UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

TRABALHO DE CONCLUSÃO DE CURSO

DESENVOLVIMENTO DE EXPERIMENTOS PARA O ENSINO DE SISTEMAS EMBARCADOS

ELVYS RAPOSO PONTES

Campina Grande – PB

Outubro de 2017

Elvys Raposo Pontes

DESENVOLVIMENTO DE EXPERIMENTOS PARA O ENSINO DE SISTEMAS EMBARCADOS

Trabalho de conclusão de curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Bacharel em Engenharia Elétrica.

Área de concentração: Eletrônica

Orientador:

Professor Dr. Gutemberg Gonçalves dos Santos Júnior

Campina Grande – PB

Outubro de 2017

Elvys Raposo Pontes

DESENVOLVIMENTO DE EXPERIMENTOS PARA O ENSINO DE SISTEMAS EMBARCADOS

Trabalho de conclusão de curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Bacharel em Engenharia Elétrica.

Aprovado em _____/_____/_____

Professor Dr. Marcos Ricardo Alcântara Morais

Universidade Federal de Campina Grande

Avaliador

Professor Dr. Gutemberg Gonçalves dos Santos Júnior

Universidade Federal de Campina Grande

Orientador

Dedico este trabalho à minha família

AGRADECIMENTOS

Em primeiro lugar a Deus, por ser minha fonte de sabedoria e por ter me dado forças para o término desse curso.

À minha mãe Jacira de Oliveira Raposo e ao meu pai Ernane Sales Moreira Pontes pelo apoio dado em todas as minhas escolhas e por sempre me indicar o melhor caminho a ser tomado.

Ao meu irmão Mário Raposo Santos, por sempre me apoiar e me recordar que um dia fui uma criança cheia de sonhos, e que agora estou concretizando um deles.

À minha namorada Maria Gabrielly de Oliveira Pereira, futura esposa, pela companhia e força que sempre me foi passada, compreendendo minhas ausências e incentivando meu sucesso profissional.

Aos amigos de infância e aos amigos feitos durante a graduação, obrigado por entender minhas ausências por ter que me dedicar aos estudos e principalmente por acreditarem em mim em todas as situações.

Aos amigos que fiz durante o intercâmbio nos Estados Unidos, que se tornaram minha família durante um das experiências mais importantes da minha vida. Agradeço em especial à Célito Rafael, Gustavo Mazon e Tiago Xavier, moradores do D-315.

Agradeço ao professor Gutemberg Gonçalves dos Santos Júnior, exemplo de profissional, por todo o conhecimento que me transmitiu como orientador do TCC e do PEM.

Aos funcionários do Departamento de Engenharia Elétrica, particularmente, Adail Paz e Tchaikowisky Oliveira, obrigado de coração pela atenção de sempre e a prontidão em ajudar.

Enfim, agradeço a todos que de alguma forma, passaram pela minha vida e contribuíram para a construção de quem sou hoje.

RESUMO

Um sistema embarcado é um sistema microprocessado no qual o computador é completamente encapsulado ou dedicado ao dispositivo ou sistema que ele controla. Diferente de computadores de propósito geral, como o computador pessoal, um sistema embarcado realiza um conjunto de tarefas predefinidas, geralmente com requisitos específicos. Já que o sistema é dedicado a tarefas predeterminadas, por meio da engenharia pode-se otimizar o projeto reduzindo tamanho, recursos computacionais e custo de produto. Este trabalho apresenta experimentos para o desenvolvimento da disciplina de Sistemas Embarcados para a graduação de Engenharia Elétrica da UFCG, utilizando microcontroladores para o ensino prático da disciplina.

Palavras-chave: Sistemas embarcados; Graduação de Engenharia Elétrica; Microcontroladores.

ABSTRACT

An embedded system is a microprocessor system where the computer is completely encapsulated or dedicated to the device or system it controls. Unlike general-purpose computers, such as the personal computer, an embedded system performs a set of predefined tasks, usually specific requirements. Since the system is dedicated to predetermined tasks, through engineering one can optimize the design by reducing size, computational resources and product cost. This work presents experiments for the development of the Embedded Systems course for the graduation of Electrical Engineering at the UFCG, using microcontrollers for the practical teaching of the discipline.

Keywords: Embedded Systems; Electrical Engineering Degree; Microcontroller.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

PEM	Programa de Excelência a Microeletrônica
ABS	Anti-lock Braking System
PC	Personal Computer
SE	Sistema Embarcado
ULA	Unidade Lógica Aritmética
UC	Unidade de Controle
USB	Universal Serial Bus
SoC	System on Chip
ARM	Advanced RISC Machine
RISC	Reduced Instruction Set Computer
CISC	Complex Instruction Set Computer
CPU	Central Processing Unit
CPSR	Current Processor Status Register
FIQ	Fast Interrupt Request
GPIO	General-purpose input/output
TI	Texas Instruments
DSP	Digital Signal Processor
GNU	GNU's Not Unix
GCC	GNU Compiler Collection
IDE	Integrated Development Enviroment
LED	Light-emmiting diode
RGB	Red Green Blue

SUMÁRIO

INTRODUÇÃO

Nesta seção são apresentados a delimitação do estudo realizado, os objetivos da pesquisa, bem como os elementos introdutórios do trabalho abordado.

APRESENTAÇÃO DO TRABALHO

Este trabalho de conclusão de curso se enquadra na linha de pesquisa de Sistemas Embarcados que engloba as áreas de Eletrônica e Controle e Automação disponibilizadas pelo Departamento de Engenharia Elétrica – DEE, da Universidade Federal de Campina Grande – UFCG.

O trabalho teve como objetivo iniciar um estudo para o desenvolvimento de uma nova disciplina na grade curricular do curso de Engenharia Elétrica. Para isso, foram propostos alguns experimentos para facilitar o ensino e a atividade prática da disciplina, assim como propor o uso de alguns microcontroladores específicos para o curso.

MOTIVAÇÃO

Um sistema embarcado é um sistema microprocessado no qual o computador é completamente encapsulado ou dedicado ao dispositivo ou sistema que ele controla. Diferente de computadores de propósito geral, como o computador pessoal, um sistema embarcado realiza um conjunto de tarefas predefinidas, geralmente com requisitos específicos. Já que o sistema é dedicado a tarefas predeterminadas, por meio da engenharia pode-se otimizar o projeto reduzindo tamanho, recursos computacionais e custo de produto.

Sabe-se que os sistemas embarcados revolucionam o nosso meio modificando a todo instante a vida das pessoas e é simples de perceber, basta olhar ao nosso redor e notar que eles estão em quase todos os lugares e que suas aplicações alavancaram o desenvolvimento tecnológico de todas as áreas de conhecimento humano. Diversos exemplos podem ser dados do uso de um sistema embarcado, como um sistema salva vidas em marcapassos, ou um sistema que garante a segurança dos transportes em computadores de aviação e freios ABS. Outro exemplo é

servir para aproximar as pessoas por meio de satélites e equipamentos de telecomunicações ou também para agregar conforto ao nosso dia a dia com TVs, impressoras e reprodutores de mídias. Outro uso do sistema embarcado que podemos citar é sua utilização para diversão presente nos consoles de *games*, seu uso em sistemas de rede elétrica, sistemas bélicos e em reatores nucleares. Enfim, os sistemas embarcados estão presentes na maioria dos eletrônicos.

Uma informação importante é que os sistemas embarcados estão cada vez mais baratos e acessíveis, demandam menor consumo de energia e, além de mais compactos, possuem maior poder de processamento. Com esse crescente poder de processamento, que é cada vez maior com o passar do tempo, o mundo em que vivemos será cada vez mais micro conectado, onde não só os computadores acessam a internet, mas também os objetos ao nosso redor.

Propõe-se então um estudo de Sistemas Embarcados durante a graduação dos futuros engenheiros eletricistas.

OBJETIVOS

Os objetivos do projeto proposto são:

- Elencar e apresentar os conceitos básicos para o desenvolvimento de um sistema embarcado utilizando microcontroladores;
- Desenvolver uma pesquisa a fim de compreender as diferenças entre arquitetura de processadores de microcontroladores e qual a influência dela para poder escolher um grupo de microcontroladores que serão utilizados na parte experimental;
- Propor e executar experimentos simples utilizando os microcontroladores escolhidos para uma introdução dos conhecimentos básicos.

METODOLOGIA

Esta seção destina-se a apresentar a metodologia utilizada no estudo, evidenciando a caracterização da pesquisa, o ambiente da pesquisa e os procedimentos metodológicos.

- Estudo da fundamentação teórica relativa ao tema;
- Escolha e aquisição do microcontrolador e do *software* para utilização na parte experimental;
- Criação de um banco de dados e ordenação pela dificuldade de possíveis experimentos;
- Execução dos experimentos selecionados;
- Análise dos resultados;
- Documentação dos experimentos concluídos e da análise;

FUNDAMENTAÇÃO TEÓRICA

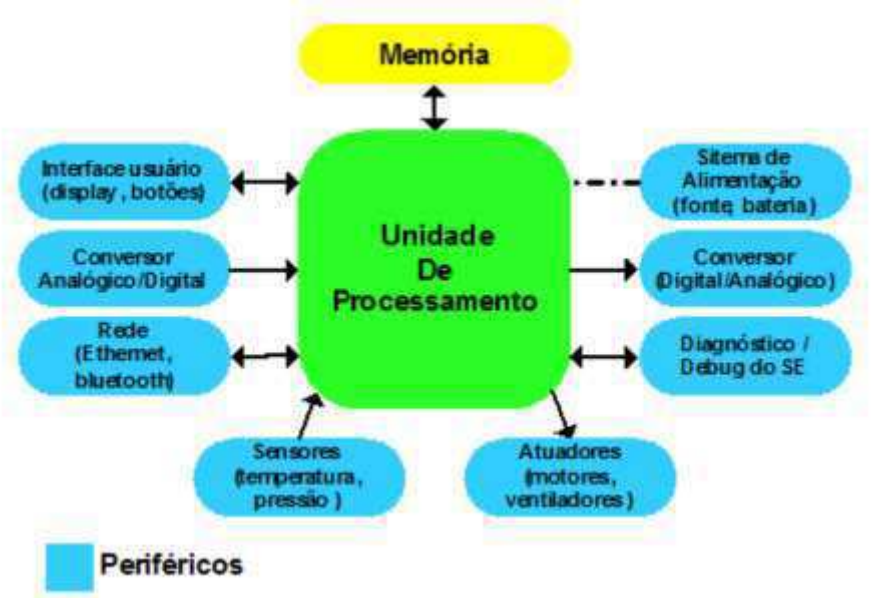
Estão descritas no decorrer deste tópico, resumidamente, algumas informações básicas sobre sistema embarcado e toda a tecnologia envolvida para a construção da mesma.

SISTEMAS EMBARCADOS

Sistemas embarcados estão relacionados ao uso de *hardware* (eletrônica) e *software* (instruções) incorporados em um dispositivo com um objetivo pré-definido. A diferença entre um sistema embarcado e um computador de propósito geral está justamente na objetividade. Computadores como PCs, *notebooks* e afins são máquinas multiobjetivo, ou seja, foram criadas e dimensionadas para atuar num domínio de funções muito grande. Já os sistemas embarcados ou SEs possuem dimensionamento de recursos direcionado a um domínio de objetivos bem menor, ou mesmo singular.

Destrinchando um projeto de SE, normalmente encontramos um subdivisão clara, que corresponde à unidade de processamento, memória e periféricos. Como visto na Figura 1.

Figura 1 - Exemplo de modelo de subdivisão de um sistema embarcado



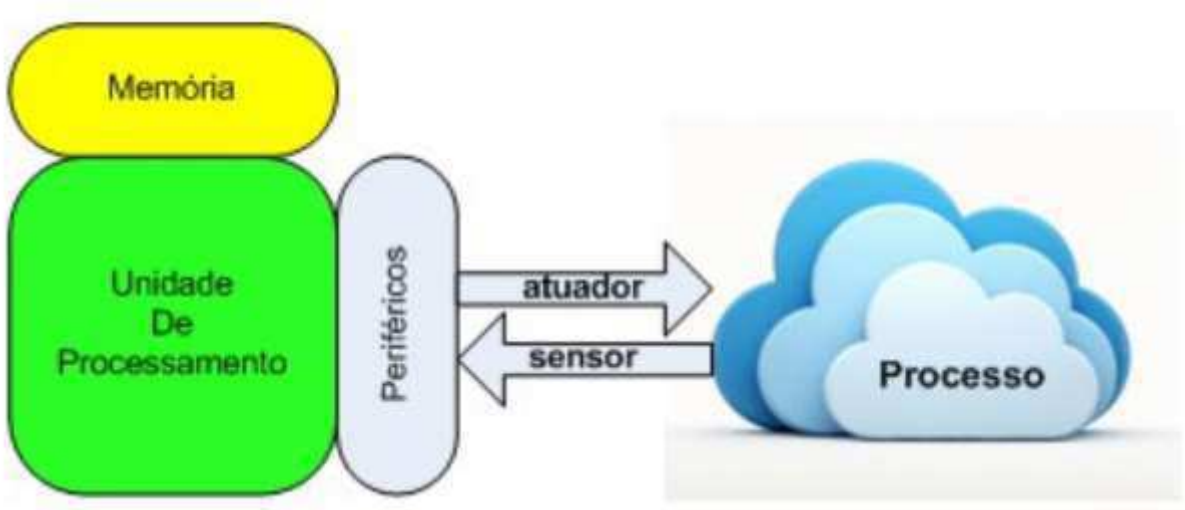
A **unidade de processamento** executa as instruções (*software/firmware*) responsáveis por realizar cálculos, tomar decisões e tratar eventos (como aquele do botão do elevador). Possui normalmente a arquitetura elementar clássica de um processador de computador convencional, como a unidade lógica/Aritmética (ULA), unidade de controle (UC), registradores, etc.

