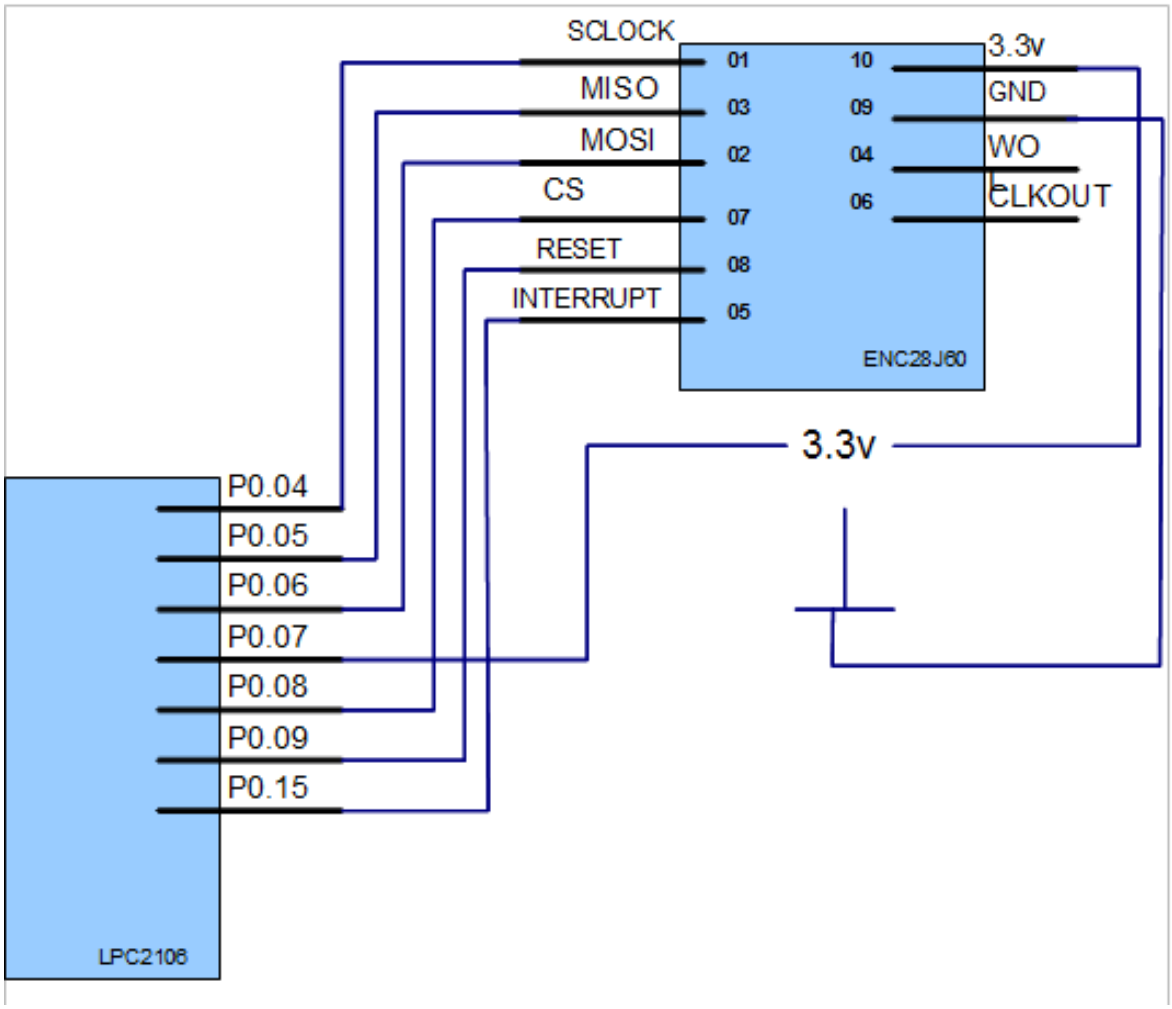


Imagem 1: Interface Web

Neste momento estamos aptos a mostrar um diagrama da forma sobre a qual os módulos existentes comunicam.

Em primeiro lugar vamos ligar o nosso kit didático ao módulo de rede:



Após o esquema de ligação podemos generalizar o esquema anterior num único módulo, que é indicado na Imagem 2: Diagrama de Blocos da Comunicação entre módulos como LPC2106+ENC28J60+JTAG, refletindo a ligação entre os módulos de hardware.