A **memória** armazena dados e instruções relacionados às operações da unidade de processamento. As instruções e dados podem dividir a mesma memória, como nos PCs (arquitetura Von Neumann) ou separados em memórias distintas (arquitetura Harvard), sendo a segunda a mais comum em SEs.

Os **periféricos** são as interfaces da unidade de processamento com o mundo externo, trazendo ou enviando informações para ele. Um exemplo de um periférico seria um conversor analógico/digital acoplado a um sensor térmico que converte a temperatura de um ambiente em números binários para que a unidade de processamento consiga interpretar e processar a informação.

O objetivo de um sistema embarcado é o de controlar processos, em outras palavras, atuar sobre um problema. Um processo pode ir de um simples acender e apagar de lâmpadas automatizado, até gerenciamento autônomo de um avião (piloto automático). Isso é feito por intermédio dos periféricos, que são escalados e dimensionados com base no problema alvo.

Figura 2 - Periféricos atuadores e sensores



Duas categorias de periféricos se destacam em um SE, os sensores e os atuadores.

Sensores são responsáveis em adquirir informação do processo a ser controlado. Essas informações são primordiais para a unidade de processamento, pois com base nelas decisões podem ser tomadas. Bons sensores devem fornecer informação confiável e não promover alterações no processo alvo. Isso significa que um sensor não pode mudar os valores da grandeza física do qual é responsável por medir, como por exemplo diminuir a velocidade de um motor em monitoramento. Isso na prática pode difícil de conseguir dependendo da tecnologia do sensor (contato mecânico, por exemplo). Exemplos desses tipos de periféricos são sensores de temperatura (termistores), pressão (piezos), contato (chaves mecânicas), toque (*touchscreen*), distância (sonar/infravermelho), movimento (acelerômetros), óticos (câmeras), etc. Esses são periféricos que enviam informação do processo para o SE.

Atuadores proporcionam ao SE a habilidade de intervir no meio onde atua. Como o próprio nome diz, são dispositivos que realizam ações que interferem no processo em controle, como motores, ventiladores, luzes, aquecedores, resfriadores, chaveadores, etc. Esses são periféricos que enviam informação do SE para o processo.

Normalmente a unidade de processamento toma a decisão de acionar os atuadores com base nas informações recebidas dos sensores, isso é conhecido como sistema em malha fechada ou **sistema realimentado**. Outra forma também usada para acionar os atuadores é com base no tempo, em um sistema conhecido como malha aberta, onde não há informações advindas do processo (não há sensores nele).

Conceitos Básicos de Desenvolvimento em Sistemas Embarcados

Devido às limitações das plataformas embarcadas é necessária uma plataforma mais robusta para o desenvolvimento das aplicações. Esta plataforma é chamada de desenvolvimento ou simplesmente *Host*. Já as plataformas onde as aplicações serão executadas são chamadas de Plataforma Alvo ou simplesmente *Target*.

Plataforma de Desenvolvimento – *Host*

Geralmente são computadores de propósito gerais (Computadores Pessoais) e que possuem grande capacidade de armazenamento e grande quantidade de processamento. Nelas temos disponíveis interfaces mais adequadas para o desenvolvedor e é onde serão executadas as ferramentas necessárias para o desenvolvimento de aplicações.

Plataforma Alvo – *Target*

É o produto propriamente dito. É nessa plataforma onde as aplicações serão executadas.

Geralmente são utilizadas plataformas de referência para o desenvolvimento até chegar ao produto final.

Tipos de Configuração *Host/Target*

É necessária uma interface entre a plataforma *host* e a plataforma *target*. Atualmente existem 3 configurações para estas interfaces que são mais comuns:

Linked Setup
Removable Storage Setup
Standalone Setup

Tipos de Configuração *Host/Target – Linked Setup*

É atualmente a configuração mais utilizada onde o host é conectado ao *Target* através de um cabo. Geralmente utiliza interface RS-232, USB ou *Ethernet* para comunicação com o *target*.

Neste tipo de configuração, temos a possibilidade da depuração do código "remotamente";

Atualmente, é utilizado este tipo de configuração para a gravação e depuração de *software* na plataforma alvo com o auxílio de JTAGs.

Tipos de Configuração *Host/Target – Removable Storage Setup*

Utiliza dispositivos removíveis para cópia da aplicação desenvolvida para a plataforma alvo.

Antigamente eram utilizadas placas de gravação contendo *sockets* para a memória a qual era utilizada na plataforma alvo.

Modelo quase obsoleto, pois este não permite a depuração da aplicação de forma eficaz.

Tipos de Configuração *Host/Target – Standalone Setup*

São utilizados em plataforma alvo bem mais complexas. Neste tipo de configuração as ferramentas de desenvolvimento são executadas na plataforma alvo, sendo assim, a plataforma de desenvolvimento é utilizada apenas para acessar a plataforma alvo.

Devido à evolução das plataformas embarcadas, este modelo já vem sendo utilizado.

MICROCONTROLADOR

Microcontrolador é um pequeno computador (SoC) em um único circuito integrado o qual contém um núcleo de processador, memória e periféricos programáveis de entrada e saída. A memória de programação pode ser RAM, NOR flash ou PROM a qual, muitas vezes, é incluída no chip. Os microcontroladores são concebidos para aplicações embarcadas, em contraste com os microprocessadores utilizados em computadores pessoais ou outras aplicações de uso geral.

Microcontroladores são usados em produtos e dispositivos automatizados, como os sistemas de controle de automóvel, dispositivos médicos implantáveis, controles remotos, máquinas de escritório, eletrodomésticos, ferramentas elétricas, brinquedos e outros sistemas embarcados. Ao reduzir o tamanho e o custo em comparação a um projeto que usa um dispositivo microprocessado, microcontroladores tornam-se econômicos para controlar digitalmente dispositivos e processos. Microcontroladores de sinal misto são comuns, integrando componentes analógicos necessários para controlar sistemas eletrônicos não-digitais.

O seu consumo de energia é relativamente baixo, normalmente na casa dos miliwatts e possui habilidade para entrar em modo de espera (*Sleep* ou *Wait*) aguardando por uma interrupção ou evento externo, como por exemplo o acionamento de uma tecla, ou um sinal que chega via uma interface de dados. O consumo destes microcontroladores em modo de espera pode chegar na casa dos nanowatts, tornando-os ideais para aplicações onde a exigência de baixo consumo de energia é um fator decisivo para o sucesso do projeto.

De forma diferente da programação para microprocessadores, que em geral contam com um sistema operacional e um BIOS, o programador ou projetista que desenvolve sistemas com microcontroladores geralmente cria todo programa que será executado pelo sistema ou pode usar um sistema operacional próprio para microcontroladores chamado de RTOS.

APLICAÇÕES

Microcontroladores são geralmente utilizados em automação e controle de produtos e periféricos, como sistemas de controle de motores automotivos, controles remotos, máquinas de escritório e residenciais, brinquedos, sistemas de supervisão, etc. Por reduzir o tamanho, custo e consumo de energia, e se comparados à forma de utilização de microprocessadores convencionais, aliados a facilidade de desenho de aplicações, juntamente com o seu baixo custo, os microcontroladores são uma alternativa eficiente para controlar muitos processos e aplicações.

Cerca de 50% dos microcontroladores vendidos são controladores "simples", outros 20% são processadores de sinais digitais (DSPs) mais especializados. Os microcontroladores podem ser encontrados em praticamente todos os dispositivos eletrônicos digitais que nos cercam: teclado do computador, dentro do monitor, disco rígido, relógio de pulso, rádio relógio, máquinas de lavar, forno de micro-ondas, telefone, etc. Você está certamente cercado de dezenas deles agora. Certamente eles foram tão ou mais importantes para a revolução dos produtos eletrônicos que os computadores. Eles permitiram a evolução de equipamentos que há anos não evoluíam, como os motores a combustão, que agora com o novo controle eletrônico podem funcionar com sistema bi-combustível e poluindo menos e as máquinas fotográficas, que migraram de processos químico/mecânico a circuitos com microcontroladores + Sensores Digitais + Memória.

ALGUNS DOS FABRICANTES DE MICROCONTROLADORES

Exemplo de fabricantes de microcontroladores.

- AMCC;
- Atmel;
- Cypress MicroSystems;
- Fujitsu;
- Holtek;
- Intel adquiriu em 2015 a empresa Altera;
- Microchip Technology;
- MIPS Technologies;
- NXP Semiconductors comprou a Freescale Semiconductor em 2015;
- NEC;
- Parallax, Inc.;
- Renesas Tech. Corp. adquiriu a empresa Synaptics em 2014;
- STMicroelectronics;
- Silicon Laboratories;
- Texas Instruments adquiriu a National Semiconductor em 2011;
- Western Design Center;
- ZiLOG.

ARQUITETURA ARM

Arquitetura ARM (primeiramente *Acorn RISC Machine*, posteriormente *Advanced RISC Machine*) é uma arquitetura de processador de 32 bits usada principalmente em sistemas embarcados. São processadores que visam a simplificação das instruções, com o intuito de atingir a máxima eficiência por ciclo, podendo realizar tarefas menores com ciclos

mais curtos, e uma maior ordenação das operações dentro do núcleo de processamento. Muito usada na indústria e na informática, seu desenvolvimento se deu visando obter o melhor desempenho possível, com a limitação de ser simples, ocupar pouca área e ter baixo consumo de energia.

Os processadores ARM são conhecidos pela sua versatilidade, pois possuem poucas instruções para programação. São encontrados em PDAs, telefones celulares, calculadoras, periféricos de computador, equipamentos POS e aplicações industriais.

O padrão RISC do processador permite que estes processadores tenham menos transístores que processadores CISC (x86). Essa abordagem reduz custos, liberação de calor e consumo de energia. Essas são características desejáveis para dispositivos portáteis, como *smartphones*, *laptops*, tablets e outros dispositivos embarcados. Uma estrutura mais simples facilita a criação de multi-core CPUs, o que impacta na redução de custos de produção. Os processadores ARM são 90% dos processadores embarcados RISC de 32 bits.

O processador ARM possui sete modos de operação que podem ser intercambiados através do *software*, interrupções externas e processamento de execuções. Normalmente as aplicações são executadas a nível de usuário. Enquanto o processador esta no modo usuário o programa sendo executado é incapaz de acessar alguns recursos protegidos do sistema ou mudar de modo.

Os outros modos além do modo usuário são denominados modos privilegiados. Eles tem acesso completo aos recursos do sistema e podem mudar de modo livremente.

Registradores: Os registradores podem ser utilizados para manipular dados de um byte, de meia palavra (16 bits) ou de uma palavra completa (32 bits). Quando instruções de um byte são utilizadas somente o byte menos significativo é utilizado. Quando instruções de meia palavra são utilizadas somente a palavra menos significativa é utilizada. Ao fazer referencia a algum destes registros de propósito específico deve-se sempre levar em consideração qual o modo de operação corrente. Um outro registrador importante é o CPSR (Current Processor Status Register), que carrega informações sobre o estado corrente do processador, inclusive o modo de execução atual.

PRINCIPAIS CARACTERÍSTICAS

Uma das características é a arquitetura *Load-Store*: as instruções somente processarão (soma, subtração, etc) valores que estiverem nos registradores e sempre armazenarão os resultados em algum registrador. Uma outra é de instruções fixas de 32 bits de largura (com exceção das instruções *Thumb* compactas de 16 bits) alinhadas em 4 bytes consecutivos da memória, com execução condicional, com poderosas instruções de carga e armazenamento de múltiplos registradores, capacidade de executar operações de deslocamento e na ULA com uma única instrução executada em um ciclo de *clock*.

Podemos citar também o formato de instruções de 3 endereços (isto é, os dois registradores operandos e o registrador de resultado são independentemente especificados) e possui 16 registradores de 32 bits para uso geral.

A manipulação de periféricos de I/O como dispositivos mapeados na memória com suporte à interrupções é uma outra especificação junto com um conjunto de instruções aberto a extensões através de co-processador, incluindo a adição de novos registradores e tipos de dados ao modelo do programador.

Outras características importantes são: *Pipelines* de 3 a 15 estágios; baixo Consumo de energia; tamanho do núcleo reduzido;

As interrupções são definidas por configurações programáveis. FIQ: *Fast Interrupt Request*, maior prioridade. IRQ: *Vectorred Interupt Request*, intermediária (0 à 15). Não - *Vectored Interupt Request*, menor prioridade. As prioridades das interrupções dos diversos dispositivos são ajustadas dinamicamente

Os blocos de conexão de pinos permitem selecionar os pinos do microcontrolador que possuem mais que uma função. Os registros de configuração controlam os multiplexadores para permitir a conexão entre os pinos e os periféricos no chip e os periféricos devem ser conectados a pinos apropriados antes de serem habilitados e antes de qualquer interrupção relacionada seja ativada. Sabendo que a ativação de qualquer função periférica que não é mapeada para um pino relacionado devem ser consideradas indefinidas.

Outra informação importante são os propósitos gerais *Paralell* I/O e *Fast* I/O: pinos que não são conectados a específicas funções periféricas são controlados pelos registros do GPIO. Esses pinos podem ser configurados dinamicamente como entrada ou saída. Separar os registros permite configurar ou limpar qualquer número de saídas simultaneamente, o valor do registro de

saída pode ser lido novamente, bem como os estados atuais das portas. No *paralell* e *Fast* I/O há controle de direção individual dos bits e todas as I/O viram input no reset.

TIVA-C

Tiva-C (ou TM4C) *Launchpads* é uma plataforma e prototipagem eletrônica de microcontroladores criado pela Texas Instruments. As placas são do tamanho aproximado de um cartão de crédito. São equipadas com um microcontrolador ARM Cortex-M4F CPU de 32 bits operando a 80 a 120 MHz, fabricado pela Texas Instruments. O TM4C Series TM4C123G *LaunchPad* é um upgrade da TI da Stellaris *LaunchPad* adicionando opções de suporte de PWMs para controle de movimento e funcionalidade de *host* USB.

São equipados com 40 ou 80 pinos que têm multifunções, ou seja, podem ser configurados como entradas ou saídas, digitais ou analógicas ou outras funções, permitindo uma grande variedade de aplicações. Seus pinos possui o padrão de 3,3 V.

Comparando com um Arduino, que é bastante conhecido, temos que o *clock* é de 80 ou 120 MHz (na versão básica), o que os torna de 5 a 7 vezes mais rápido do os microcontroladores do Arduino UNO que são 16 MHz ATMEGA328P. Como acontece com qualquer Cortex M4, a CPU tem compatibilidade com instruções DSP (*Digital Signal Processor*), com algumas limitações. Neste caso, ele consegue realizar processamento de sinal, por exemplo, a amostragem da voz humana com uma boa qualidade, capaz de ser processada em Matlab.

A linguagem de programação pode ser *Assembly*, mas também pode ser usada a linguagem C com um compilador fornecido pela Texas Instruments, pode ser usado o compilador GCC (GNU *Compiler Collection*) ou através do projeto Energia que é uma variante livre do ambiente de desenvolvimento integrado (IDE) do Arduino. Um programa *bootloader* é pré-instalado permitindo que a placa possa ser reprogramada por uma porta padrão USB 2.0 (que não requer *hardware* especial).

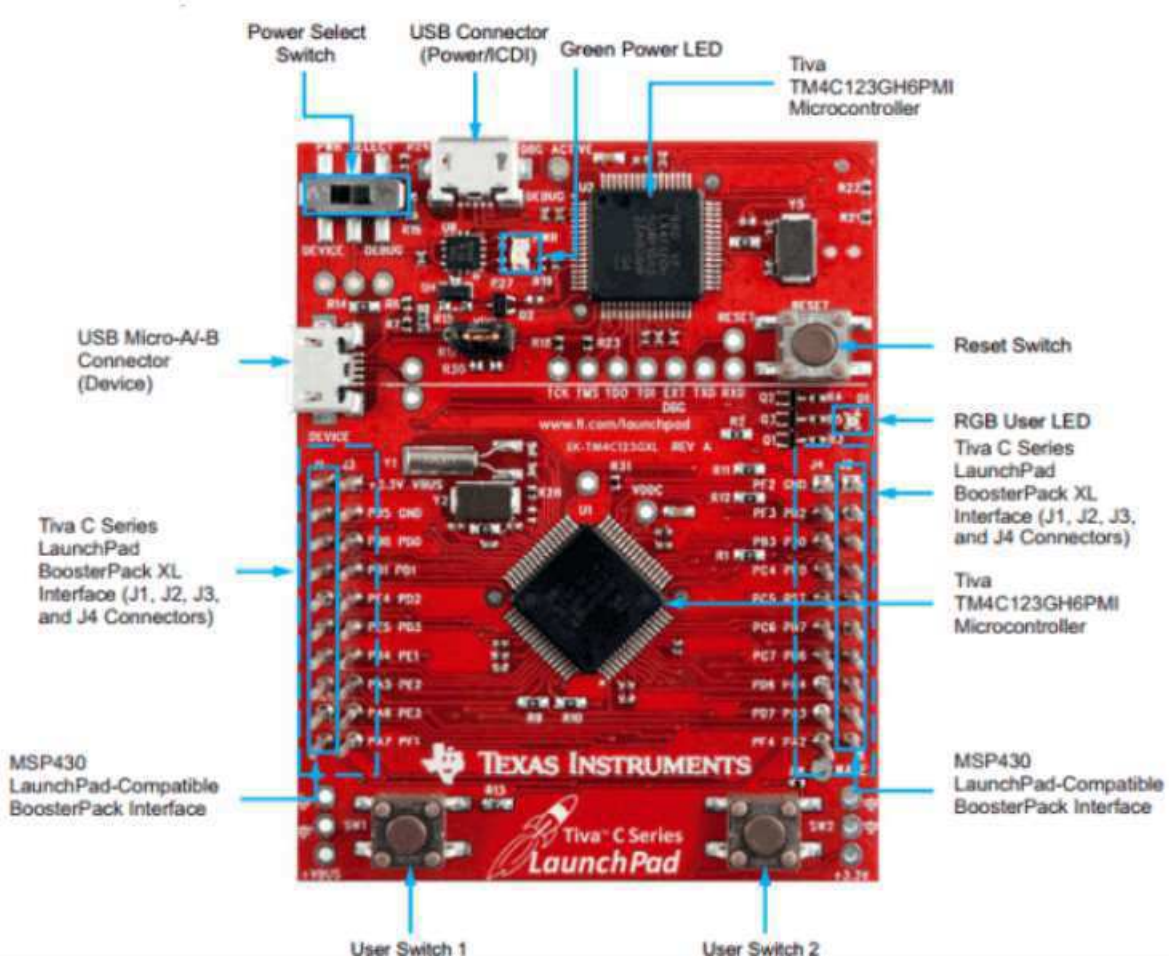
TM4C123G *LaunchPad*

A *LaunchPad* Tiva C TM4C123G é uma placa de baixo custo para avaliação dos microcontroladores ARM Cortex-M4F da família Tiva C da Texas Instruments.

A placa possui uma série de recursos que facilitam os testes e aplicações durante o processo de aprendizagem do microcontrolador TM4C123GH6PMI. Além disso ela possui pinos de expansão que permitem conectar placas, conhecidas como *BoosterPacks*, que expandem as funcionalidades e possibilidades de aplicações. Por exemplo, podemos utilizar os pinos de expansão para conexão de *LEDs*, teclas, displays entre outros.

A figura 3 exibe a LaunchPad Tiva C TM4C123G e um resumo dos seus recursos:

Figura 3 - LaunchPad Tiva C TM4C123G



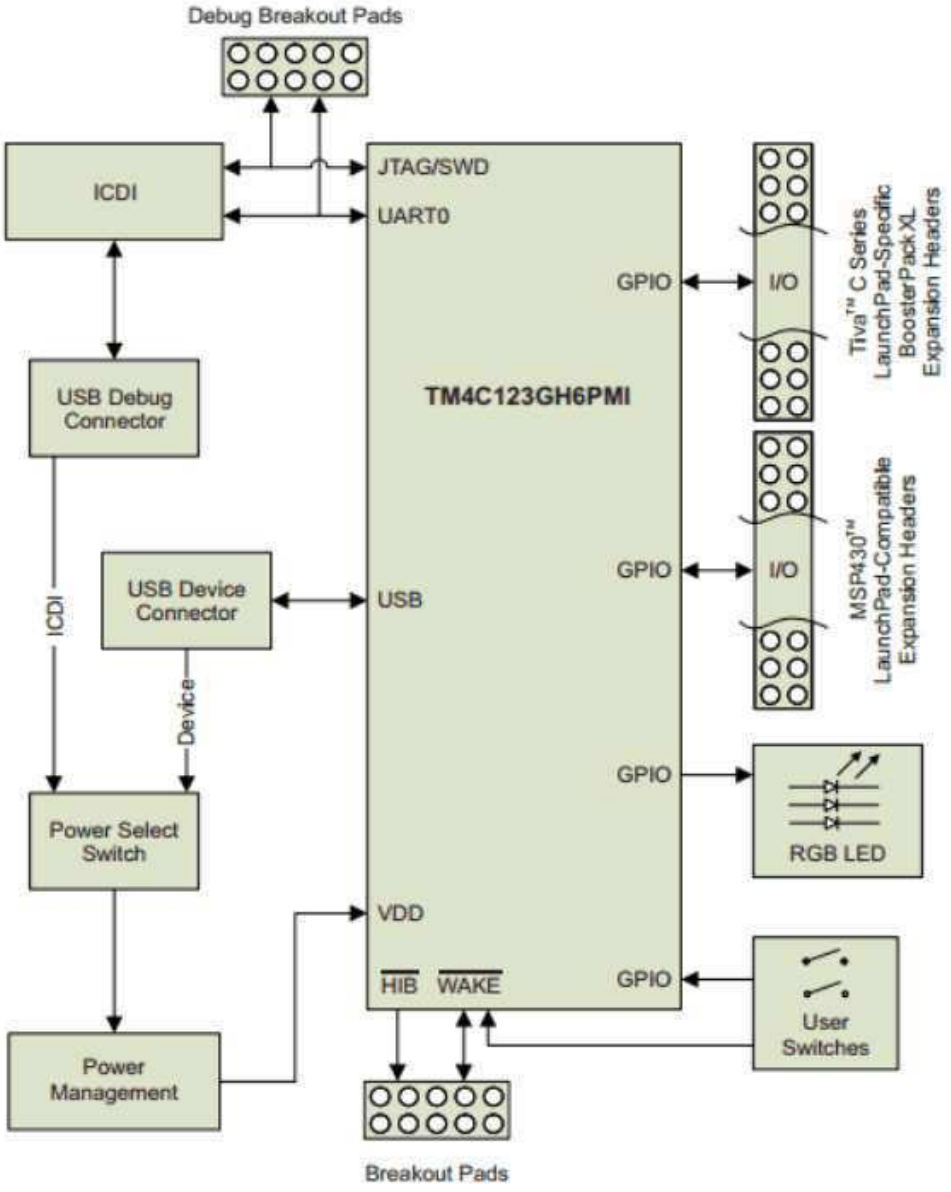
Recursos da placa LaunchPad Tiva C TM4C123G

A LaunchPad Tiva C TM4C123G apresenta os seguintes recursos:

- Microcontrolador Tiva TM4C123GH6PMI ARM Cortex-M4F;
- LED RGB;
- 2 teclas para uso geral;
- 40 pinos de expansão disponíveis em conector headers com passo de 2.54 mm;
- Circuito de depuração integrado - ICDI;
- Botão de Reset;
- 2 Conectores USB Micro-B:
    - Device;
    - Debugger.
- Chave para seleção de fonte de alimentação:
    - ICDI;
    - USB device.