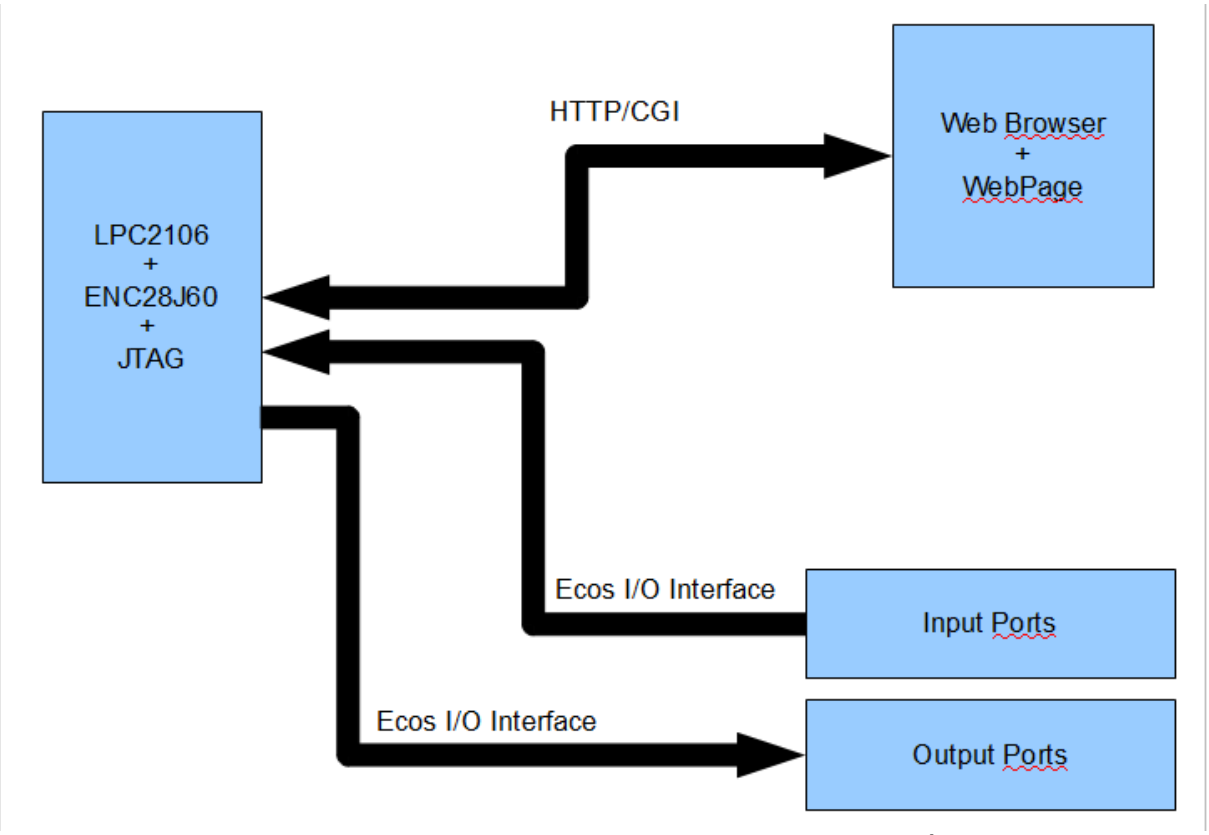


Imagem 2: Diagrama de Blocos da Comunicação entre módulos

Como se pode constatar no diagrama de blocos, o programa está divido nas seguintes secções:

- Secção 1:
  - WebBrowser + WebPage
    - Secção colocada do lado do cliente que através de um web browser consegue interpretar a página web, mostrar o seu conteúdo e devolver o pedido http com a resposta de um formulário em modo GET, de forma que o servidor possa processar as mensagens
- Secção 2:
  - Input/Output ports
    - Secção onde estarão associados portos a eventos a serem realizados.
- Secção 3:
  - LPC2106+ENC28J60+JTAG
    - Coração da aplicação onde estará configurada threads para tomar conta os eventos nos portos, assim como tomar conta dos pedidos http ou mesmo efetuar processamento de

mensagens.

Podemos então apresentar a Imagem 3: Diagrama de Blocos da Solução com o diagrama da solução, onde podemos verificar onde os módulos previamente apresentados foram encaixados na solução.