A figura 4 exibe o diagrama de blocos da LaunchPad Tiva C TM4C123G:

Figura 4 - Diagrama de blocos da placa



Características do microcontrolador TM4C123GH6PMI

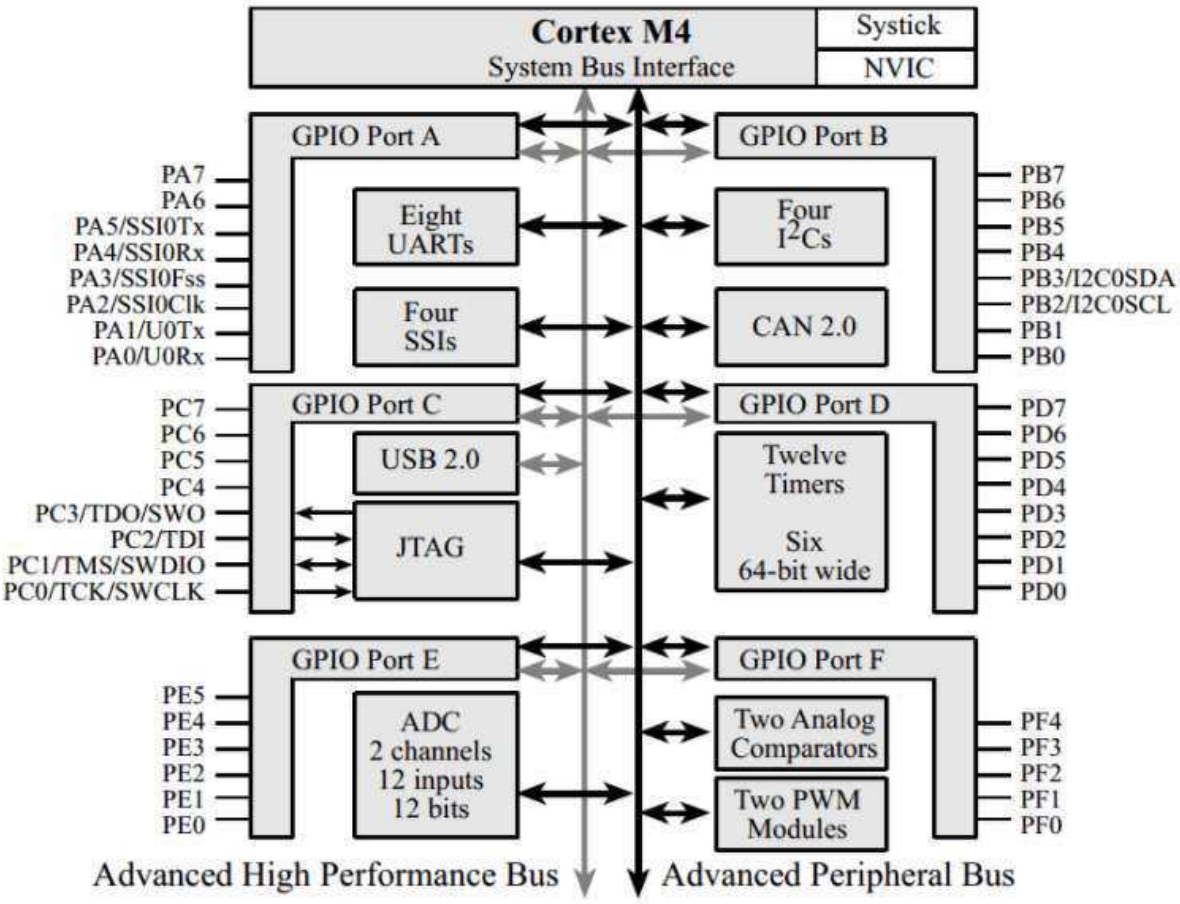
Como apresentado acima, o microcontrolador alvo da placa é o TM4C123GH6PMI, um ARM Cortex-M4F que possui as seguintes características:

Tabela 1 - Resumo de recursos do TM4C123GH6PMI

Núcleo	32 bit ARM Cortex-M4F
Performance	80-MHz; 100 DMIPS
Memória Flash	256 KB
Memória SRAM	32 KB
Memória EEPROM	2KB
UART	8 módulos
SSI - Synchronous Serial Interface	4 módulos
I2C	4 módulos com 4 velocidades de transmissão incluindo modo de alta velocidade
CAN	2 módulos CAN 2.0
USB	USB 2.0 OTG/Host/Device
ADC	2 ADCs de 12 bits
PWM	2 módulos PWM, totalizando 16 saídas PWM. Recursos para controle de motores
General-Purpose Timer (GPTM)	6 blocos de 16/32 bits e 6 blocos de 32/64 bits
Package	LQFP 64 pinos

A figura 5 exibe o diagrama de blocos do microcontrolador TM4C123GH6PMI:

Figura 5 - Diagrama de blocos do microcontrolador TM4C123GH6PMI



Especificações para alimentação da placa

A placa pode ser alimentada de 4,75 VDC a 5,25 VDC proveniente de uma das seguintes fontes:

- Conector USB Micro-B da interface de debugger (ICDI) conectado ao PC ou fonte externa;
- Conector USB Micro-B device conectado ao PC ou fonte externa.

A origem da alimentação deve ser selecionada na chave POWER SELECT (SW3).

Exemplos de experimentos

Após ter decidido uma opção de microcontrolador para a utilizar na parte prática da disciplina. É necessário a realização de alguns experimentos simples para entender o funcionamento de um microcontrolador e como fazer a sua programação. Para isso, foi pensado alguns exemplos de experimentos simples para a disciplina, com o intuito de apresentar as ferramentas básicas.

Segue a relação de laboratórios:

LABORATÓRIO 1:

Será passado um código em Assembly para que os alunos façam um download, e executem o programa uma vez. Logo em seguida, é pedido que o aluno altere a velocidade do piscar de um LED. O próximo passo é pedir ao aluno que faça com que determinadas cores pisquem com um certo valor de repetição.

O objetivo desse primeiro laboratório é mostrar como funciona a ferramenta de programação e entender um pouco como funciona a codificação em Assembly.

LABORATÓRIO 2:

Será implementar o laboratório passado adicionando a ele botões. Atividades como pressionar um botão e mudar as cores, ou alternar a velocidade do piscar do LED.

O objetivo aqui é mostrar como configurar as portas de entrada e saída de dados do microcontrolador, além de adquirir experiência com a linguagem Assembly.

LABORATÓRIO 3:

Será pedido um novo acréscimo aos códigos anteriores. Neste caso, será pedido para que os LED’s pisquem com um determinado tempo, ainda de acordo com o pressionar dos botões.

Neste experimento, é necessário o conhecimento do clock e também como calcular a conversão do clock para o tempo desejado.

LABORATÓRIO 4:

A partir daqui, podemos iniciar a programação em C, e pedir para que os alunos realizassem os experimentos anteriores modificando a linguagem de Assembly para C e com isso desenvolvesse um simples jogo.

O objetivo aqui é apenas de mostrar que a linguagem em C, irá facilitar a programação do microcontrolador, e dessa forma entender melhor quando usar o Assembly ou C.

Esses são apenas exemplos simples de como iniciar o estudo de Sistemas Embarcados, o nível de dificuldade dos próximos experimentos devem ser maiores afim de ir incentivando o aluno a continuar o aprendizado.

LABORATÓRIO 5:

Este laboratório o intuito é ensinar como é feita a comunicação UART (Universal Asynchronous Receiver/Transmitter), e é pedido que o aluno faça algumas rotinas para que mostre como é feita a comunicação para o computador. Um novo software é necessário nesse laboratório, o *Putty*.

LABORATÓRIO 6:

Este laboratório o intuito é continuar ensinando como é feita a comunicação UART (Universal Asynchronous Receiver/Transmitter), e é pedido que o aluno faça algumas rotinas para a montagem de um despertador. Este laboratório será um pouco mais difícil por ter que envolver a transmissão de dados e um tratamento dele.

LABORATÓRIO 7:

Este laboratório o intuito é ensinar problemas de concorrência, causalidade, e seções criticas de dados. Ou seja, será dada uma abrangência ao tratamento de interrupções e prioridades dos dados.

LABORATÓRIO 8:

Para finalizar os laboratórios, o problema a ser resolvido agora é a conversão de analógico para digital e de digital para analógico.

PROJETOS

A partir desses laboratórios, o aluno estará pronto para realizar qualquer outro tipo de atividade, e é nesse momento que começa o estudo para um projeto final da disciplina. Segue uma lista de prováveis projetos:

- Controle da posição de painéis solares para uma melhor captação dos raios;
- Controle de um carro autônomo;
- Controle de um pêndulo invertido;
- Controle de um motor utilizando conhecimentos de controladores PID;
- Estação de controle de clima;
- Montagem de um animal de estimação;
- Tradutor de código morse;
- Diversos tipos de jogos;
- Desenvolver um mini-robô;
- Montagem de um piano;
- Tratamento de um sinal de voz;
- Identificação do gênero pela voz;
- Montagem de um rádio;
- Modulador de som;
- Utilizar como um mouse para o PC;
- Controlar utilizando a voz;
- Desenvolver dois rôbos, onde um copia o outro;
- Criar um joystick;

CONCLUSÃO

Com este trabalho, avaliou-se a importância de uma nova disciplina para a graduação de Engenharia Elétrica da UFCG. A disciplina de Sistemas Embarcados irá proporcionar um aprendizado necessário para o desenvolvimento de novas tecnologias e também o conhecimento de tecnologias já existentes no nosso meio.

O microcontrolador TM4C123G foi escolhido por ser um microcontrolador de baixo custo, com arquitetura ARM e principalmente por ser capaz de ser utilizado nas mais diversas aplicações, o que facilita o uso dos alunos e uma diversificação de projetos a cada semestre. Um outro motivo é de existir livros que utilizam esse mesmo microcontrolador para o ensino da disciplina. Existem outros microcontroladores que também podem ser utilizados e nesse caso, fica a cargo do professor escolher.

Os laboratórios iniciais devem ser planejados e ter os códigos abertos, o intuito deles é de mostrar aos alunos como é feita todas as configurações inicias e necessárias para o funcionamento correto do microcontrolador.

Por fim, finalizo esse trabalho com o intuito de ajudar na criação dessa disciplina e fomentar o estudo de Sistemas Embarcados dentro da UFCG, assunto esse que poderá ser fonte de inúmeras pesquisas e estimular o estudo para outras áreas ainda com o desenvolvimento precário nessa instituição, como por exemplo, a robótica.

REFERÊNCIAS

1. VALVANO, Jonathan W. **Introduction to the MSP432 Microcontroller**. Volume 1, 1ª Edição. 2015.

2. PEREIRA, L. A. M; CARVALHO, F. R; BORTULUCCI, T; MORAES, M. H; **Software Embarcado, o crescimento e as novas tendências deste mercado**. Revista de Ciências Exatas e Tecnologia, 2014.

3. **TI Lauchpad.** Disponível em: http://www.ti.com/launchpad Acesso em 27/09/2017

4. **Arquitetura ARM**. Disponível em: http://www.roboliv.re/uploads/documentos/arquiteturas-de-computadores/ARM-2011-08-08_1_1343264119.pdf Acesso em 26/09/2017

5. **Arquitetura de Microcontroladores Modernos.** Disponível em: http://www.mzeditora.com.br/artigos/mic_modernos.htm Acesso em 26/09/2017

6. **Sistemas Embarcados, Computação Invisível.** Disponível em: http://www.hardware.com.br/artigos/sistemas-embarcados-computacao-invisivel/ Acesso em 27/09/2017

7. **Sistemas Embarcados.** Disponível em: https://pt.wikibooks.org/wiki/Sistemas_operacionais/Sistemas_embarcados Acesso em 24/09/2017

8. **O que é um sistema embarcado.** Disponível em: http://www.embarc.com.br/p1600.aspx Acesso em 24/07/2017

9. **Embedded Systems.** Disponível em: http://www.ni.com/embedded-systems/pt/ Acesso em 20/07/2017

10. **Disciplina de Sistemas Embarcados da Universidade de Kentucky**. Disponível em: http://idea.engr.uky.edu/wiki/doku.php?id=classes:15s:383:default Acesso em 20/07/2017

11. **Application for Microcontrollers.** Disponível em: http://www.ti.com/lsds/ti/microcontrollers-16-bit-32-bit/applications.page#auto Acesso em 20/07/2017

12. **Students Projects.** Disponível em: https://os.mbed.com/cookbook/Student-Projects Acesso em 20/07/2017

13. **Microcontroller Based Projects.** Disponível em: http://nevonprojects.com/microcontroller-based-projects/ Acesso em 20/07/2017

ANEXOS:

GUIA DO LABORATÓRIO 1:

LABORATÓRIO 1

Objetivo

- Adquirir experiência com o Assembly e com o TI TIVA C Lauchpad;
- Adquirir experiência com o ambiente de desenvolvimento.

Leitura Recomendada

- Jonathan W. Valvano, “Introduction to ARM Cortex-M Microcontrollers” (fifth edition-5/25/2013), 2013 – (Cap. 3.1- 3.3, 4.1.3-4.2.1)

Introdução

O primeiro experimento é para a familiarização com o ambiente e para iniciar as atividades com o microcontrolador. Para isso iremos começar instalando o programa para desenvolver nossas atividades.

O passo a passo de como instalar é encontrado no seguinte link:

http://www.ti.com/lit/ml/spmu355/spmu355.pdf

1. A primeira atividade é baixar o arquivo es_lab1.zip e abrir ele na plataforma do programa. Verificar se o programa funciona corretamente e tentar alterar o tempo que os LED’s piscam.
2. A segunda atividade desse laboratório é reescrever o programa de modo que o LED azul pisque 3 vezes, o LED vermelho 5 vezes e o LED verde 4 vezes, usando estruturas de loop. (OBS: Os registradores R0 e R1 já estão sendo usados).
3. Usar a ferramente de breakpoint do Keil e configurar lugares de parada do código para utilizar futuramente como uma forma de descobrir erros.

GUIA DO LABORATÓRIO 2:

LABORATÓRIO 2

Objetivo

- Adquirir experiência com o Assembly e com o TI Tiva C Lauchpad;
- Adquirir experiência com o ambiente de desenvolvimento.
- Adquirir experiência com a configuração das Portas I/O e a interface de pinos GPIO dos LED's e dos botões.

Leitura Recomendada

- Jonathan W. Valvano, "Introduction to ARM Cortex-M Microcontrollers" (fifth edition-5/25/2013), 2013 – (Cap. 3.1- 3.3, 4.1.3-4.2.1);
- Data-sheet: Tiva TM4C123GH6PM Microcontroller

Introdução

O segundo experimento é para aprender como usar os botões e os LED's já disponíveis no microcontrolador. O desafio é ler de forma correta os botões e assim dar uma resposta com os LED's.

Experimento

4. A primeira atividade é baixar o arquivo es_lab2.zip e utilizá-lo para iniciar o laboratório. Com isso é necessário fazer uma alteração na inicialização da porta F. Baseando na leitura recomendada, realizar a inicialização para que PF0-PF4 funcionem corretamente; Feito isto, realizar um loop para ler a porta F e armazenar esse valor em uma variável;
5. A segunda atividade desse laboratório é reescrever o programa para inserimos uma lógica: se os dois botões forem pressionados, o LED verde deverá acender; Se o botão da direita (SW2) for pressionado, o LED vermelho deverá acender, e se o botão da esquerda (SW1) for pressionado, o LED azul deverá acender. Se nenhum botão for pressionado, o LED deverá permanecer desligado.

GUIA DO LABORATÓRIO 3:

LABORATÓRIO 3

Objetivo

- Adquirir experiência com o Assembly e com o TI Tiva C Lauchpad;
- Adquirir experiência com o ambiente de desenvolvimento.
- Adquirir experiência com a configuração das Portas I/O e a interface de pinos GPIO dos LED's e dos botões.

Leitura Recomendada

- Jonathan W. Valvano, "Introduction to ARM Cortex-M Microcontrollers" (fifth edition-5/25/2013), 2013 – (Cap. 3.1- 3.3, 4.1.3-4.2.1);
- Data-sheet: Tiva TM4C123GH6PM Microcontroller

Introdução

O terceiro experimento é para aprender como controlar o tempo, algo bastante necessário para um real controle. Continuaremos a utilizar os botões e o LED's presentes na placa.

Experimento

1. A primeira atividade é baixar o arquivo es_lab3.zip e utilizá-lo para iniciar o laboratório. Escreva um loop que irá fazer o LED vermelho ficar ligado por 1ms e desligado por 1ms. Assuma que a taxa do processamento do clock é 16MHz e calcule como encontrar um delay de 1ms. Utilizar um osciloscópio para checar se o resultado está correto.
2. Sabemos que variando o duty cycle podemos controlar a tensão média nos periféricos. Neste ponto queremos que os botões gerem duty cycle diferentes, por exemplo: Se os dois estiverem pressionados, o duty cycle é de 0%, se apenas o esquerdo estiver pressionado, o duty cycle é de 25%, se apenas o direito pressionado, o duty cycle é de 75%, e se ambos não estiverem pressionados, o duty cycle é de 100%. Continuar utilizando o LED vermelho para analisar;
3. A terceira atividade é um pouco mais complicada. Se nenhum botão estiver pressionado, acenda o LED amarelo; Se qualquer botão for pressionado, desligar o LED amarelo; Se o botão SW1 for solto, o LED vermelho deverá piscar com um duty cycle de 50% com um período de 2ms; Se o botão SW2 for solto, o LED verde deverá piscar com um duty clycle de 33% com um período de 1.5ms; Se ambos os botões forem soltos, os dois casos anteriormentes informados deverão acontecer.

GUIA DO LABORATÓRIO 4:

LABORATÓRIO 4

Objetivo

- Adquirir experiência com a Linguagem C e com o TI Tiva C Lauchpad;
- Adquirir experiência com o ambiente de desenvolvimento.
- Adquirir experiência com a configuração das Portas I/O e a interface de pinos GPIO dos LED's e dos botões.

Leitura Recomendada

- Jonathan W. Valvano, "Introduction to ARM Cortex-M Microcontrollers" (fifth edition-5/25/2013), 2013 – (Cap. 4 e 5);
- Data-sheet: Tiva TM4C123GH6PM Microcontroller

Introdução

O quarto experimento é para aprender como o controle do tempo e o seu uso de maneira correta soluciona inúmeros problemas da vida real. O objetivo com esse experimento é utilizar agora o uso da linguagem C para realizar os problemas propostos.

Experimento

4. A primeira atividade é baixar o arquivo es_lab4.zip e utilizá-lo para iniciar o laboratório. Perceba que o arquivo startup.s é um pouco diferente em C, porém consiste em fazer os mesmos passos já feitos em Assembly.
5. O problema proposto é um jogo. O LED vermelho irá ficar ligado durante um certo tempo e o usuário deverá pressionar um dos botões logo que visualizar a cor vermelha para que seja calculada o tempo de reação.
    a. Primeiramente, o LED verde será ligado demonstrando que o jogo está completamente configurado e pronto para iniciar;
    b. O usuário pressiona SW1 para indicar que está pronto para jogar;
    c. O LED verde irá desligar;
    d. Randomicamente, o LED vermelho irá acender em um intervalo de tempo entre 500ms e 3500ms;
    e. Logo que o LED acender, inicia-se o calculo do tempo;
    f. O usuário deverá pressionar o botão SW2, se isso não for feito em no máximo 2s piscará as seguintes cores: amarelo, vermelho, amarelo, vermelho e volta pra o estado inicial (a);

g. Se o usuário clicar no intervalo dado, o tempo é salvo e feita a diferença, guarda-se o tempo de reação no registrador R8;
h. Espera-se 500ms e chama a função resultado();
i. Retorna para o estado inicial (a);

6. É pedido que desenvolva a função resultado(), tomando as seguintes regras:
    a. Divida o tempo encontrado por 100, e faça que o LED verde pisque a quantidade de vezes igual ao quociente; Salve o resto dessa operação;
    b. Divida o novo valor por 10, e faça que o LED amarelo pisque a quantidade de vezes igual ao quociente; Salve o resto dessa operação;
    c. Faça o LED vermelho piscar a quantidade de vezes do restante do valor salvo;
    d. Retorne a função.

A resposta pode ser entendida pelo usuário da seguinte foram:

Tempo de reação (ms) = (piscadas do LED verde)*100 + (piscadas do LED amarelo)*10 + (piscadas do LED vermelho).

GUIA DO LABORATÓRIO 5:

LABORATÓRIO 5

Objetivo

- Adquirir experiência com o subsistema ARM UART;
- Adquirir experiência com o ambiente de desenvolvimento.

Leitura Recomendada

- Jonathan W. Valvano, “Introduction to ARM Cortex-M Microcontrollers” (fifth edition-5/25/2013), 2013 – (Cap. 4);
- Data-sheet: Tiva TM4C123GH6PM Microcontroller

Introdução

O quinto experimento é para aprender como realizar uma comunicação entre a placa e um computador utilizado a comunicação UART.

Experimento

7. A primeira atividade é baixar o arquivo es_lab5.zip e utilizá-lo para iniciar o laboratório.
8. O segundo passo é instalar o programa Putty, que é um emulador de um terminal. É necessário saber em qual porta a placa está conectada, escolher a opção de 8 data bits, 1 stop bit, no parity bits, 115200 baud. O seguinte texto deverá ser apresentado:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
```

9. Utilizar os pinos TXD e RXD para verificar em um osciloscópio que os caracteres estão sendo transmitidos e recebidos. Comparar com o código ASCII de cada caractere.

GUIA DO LABORATÓRIO 6:

LABORATÓRIO 6

Objetivo

- Adquirir experiência com o subsistema ARM UART;
- Adquirir experiência com o ambiente de desenvolvimento.

Leitura Recomendada

- Jonathan W. Valvano, “Introduction to ARM Cortex-M Microcontrollers” (fifth edition-5/25/2013), 2013 – (Cap. 9 -9.6);
- Data-sheet: Tiva TM4C123GH6PM Microcontroller

Introdução

O sexto experimento é para continuar aprendendo sobre a comunicação UART. Iremos utilizar um nova biblioteca para implementar uma real-time clock (RTC)

Experimento

10. A primeira atividade é baixar o arquivo es_lab6.zip e utilizá-lo para iniciar o laboratório. O aluno deverá implementar duas funções fgetc() e fputc(). Essa implementação deverá chamar as funções em C UART_inchar() e UART_Outchar() presentes no arquivo.
11. Escreva uma função chamada int32_t TIME_GetTime() e solicite o tempo no formato AAAAMMDDHHMISS. Faça um loop que peça o tempo e depois imprima dessa forma:

    Ano: 2017, Mês: 10, Dia: 08, Hora: 12, Minutos: 00, Segundos: 00.

12. Escreva uma nova função que calcule a diferença entres duas datas e dê a resposta em segundos; O aluno deverá considerar que todos os anos tem a mesma quantidade de dias.
13. Escreva uma nova função que incremente um segundo para uma das datas. O Programa deverá adicionar um segundo na primeira data, até que atinga o valor da segunda data. (data1<data2). Quando isso ocorrer, uma nova mensagem deve ser apresentada.

GUIA DO LABORATÓRIO 7:

LABORATÓRIO 7

Objetivo

- Adquirir experiência com o subsistema ARM NVIC e conceitos de interrupção;

Leitura Recomendada

- Jonathan W. Valvano, “Introduction to ARM Cortex-M Microcontrollers” (fifth edition-5/25/2013), 2013 – (Cap. 11.2);
- Data-sheet: Tiva TM4C123GH6PM Microcontroller

Introdução

O sétimo experimento é para entendimento de como realizar as interrupções e como configurar a prioridade do mesmo.

Experimento

14. A primeira atividade é baixar o arquivo com todos os códigos auxiliares. Para ter certeza que tudo está funcionando da maneira correta, teste o o seguinte código;
    a. Void SysTick_Handler (void){
        PF2 ^= 0x04;

        }
15. Utilizando o código auxiliar, entender como é feita a contagem de um tempo que depende de duas variáveis por causa da quantidade de bits;
16. Modificar o jogo feito no experimento 4, para coletar respostas com um tempo mais que 12.5ns;
17. Realizar uma interrupção para capturar o tempo de resposta do jogador e melhorar assim o jogo feito.

GUIA DO LABORATÓRIO 8:

LABORATÓRIO 8

Objetivo

- Adquirir experiência com a conversão de analógico para digital e de digital para analógico;

Leitura Recomendada

- Jonathan W. Valvano, “Introduction to ARM Cortex-M Microcontrollers” (fifth edition-5/25/2013), 2013 – (Cap. 10);
- Data-sheet: Tiva TM4C123GH6PM Microcontroller

Introdução

O oitavo experimento é para entendimento de como realizar conversões ADC e DAC com o microcontrolador.

Experimento

18. A primeira atividade é baixar o arquivo com todos os códigos auxiliares.
19. Logo após, é necessário escrever duas funções: void DAC_Init (void) e void DAC_Write(unsigned char); Usar o array SineWave32 e SineWave16, e relatar qual é a melhor; Olhar resultados no osciloscópio.
20. Utilizando-se de um circuito simples com potenciômetro e de um dos códigos auxiliares, mostrar que os valores variam de 0.0 a 3.3V. Verificar que os valores no registrador variam de 0x000 a 0xFFF;
21. Combinar a etapa 2 com a 3, utilizando-se de um gerador de sinais; Lembrar de não colocar uma tensão na entrada maior que 3.3V;
22. Utilizando-se de um filtro FIR presente nos códigos auxiliares, modificar a ordem do mesmo e analisar como fica a resposta.

CÓDIGOS PARA O AUXÍLIO DOS LABORATÓRIOS

Códigos para Laboratório 1