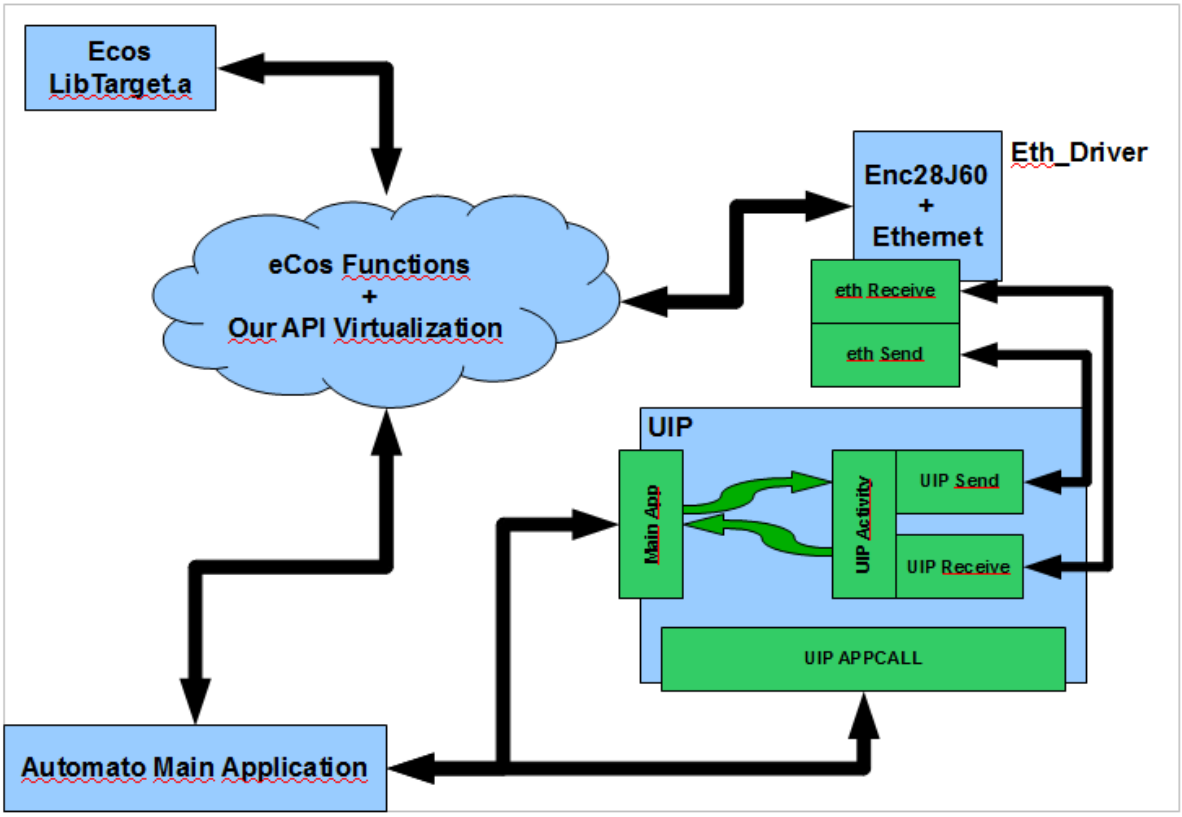


Imagem 3: Diagrama de Blocos da Solução

De um modo genérico podemos referir que temos uma thread que é responsável pelo serviço do módulo http (httpd). Este módulo é responsável pelo envio "envio de páginas web" e pelo processamento das mensagens de retorno.

Temos um conjunto de threads (uma ou mais, dependendo da configuração obtida) que irão executar a configuração programada ou mesmo acertar o relógio.

Todas as tarefas estarão em modo suspenso e dependendo da sua utilização serão "acordadas" ou "adormecidas". Os pedidos de processamento serão colocados numa mail box, onde a threads depois iram receber as suas instruções de execução.

Existe uma zona de memoria partilhada onde estarão as estruturas onde será guardada a informação quando é processados os pedidos do httpd e lida no contexto de execução de uma das funções.

Conceptualização da Implementação

Dada a estrutura estabelecida, é possível desde já concretizar alguns pontos.

Após análise do portos do kit, constatou-se que existem cerca de 8 portos disponíveis que podemos utilizar para a implementação do projeto.

Tendo esta limitação em mente, o próximo passo é definir um conjunto de variáveis que vão ajudar a complementar a solução, tornando-a mais eficiente e rápida.

Na primeira etapa deste passo, será alocar espaço para as Threads que vão ficar reservadas ao processamento de ações definidas sobre estes portos. Assim sendo estão alocadas numa estrutura de dados (potencialmente em array, para maior eficiência) o espaço para 8 Threads e noutra estrutura de dados, reservado o espaço para ser utilizado como stack da Thread. O tamanho do stack será obtido após a analise da função que irá realizar a ação sobre o(s) porto(s) definidos. Aproveitando o facto de se estar a trabalhar as Threads, aloca-se outra estrutura de dados para guardar o *handler* para cada Thread, desta forma pode-se aceder ao *handler* de cada uma delas.