```
;****************** main.s ***************

      IMPORT   PortF_Init
               IMPORT   delay
      IMPORT   blue_led_on
               IMPORT   blue_led_off
      IMPORT   red_led_on
               IMPORT   red_led_off
               IMPORT   green_led_on
               IMPORT   green_led_off

      AREA    |.text|, CODE, READONLY, ALIGN=2
      THUMB
      EXPORT  Start


Start
   BL  PortF_Init               ; initialize input and output pins of Port F

loop
         BL  blue_led_on
   BL  delay
         BL  blue_led_off
   BL  delay

         BL  red_led_on
         BL  delay
         BL  red_led_off
         BL  delay

         BL  green_led_on
         BL  delay
         BL  green_led_off
         BL  delay

         B   loop


               END
```

```
;****************** LED.s ***************
; Credit: Based on software by Valvano
;****************************************

GPIO_PORTF_DATA_R  EQU 0x400253FC
GPIO_PORTF_DIR_R   EQU 0x40025400
GPIO_PORTF_AFSEL_R EQU 0x40025420
GPIO_PORTF_PUR_R   EQU 0x40025510
GPIO_PORTF_DEN_R   EQU 0x4002551C
GPIO_PORTF_LOCK_R  EQU 0x40025520
GPIO_PORTF_CR_R    EQU 0x40025524
GPIO_PORTF_AMSEL_R EQU 0x40025528
GPIO_PORTF_PCTL_R  EQU 0x4002552C
GPIO_LOCK_KEY      EQU 0x4C4F434B  ; Unlocks the GPIO_CR register
SYSCTL_RCGC2_R     EQU 0x400FE108
SYSCTL_RCGC2_GPIOF EQU 0x00000020  ; port F Clock Gating Control
RED                EQU 0x02
BLUE               EQU 0x04
GREEN              EQU 0x08
PF1                     EQU 0x40025008
PF2                     EQU 0x40025010
PF3                     EQU 0x40025020


      AREA    |.text|, CODE, READONLY, ALIGN=2
      THUMB
      EXPORT   PortF_Init
               EXPORT   delay
      EXPORT   blue_led_on
               EXPORT   blue_led_off
      EXPORT   red_led_on
               EXPORT   red_led_off
               EXPORT   green_led_on
               EXPORT   green_led_off


                        ;------------PortF_Init------------
; Initialize GPIO Port F for negative logic switches on PF0 and
; PF4 as the Launchpad is wired.  Weak internal pull-up
; resistors are enabled, and the NMI functionality on PF0 is
; disabled.  Make the RGB LED's pins outputs.
; Input: none
; Output: none
; Modifies: R0, R1, R2

PortF_Init
```

```
   LDR R1, =SYSCTL_RCGC2_R         ; 1) activate clock for Port F
   LDR R0, [R1]
   ORR R0, R0, #0x20               ; set bit 5 to turn on clock
   STR R0, [R1]
   NOP
   NOP                          ; allow time for clock to finish
   LDR R1, =GPIO_PORTF_LOCK_R      ; 2) unlock the lock register
   LDR R0, =0x4C4F434B             ; unlock GPIO Port F Commit Register
   STR R0, [R1]
   LDR R1, =GPIO_PORTF_CR_R        ; enable commit for Port F
   MOV R0, #0xFF                   ; 1 means allow access
   STR R0, [R1]
   LDR R1, =GPIO_PORTF_AMSEL_R     ; 3) disable analog functionality
   MOV R0, #0                      ; 0 means analog is off
   STR R0, [R1]
   LDR R1, =GPIO_PORTF_PCTL_R      ; 4) configure as GPIO
   MOV R0, #0x00000000             ; 0 means configure Port F as GPIO
   STR R0, [R1]
   LDR R1, =GPIO_PORTF_DIR_R       ; 5) set direction register
   MOV R0, #0x0E                   ; PF0 and PF7-4 input, PF3-1 output
   STR R0, [R1]
   LDR R1, =GPIO_PORTF_AFSEL_R     ; 6) regular port function
   MOV R0, #0                      ; 0 means disable alternate function
   STR R0, [R1]
   LDR R1, =GPIO_PORTF_PUR_R       ; pull-up resistors for PF4,PF0
   MOV R0, #0x11                   ; enable weak pull-up on PF0 and PF4
   STR R0, [R1]
   LDR R1, =GPIO_PORTF_DEN_R       ; 7) enable Port F digital port
   MOV R0, #0xFF                   ; 1 means enable digital I/O
   STR R0, [R1]
   BX  LR

;------------delay------------
; Delay function for testing, which delays about 3*count cycles.
; Input: R0 count
; Output: none
ONESEC            EQU 5333333      ; approximately 1s delay at ~16 MHz clock
QUARTERSEC        EQU 1333333      ; approximately 0.25s delay at ~16 MHz clock
FIFTHSEC          EQU 1066666      ; approximately 0.2s delay at ~16 MHz clock
delay
         LDR R0, =QUARTERSEC
delay_loop
   SUBS R0, R0, #1                 ; R0 = R0 - 1 (count = count - 1)
   BNE delay_loop                     ; if count (R0) != 0, skip to 'delay'
   BX  LR                       ; return
```

```
;------------PortF_Output------
; Set the output state of PF3-1.
; Input: R0  new state of PF
; Output: none
; Modifies: R1
PortF_Output
   LDR R1, =GPIO_PORTF_DATA_R                ; pointer to Port F data
   STR R0, [R1]                              ; write to PF3-1
   BX  LR

;------------blue_led_on------
; Turn the blue LED on
; Input: none
; Output: none
blue_led_on
         LDR R1, =PF2
         MOV R0, #BLUE                ; R0 = BLUE (blue LED on)
   STR R0, [R1]                 ; turn the blue LED on
   BX  LR

;------------blue_led_off-----
; Turn the blue LED off
; Input: none
; Output: none
blue_led_off
         LDR R1, =PF2
         MOV R0, #0                   ; R0 = 0
   STR R0, [R1]                 ; turn the blue LED OFF
   BX  LR

;------------red_led_on-------
; Turn the red LED on
; Input: none
; Output: none
red_led_on
         LDR R1, =PF1
         MOV R0, #RED                 ; R0 = RED (red LED on)
   STR R0, [R1]                 ; turn the red LED on
   BX  LR

;------------red_led_off------
; Turn the red LED off
; Input: none
; Output: none
red_led_off
         LDR R1, =PF1
```

```
        MOV R0, #0                  ; R0 = 0
    STR R0, [R1]                ; turn the red LED OFF
    BX  LR

;------------green_led_on------
; Turn the green LED on
; Input: none
; Output: none
green_led_on
        LDR R1, =PF3
        MOV R0, #GREEN              ; R0 = GREEN (green LED on)
    STR R0, [R1]                ; turn the green LED on
    BX  LR

;------------green_led_off-----
; Turn the green LED off
; Input: none
; Output: none
green_led_off
        LDR R1, =PF3
        MOV R0, #0                  ; R0 = 0
    STR R0, [R1]                ; turn the green LED OFF
    BX  LR

    ALIGN                       ; make sure the end of this section is aligned
    END                         ; end of file
```

Códigos para Laboratório 2

```
;****************** main.s ***************

   THUMB

; PortF register declarations
GPIO_PORTF_DATA_R  EQU 0x400253FC
GPIO_PORTF_DIR_R   EQU 0x40025400
GPIO_PORTF_AFSEL_R EQU 0x40025420
GPIO_PORTF_PUR_R   EQU 0x40025510
GPIO_PORTF_DEN_R   EQU 0x4002551C
GPIO_PORTF_LOCK_R  EQU 0x40025520
GPIO_PORTF_CR_R    EQU 0x40025524
GPIO_PORTF_AMSEL_R EQU 0x40025528
GPIO_PORTF_PCTL_R  EQU 0x4002552C
GPIO_LOCK_KEY      EQU 0x4C4F434B  ; Unlocks the GPIO_CR register
SYSCTL_RCGC2_R     EQU 0x400FE108
SYSCTL_RCGC2_GPIOF EQU 0x00000020  ; port F Clock Gating Control

        AREA DATA, ALIGN=2
                ; Global variables go here, you will not need any in lab2.

        ALIGN ; make sure the end of this section is aligned
        AREA |.text|, CODE, READONLY, ALIGN=2
        EXPORT Start

Start

PortF_Init
   LDR R1, =SYSCTL_RCGC2_R        ; 1) activate clock for Port F
   LDR R0, [R1]
   ORR R0, R0, #0x20              ; set bit 5 to turn on clock
   STR R0, [R1]
   NOP
   NOP                        ; allow time for clock to finish
                                                        ; 2) unlock the lock register
                                                        ;    unlock    GPIO    Port    F
Commit Register
                                                        ; enable commit for Port F
                                                        ; 1 means allow access
                                                        ;       3)       disable       analog
functionality
                                                        ; 0 means analog is off
                                                        ; 4) configure as GPIO
```

```
                                                    ; 0 means configure Port F as
GPIO
                                                    ; 5) set direction register
                                                    ; PF0 and PF7-4 input, PF3-1
output
                                                    ; 6) regular port function
                                                    ; 0 means disable alternate
function
                                                    ; pull-up resistors for
PF4,PF0
                                                    ; enable weak pull-up on PF0
and PF4
                                                    ; 7) enable Port F digital port
                                                    ; 1 means enable digital I/O

mainloop
                ; Write your main program here. This segment loops indefinitely while the
microcontroller is on
                ; You must use the Portf_Input and Portf_Output functions to access PortF

        B mainloop  ;infinite main loop

; Useful Functions
; (From Valvano Text Program 4.1)
;------------PortF_Input------
; Read ProtF
; Input: none
; Output: R0  value read from PortF
; Modifies: R1
PortF_Input
   LDR R1, =GPIO_PORTF_DATA_R           ; pointer to Port F data
        LDR R0, [R1]                    ; read all of PortF
        AND R0,R0,#0x11                         ; just the input pins, bits 4,0
   BX  LR                               ; return R0 with inputs

;------------PortF_Output------
; Set the output state of PF3-1.
; Input: R0  new state of PF
; Output: none
; Modifies: R1
PortF_Output
   LDR R1, =GPIO_PORTF_DATA_R           ; pointer to Port F data
   STR R0, [R1]                 ; write to PF3-1
   BX  LR
```

```
    ALIGN ; make sure the end of this section is aligned (For the code/data to follow)
    END ; mark end of file
```

Códigos para Laboratório 3

```
;****************** main.s ***************

GPIO_PORTF_DATA_R  EQU 0x400253FC
GPIO_PORTF_DIR_R   EQU 0x40025400
GPIO_PORTF_AFSEL_R EQU 0x40025420
GPIO_PORTF_PUR_R   EQU 0x40025510
GPIO_PORTF_DEN_R   EQU 0x4002551C
GPIO_PORTF_LOCK_R  EQU 0x40025520
GPIO_PORTF_CR_R    EQU 0x40025524
GPIO_PORTF_AMSEL_R EQU 0x40025528
GPIO_PORTF_PCTL_R  EQU 0x4002552C
GPIO_LOCK_KEY      EQU 0x4C4F434B  ; Unlocks the GPIO_CR register
SYSCTL_RCGC2_R     EQU 0x400FE108
SYSCTL_RCGC2_GPIOF EQU 0x00000020  ; port F Clock Gating Control

YELLOW                  EQU 0x0B
RED                EQU 0x02
BLUE               EQU 0x02
GREEN              EQU 0x08
SW1                     EQU 0x10
SW2                     EQU 0x01
PF1                     EQU 0x40025008
PF2                     EQU 0x40025010
PF3                     EQU 0x40025020



    AREA    |.text|, CODE, READONLY, ALIGN=2
    THUMB
    EXPORT  Start

Start

  BL  PortF_Init              ; initialize input and output pins of Port F

mainloop

        BL  PortF_Input
```

```
        CMP R0, #0x00
    BEQ bothpressed
        CMP R0, #0x01
    BEQ leftpressed
    CMP R0, #0x10
    BEQ rightpressed
    CMP R0, #0x11
    BEQ nopressed
        MOV R0, #YELLOW
    BL  PortF_Output
        B   mainloop

nopressed
        MOV R0, #YELLOW                 ; R0 = 0 (no LEDs on)
    BL  PortF_Output
        BL  delay1
        BL  delay1
    B   mainloop
rightpressed
        MOV R0, #GREEN              ; R0 = BLUE (blue LED on)
    BL  PortF_Output
        BL  delay0_5
        MOV R0, #0x00
        BL  PortF_Output ; turn the blue LED on
        BL  delay1
        B   mainloop
leftpressed
        MOV R0, #RED                ; R0 = BLUE (blue LED on)
    BL  PortF_Output
        BL  delay1
        MOV R0, #0x00
        BL  PortF_Output                        ; turn the blue LED on
        BL  delay1
        B   mainloop
bothpressed
        MOV R0, #(RED+GREEN)                ; R0 = BLUE (blue LED on)
    BL  PortF_Output
        BL  delay0_5
        MOV R0, #RED
        BL  PortF_Output                        ; turn the blue LED on
        BL  delay0_5
        MOV R0, #0x00
        BL  PortF_Output                        ; turn the blue LED on
        BL  delay0_5
        MOV R0, #GREEN
```