As Threads no caso de ausência de atividade estarão em modo idle, de forma que possam ser retomadas quando existir ação para ser realizada.

Nota: Esta solução têm o pressuposto de que a estrutura de dados com tamanho 8, implica que cada indicie é representação em software de toda a ação imposta sobre um porto representado por esse indicie.

Existe no sistema uma estrutura que realiza o mapeamento entre o indicie e respetivo porto de input/output de hardware, criando mais uma camada de abstração entre o software e o hardware.

A segunda etapa passa por criar um meio de sincronismo, visto que para estas estruturas de dados funcionarem, é necessário garantir paralelismo no processamento e atomicidade no acesso aos dados. Foram analisados alguns modelos de sincronismo, mas a opção caiu sobre a *mailbox*, visto que a sua semântica é semelhante à do semáforo e pelo facto de ter uma fila de dados para processamento de pedidos, tornando simples o acesso.

Num segundo passo, pretende-se implementar a função void cyg_user_start(void);.

Na implementação vão ser inicializados:

- As Threads
  - Para os portos Input/Output
  - Para o servidor http
- As MailBox
- O mapeamento dos handlers
- Iniciação do RTC, para utilizar o relógio

- Iniciação dos Portos Input/Output
- Iniciação dos periféricos, neste caso somente o ENC28J60

Numa terceira etapa, procedem-se à concretização do método main, que simplesmente terá um ciclo infinito (com o uso da diretiva cyg_thread_delay, para evitar o uso intensivo), a efetuar uma chamada a um método que irá verificar se existem pedidos http, recolher os dados e enviar uma notificação para a mailbox.

Num terceiro método é implementado a lógica de:

- Verificar o tipo de pedido. Se for programado ou dependente de uma entrada, será necessário que a Thread continue a ser executada.
- Executa o pedido solicitado, alterando no fim o estado do pedido
- Acesso a dados partilhados em exclusividade
- Colocar as threads em modo “sleep” quando finalizar o seu processamento.

Na quarta e última etapa, realiza-se a implementação do lado do cliente, em que o mesmo recebe uma página em formato html, com a configuração que está no sistema, com a indicação de que o pedido está ou não ativo. Recebe também um relógio que poderá atualizar, este pedido terá um processamento diferente dos que requerem manipulação de portos input/output.

Considerações Finais

A realização deste projeto teve duas plataformas completamente distintas. Uma plataforma de implementação de funcionalidades, seguindo a estrutura do código do semestre passado, utilizando as mesmas ferramentas e tendo somente a compreensão do problema em mão para solucionar. Outra plataforma de testes, analises, implementação, configuração de alternativas para desenvolvimento da solução.

Esta alternativa, apesar de facilitar o desenvolvimento, não facilitava a implementação, uma vez que não separava devidamente os módulos da forma desejada, o que estava predisposto a causar erros e problemas.

Outra questão de dificuldade cinge-se na transposição para o sistema eCos, sistema que trás uma sintaxe bastante desagradável e difícil de utilizar (principalmente nos exames).

De modo geral a cadeira foi entusiasta e aprendemos imenso, apesar de acharmos que haveria alterações que poderiam ser realizadas, que será tema de outro documento. Porém existe algo que não concordamos, que é o facto de fornecer código fonte. Não cremos que seja pedagógico a utilização código emprestado, partilhamos a opinião que é mais benéfico dar bibliotecas fechadas com API pública e os alunos tentarem implementar as suas soluções (eventualmente seguindo as instruções de um pseudo-código).

Agradecimentos

Correndo o risco de sermos repetitivos, vamos parafrasear os agradecimentos do relatório anterior.

Não queremos terminar este relatório sem agradecer o auxílio, esforço e dedicação de um elemento sempre presente e que raramente reconhecido.

Este elemento durante todo o semestre nos auxiliou e foi nos exigindo qualidade e dedicação num pedaço do mundo com muitas potencialidades.

Já no passado a cooperação dele foi essencial no nosso desenvolvimento quer como alunos quer como profissionais, pois a sua atitude pró-activa e altruísta leva uma motivação extra na nossa parte.

Por tudo que já passou e pelo que há de vir, obrigado Prof. Pedro Sampaio.