```
        BL  PortF_Output                    ; turn the blue LED on
        BL  delay0_5
        MOV R0, #(RED)          ; R0 = BLUE (blue LED on)
    BL  PortF_Output
        BL  delay0_5
        MOV R0, #RED
        BL  PortF_Output                    ; turn the blue LED on
        BL  delay0_5
        MOV R0, #GREEN
        BL  PortF_Output                    ; turn the blue LED on
        BL  delay0_5
        MOV R0, #0X00
        BL  PortF_Output                    ; turn the blue LED on
        BL  delay0_5
        MOV R0, #(RED)          ; R0 = BLUE (blue LED on)
    BL  PortF_Output
        BL  delay0_5
        MOV R0, #(RED+GREEN)
        BL  PortF_Output                    ; turn the blue LED on
        BL  delay0_5
        MOV R0, #0x00
        BL  PortF_Output                    ; turn the blue LED on
        BL  delay0_5
        MOV R0, #0x00
        BL  PortF_Output                    ; turn the blue LED on
        BL  delay0_5
        B   mainloop

PortF_Init
    LDR R1, =SYSCTL_RCGC2_R     ; 1) activate clock for Port F
    LDR R0, [R1]
    ORR R0, R0, #0x20           ; set bit 5 to turn on clock
    STR R0, [R1]
    NOP
    NOP                                     ; allow time for clock to finish
        LDR R1, = GPIO_PORTF_LOCK_R         ; 2) unlock the lock register
        LDR R0, = 0x4C4F434B                ; unlock GPIO Port F Commit Register
        STR R0, [R1]
        LDR R1, = GPIO_PORTF_CR_R           ; enable commit for Port F
        MOV R0, #0xFF                               ; 1 means allow access
        STR R0, [R1]
        LDR R1, =GPIO_PORTF_AMSEL_R     ; 3) disable analog functionality
    MOV R0, #0              ; 0 means analog is off
    STR R0, [R1]
    LDR R1, =GPIO_PORTF_PCTL_R      ; 4) configure as GPIO
    MOV R0, #0x00000000         ; 0 means configure Port F as GPIO
```

```
   STR R0, [R1]
        LDR R1, = GPIO_PORTF_DIR_R          ; 5) set direction register
        MOV R0, #0x0E                              ; PF0 and PF7-4 input, PF3-1 output
        STR R0, [R1]
        LDR R1, =GPIO_PORTF_AFSEL_R    ; 6) regular port function
   MOV R0, #0                 ; 0 means disable alternate function
   STR R0, [R1]
        LDR R1, = GPIO_PORTF_PUR_R          ; pull-up resistors for PF4, PF0
        MOV R0, #0x11                              ; enable weak pull-up on PF0 and
PF4
        STR R0, [R1]
        LDR R1, = GPIO_PORTF_DEN_R          ; 7) enable Port F digital port
        MOV R0, #0xFF                              ; 1 means enable digital I/ O
        STR R0, [R1]
        BX LR


PortF_Input
   LDR R1, =GPIO_PORTF_DATA_R               ; pointer to Port F data
        LDR R0, [R1]                               ; read all of PortF
        AND R0,R0,#0x11                            ; just the input pins, bits 4,0
   BX  LR                                          ; return R0 with inputs


PortF_Output
   LDR R1, =GPIO_PORTF_DATA_R               ; pointer to Port F data
   STR R0, [R1]                        ; write to PF3-1
   BX  LR


MILLISECOND                EQU 5333         ; approximately 1ms delay at ~16 MHz
clock
ZEROCINCOMILLISECOND       EQU 2666       ; approximately 0.5ms delay at ~16 MHz
clock
UMCINCOMILLISECOND         EQU 8000     ; approximately 1.5ms delay at ~16 MHz clock

delay1
        LDR R0, =MILLISECOND
delayloop
   SUBS R0, R0, #1                  ; R0 = R0 - 1 (count = count - 1)
   BNE delayloop              ; if count (R0) != 0, skip to 'delay'
   BX  LR

delay0_5
        LDR R2, =ZEROCINCOMILLISECOND
delay2loop
```

```
   SUBS R2, R2, #1                ; R0 = R0 - 1 (count = count - 1)
   BNE delay2loop                 ; if count (R0) != 0, skip to 'delay'
   BX  LR

delay1_5
        LDR R2, =UMCINCOMILLISECOND
delay3loop
   SUBS R2, R2, #1                ; R0 = R0 - 1 (count = count - 1)
   BNE delay3loop                 ; if count (R0) != 0, skip to 'delay'
   BX  LR



        ALIGN ; make sure the end of this section is aligned (For the code/data to follow)
        END ; mark end of file



;****************** FUNCTIONS***************


GPIO_PORTF_DATA_R  EQU 0x400253FC
GPIO_PORTF_DIR_R   EQU 0x40025400
GPIO_PORTF_AFSEL_R EQU 0x40025420
GPIO_PORTF_PUR_R   EQU 0x40025510
GPIO_PORTF_DEN_R   EQU 0x4002551C
GPIO_PORTF_LOCK_R  EQU 0x40025520
GPIO_PORTF_CR_R    EQU 0x40025524
GPIO_PORTF_AMSEL_R EQU 0x40025528
GPIO_PORTF_PCTL_R  EQU 0x4002552C
GPIO_LOCK_KEY      EQU 0x4C4F434B  ; Unlocks the GPIO_CR register
SYSCTL_RCGC2_R     EQU 0x400FE108
SYSCTL_RCGC2_GPIOF EQU 0x00000020  ; port F Clock Gating Control

RED                EQU 0x02
BLUE               EQU 0x04
GREEN              EQU 0x08
SW1                     EQU 0x10
SW2                     EQU 0x01
PF1                     EQU 0x40025008
PF2                     EQU 0x40025010
PF3                     EQU 0x40025020

      AREA    |.text|, CODE, READONLY, ALIGN=2
      THUMB
      EXPORT   PortF_Init
```

```
;                   //EXPORT   PortF_Input
                    EXPORT   nopressed
          EXPORT   rightpressed
                    EXPORT   leftpressed
                    EXPORT   bothpressed
                    EXPORT   routine



                              ;------------PortF_Init------------
; Initialize GPIO Port F for negative logic switches on PF0 and
; PF4 as the Launchpad is wired.  Weak internal pull-up
; resistors are enabled, and the NMI functionality on PF0 is
; disabled.  Make the RGB LED's pins outputs.
; Input: none
; Output: none
; Modifies: R0, R1, R2

PortF_Init
   LDR R1, =SYSCTL_RCGC2_R        ; 1) activate clock for Port F
   LDR R0, [R1]
   ORR R0, R0, #0x20              ; set bit 5 to turn on clock
   STR R0, [R1]
   NOP
   NOP                                                  ; allow time for clock to finish
         LDR R1, = GPIO_PORTF_LOCK_R          ; 2) unlock the lock register
         LDR R0, = 0x4C4F434B                 ; unlock GPIO Port F Commit Register
         STR R0, [R1]
         LDR R1, = GPIO_PORTF_CR_R            ; enable commit for Port F
         MOV R0, #0xFF                                ; 1 means allow access
         STR R0, [R1]
         LDR R1, =GPIO_PORTF_AMSEL_R     ; 3) disable analog functionality
   MOV R0, #0                    ; 0 means analog is off
   STR R0, [R1]
   LDR R1, =GPIO_PORTF_PCTL_R      ; 4) configure as GPIO
   MOV R0, #0x00000000           ; 0 means configure Port F as GPIO
   STR R0, [R1]
         LDR R1, = GPIO_PORTF_DIR_R           ; 5) set direction register
         MOV R0, #0x0E                                ; PF0 and PF7-4 input, PF3-1 output
         STR R0, [R1]
         LDR R1, =GPIO_PORTF_AFSEL_R     ; 6) regular port function
   MOV R0, #0                    ; 0 means disable alternate function
   STR R0, [R1]
         LDR R1, = GPIO_PORTF_PUR_R           ; pull-up resistors for PF4, PF0
         MOV R0, #0x11                                ; enable weak pull-up on PF0 and
PF4
```

```
        STR R0, [R1]
        LDR R1, = GPIO_PORTF_DEN_R          ; 7) enable Port F digital port
        MOV R0, #0xFF                               ; 1 means enable digital I/ O
        STR R0, [R1]
        BX LR

                                            ; return R0 with inputs


PortF_Output
   LDR R1, =GPIO_PORTF_DATA_R               ; pointer to Port F data
   STR R0, [R1]                             ; write to PF3-1
   BX  LR

nopressed
        MOV R0, #0                  ; R0 = 0 (no LEDs on)
   BL  PortF_Output
        BL  delay1
        BL  delay1
        BX  LR

rightpressed
        MOV R0, #BLUE               ; R0 = BLUE (blue LED on)
   BL  PortF_Output
        BL  delay0_5
        MOV R0, #0x00
        BL  PortF_Output ; turn the blue LED on
        BL  delay1_5
        BX  LR

leftpressed
        MOV R0, #BLUE               ; R0 = BLUE (blue LED on)
   BL  PortF_Output
        BL  delay1
        MOV R0, #0x00
        BL  PortF_Output                        ; turn the blue LED on
        BL  delay1
        BX  LR

bothpressed
        MOV R0, #BLUE               ; R0 = BLUE (blue LED on)
   BL  PortF_Output
        BL  delay1_5
        MOV R0, #0x00
        BL  PortF_Output                        ; turn the blue LED on
        BL  delay0_5
```

```
        BX  LR

routine
        MOV R0, #(RED+GREEN+BLUE)
   BL  PortF_Output
   BX  LR


MILLISECOND                     EQU 5333333      ; approximately 1ms delay at ~16 MHz
clock
ZEROCINCOMILLISECOND        EQU 2666666      ; approximately 0.5ms delay at ~16 MHz
clock
UMCINCOMILLISECOND          EQU 8000000      ; approximately 1.5ms delay at ~16 MHz
clock

delay1
        LDR R0, =MILLISECOND
delayloop
   SUBS R0, R0, #1                    ; R0 = R0 - 1 (count = count - 1)
   BNE delayloop                  ; if count (R0) != 0, skip to 'delay'
   BX  LR

delay0_5
        LDR R2, =ZEROCINCOMILLISECOND
delay2loop
   SUBS R2, R2, #1                 ; R0 = R0 - 1 (count = count - 1)
   BNE delay2loop                 ; if count (R0) != 0, skip to 'delay'
   BX  LR

delay1_5
        LDR R2, =UMCINCOMILLISECOND
delay3loop
   SUBS R2, R2, #1                 ; R0 = R0 - 1 (count = count - 1)
   BNE delay3loop                 ; if count (R0) != 0, skip to 'delay'
   BX  LR



        ALIGN ; make sure the end of this section is aligned (For the code/data to follow)
        END ; mark end of file
```

Códigos para Laboratório 4:

```
// PLL.c
```

```
/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2014
   Program 2.10, Figure 2.37

 Copyright 2014 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

#include "PLL.h"
#include <stdint.h>
#include "inc/tm4c123gh6pm.h"

// The #define statement SYSDIV2 in PLL.h
// initializes the PLL to the desired frequency.

// bus frequency is 400MHz/(SYSDIV2+1) = 400MHz/(7+1) = 50 MHz
// see the table at the end of this file

#define SYSCTL_RIS_PLLLRIS      0x00000040  // PLL Lock Raw Interrupt Status
#define SYSCTL_RCC_XTAL_M       0x000007C0  // Crystal Value
#define SYSCTL_RCC_XTAL_6MHZ    0x000002C0  // 6 MHz Crystal
#define SYSCTL_RCC_XTAL_8MHZ    0x00000380  // 8 MHz Crystal
#define SYSCTL_RCC_XTAL_16MHZ   0x00000540  // 16 MHz Crystal
#define SYSCTL_RCC2_USERCC2     0x80000000  // Use RCC2
#define SYSCTL_RCC2_DIV400      0x40000000  // Divide PLL as 400 MHz vs. 200
                                            // MHz
#define SYSCTL_RCC2_SYSDIV2_M   0x1F800000  // System Clock Divisor 2
#define SYSCTL_RCC2_SYSDIV2LSB  0x00400000  // Additional LSB for SYSDIV2
#define SYSCTL_RCC2_PWRDN2      0x00002000  // Power-Down PLL 2
#define SYSCTL_RCC2_BYPASS2     0x00000800  // PLL Bypass 2
#define SYSCTL_RCC2_OSCSRC2_M   0x00000070  // Oscillator Source 2
#define SYSCTL_RCC2_OSCSRC2_MO  0x00000000  // MOSC
```

```
// configure the system to get its clock from the PLL
void PLL_Init(void){
  // 0) configure the system to use RCC2 for advanced features
  //    such as 400 MHz PLL and non-integer System Clock Divisor
  SYSCTL_RCC2_R |= SYSCTL_RCC2_USERCC2;
  // 1) bypass PLL while initializing
  SYSCTL_RCC2_R |= SYSCTL_RCC2_BYPASS2;
  // 2) select the crystal value and oscillator source
  SYSCTL_RCC_R &= ~SYSCTL_RCC_XTAL_M;   // clear XTAL field
  SYSCTL_RCC_R += SYSCTL_RCC_XTAL_16MHZ;// configure for 16 MHz crystal
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_OSCSRC2_M;// clear oscillator source field
  SYSCTL_RCC2_R += SYSCTL_RCC2_OSCSRC2_MO;// configure for main oscillator source
  // 3) activate PLL by clearing PWRDN
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_PWRDN2;
  // 4) set the desired system divider and the system divider least significant bit
  SYSCTL_RCC2_R |= SYSCTL_RCC2_DIV400;  // use 400 MHz PLL
  SYSCTL_RCC2_R = (SYSCTL_RCC2_R&~0x1FC00000) // clear system clock divider field
                  + (SYSDIV2<<22);      // configure for 80 MHz clock
  // 5) wait for the PLL to lock by polling PLLLRIS
  while((SYSCTL_RIS_R&SYSCTL_RIS_PLLLRIS)==0){};
  // 6) enable use of PLL by clearing BYPASS
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_BYPASS2;
}
```

```
// PLL.h
// Runs on LM4F120/TM4C123
// A software function to change the bus frequency using the PLL.
// Daniel Valvano
// September 10, 2013

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2014
   Program 2.10, Figure 2.37

 Copyright 2014 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
```

```
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

// The #define statement SYSDIV2 initializes
// the PLL to the desired frequency.
#define SYSDIV2 4
// bus frequency is 400MHz/(SYSDIV2+1) = 400MHz/(4+1) = 80 MHz

// configure the system to get its clock from the PLL
void PLL_Init(void);



// Random.c

#include <stdint.h>
uint32_t M;

void Random_Init(uint32_t seed){
  M = seed;
}

uint32_t Random(void){
  M = 1664525*M+1013904223;
  return(M);
}

// SysTick.c

#include <stdint.h>
#include "inc/tm4c123gh6pm.h"
#define NVIC_ST_CTRL_COUNT      0x00010000  // Count flag
#define NVIC_ST_CTRL_CLK_SRC    0x00000004  // Clock Source
#define NVIC_ST_CTRL_INTEN      0x00000002  // Interrupt enable
#define NVIC_ST_CTRL_ENABLE     0x00000001  // Counter mode
#define NVIC_ST_RELOAD_M        0x00FFFFFF  // Counter load value

// Initialize SysTick with busy wait running at bus clock.
void SysTick_Init(void){
  NVIC_ST_CTRL_R = 0;                   // disable SysTick during setup
  NVIC_ST_RELOAD_R = NVIC_ST_RELOAD_M;  // maximum reload value
  NVIC_ST_CURRENT_R = 0;                // any write to current clears it
```

```
                                // enable SysTick with core clock
  NVIC_ST_CTRL_R = NVIC_ST_CTRL_ENABLE+NVIC_ST_CTRL_CLK_SRC;
}
// Time delay using busy wait.
// The delay parameter is in units of the core clock. (units of 20 nsec for 50 MHz clock)
void SysTick_Wait(uint32_t delay){
  volatile uint32_t elapsedTime;
  uint32_t startTime = NVIC_ST_CURRENT_R;
  do{
    elapsedTime = (startTime-NVIC_ST_CURRENT_R)&0x00FFFFFF;
  }
  while(elapsedTime <= delay);
}
// Time delay using busy wait.
// This assumes 50 MHz system clock.
void SysTick_Wait10ms(uint32_t delay){
  uint32_t i;
  for(i=0; i<delay; i++){
    SysTick_Wait(500000);  // wait 10ms (assumes 50 MHz clock)
  }
}
```

Códigos para Laboratório 5 e 6:

```
; UART_ASM.s

; comment out the C versions of these functions and add this file to your project
      AREA    |.text|, CODE, READONLY

;*****************************************************************************
*
;
; Let functions be called in other modules.
;
;*****************************************************************************
*
      EXPORT  UART_InChar
      EXPORT  UART_OutChar


;********** UART_InChar ***************
;// Wait for new serial port input
;// Input: none
;// Output: ASCII code for key typed
```

```
UART_InChar

        ; Place your solution here

    BX  LR

;//------------UART_OutChar------------
;// Output 8-bit to serial port
;// Input: letter is an 8-bit ASCII character to be transferred
;// Output: none
UART_OutChar

        ; Place your solution here

    BX  LR

;*****************************************************************************
*
;
; Tell the assembler that we're done.
;
;*****************************************************************************
*
    ALIGN
    END

// UART.c

#include <stdint.h>
#include "UART.h"
#include "tm4c123gh6pm.h"


#define UART_FR_TXFF            0x00000020  // UART Transmit FIFO Full
#define UART_FR_RXFE            0x00000010  // UART Receive FIFO Empty
#define UART_LCRH_WLEN_8        0x00000060  // 8 bit word length
#define UART_LCRH_FEN           0x00000010  // UART Enable FIFOs
#define UART_CTL_UARTEN         0x00000001  // UART Enable


void UART_Init(void){
  SYSCTL_RCGCUART_R |= 0x01;            // activate UART0
  SYSCTL_RCGCGPIO_R |= 0x01;            // activate port A
  while((SYSCTL_PRGPIO_R&0x01) == 0){};
  UART0_CTL_R &= ~UART_CTL_UARTEN;      // disable UART
  UART0_IBRD_R = 27;                    // IBRD = int(50,000,000 / (16 * 115,200)) = int(27.1267)
```

```
  UART0_FBRD_R = 8;                 // FBRD = int(0.1267 * 64 + 0.5) = 8
                                    // 8 bit word length (no parity bits, one stop bit, FIFOs)
  UART0_LCRH_R = (UART_LCRH_WLEN_8|UART_LCRH_FEN);
  UART0_CTL_R |= UART_CTL_UARTEN;       // enable UART
  GPIO_PORTA_AFSEL_R |= 0x03;           // enable alt funct on PA1-0
  GPIO_PORTA_DEN_R |= 0x03;             // enable digital I/O on PA1-0
                                    // configure PA1-0 as UART
  GPIO_PORTA_PCTL_R = (GPIO_PORTA_PCTL_R&0xFFFFFF00)+0x00000011;
  GPIO_PORTA_AMSEL_R &= ~0x03;          // disable analog functionality on PA
}


char UART_InChar(void){
  while((UART0_FR_R&UART_FR_RXFE) != 0);
  return((char)(UART0_DR_R&0xFF));
}

void UART_OutChar(char data){
  while((UART0_FR_R&UART_FR_TXFF) != 0);
  UART0_DR_R = data;
}



void UART_OutString(char *pt){
  while(*pt){
    UART_OutChar(*pt);
    pt++;
  }
}


uint32_t UART_InUDec(void){
uint32_t number=0, length=0;
char character;
  character = UART_InChar();
  while(character != CR){ // accepts until <enter> is typed
// The next line checks that the input is a digit, 0-9.
// If the character is not 0-9, it is ignored and not echoed
    if((character>='0') && (character<='9')) {
      number = 10*number+(character-'0');   // this line overflows if above 4294967295
      length++;
      UART_OutChar(character);
    }
// If the input is a backspace, then the return number is
// changed and a backspace is outputted to the screen
```

```
    else if((character==BS) && length){
      number /= 10;
      length--;
      UART_OutChar(character);
    }
    character = UART_InChar();
  }
  return number;
}


void UART_OutUDec(uint32_t n){
// This function uses recursion to convert decimal number
//   of unspecified length as an ASCII string
  if(n >= 10){
    UART_OutUDec(n/10);
    n = n%10;
  }
  UART_OutChar(n+'0'); /* n is between 0 and 9 */
}

uint32_t UART_InUHex(void){
uint32_t number=0, digit, length=0;
char character;
  character = UART_InChar();
  while(character != CR){
    digit = 0x10; // assume bad
    if((character>='0') && (character<='9')){
      digit = character-'0';
    }
    else if((character>='A') && (character<='F')){
      digit = (character-'A')+0xA;
    }
    else if((character>='a') && (character<='f')){
      digit = (character-'a')+0xA;
    }
// If the character is not 0-9 or A-F, it is ignored and not echoed
    if(digit <= 0xF){
      number = number*0x10+digit;
      length++;
      UART_OutChar(character);
    }
// Backspace outputted and return value changed if a backspace is inputted
    else if((character==BS) && length){
      number /= 0x10;
      length--;
```

```
      UART_OutChar(character);
    }
    character = UART_InChar();
  }
  return number;
}

void UART_OutUHex(uint32_t number){
// This function uses recursion to convert the number of
//   unspecified length as an ASCII string
  if(number >= 0x10){
    UART_OutUHex(number/0x10);
    UART_OutUHex(number%0x10);
  }
  else{
    if(number < 0xA){
      UART_OutChar(number+'0');
     }
    else{
      UART_OutChar((number-0x0A)+'A');
    }
  }
}

void UART_InString(char *bufPt, uint16_t max) {
int length=0;
char character;
  character = UART_InChar();
  while(character != CR){
    if(character == BS){
      if(length){
        bufPt--;
        length--;
        UART_OutChar(BS);
      }
    }
    else if(length < max){
      *bufPt = character;
      bufPt++;
      length++;
      UART_OutChar(character);
    }
    character = UART_InChar();
  }
  *bufPt = 0;
}
```

Códigos para Laboratório 7:

```
// PeriodicSysTickInts.c

#include <stdint.h>
#include "inc/tm4c123gh6pm.h"

#include "SysTickInts.h"
#include "PLL.h"


#define PF2     (*((volatile uint32_t *)0x40025010))


void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value
void WaitForInterrupt(void);  // low power mode
volatile uint32_t Counts = 0;

int main(void){
  PLL_Init();               // bus clock at 80 MHz
  SYSCTL_RCGCGPIO_R |= 0x20;  // activate port F
  Counts = 0;
  GPIO_PORTF_DIR_R |= 0x04;   // make PF2 output (PF2 built-in LED)
  GPIO_PORTF_AFSEL_R &= ~0x04;// disable alt funct on PF2
  GPIO_PORTF_DEN_R |= 0x04;   // enable digital I/O on PF2
                              // configure PF2 as GPIO
  GPIO_PORTF_PCTL_R = (GPIO_PORTF_PCTL_R&0xFFFFF0FF)+0x00000000;
  GPIO_PORTF_AMSEL_R = 0;     // disable analog functionality on PF
  SysTick_Init(80000);        // initialize SysTick timer
  EnableInterrupts();

  while(1){                 // interrupts every 1ms, 500 Hz flash
    WaitForInterrupt();
  }
}

// Interrupt service routine
// Executed every 12.5ns*(period)
void SysTick_Handler(void){
  PF2 ^= 0x04;              // toggle PF2
  PF2 ^= 0x04;              // toggle PF2
  Counts = Counts + 1;
  PF2 ^= 0x04;              // toggle PF2
```

```
}

// SysTickInts.c

#include <stdint.h>
#include "inc/tm4c123gh6pm.h"
#include "SysTickInts.h"

#define NVIC_ST_CTRL_CLK_SRC    0x00000004  // Clock Source
#define NVIC_ST_CTRL_INTEN      0x00000002  // Interrupt enable
#define NVIC_ST_CTRL_ENABLE     0x00000001  // Counter mode

void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value
void WaitForInterrupt(void);  // low power mode

void SysTick_Init(uint32_t period){long sr;
  sr = StartCritical();
  NVIC_ST_CTRL_R = 0;         // disable SysTick during setup
  NVIC_ST_RELOAD_R = period-1;// reload value
  NVIC_ST_CURRENT_R = 0;      // any write to current clears it
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000; // priority 2
                        // enable SysTick with core clock and interrupts
  NVIC_ST_CTRL_R                                                              =
NVIC_ST_CTRL_ENABLE+NVIC_ST_CTRL_CLK_SRC+NVIC_ST_CTRL_INTEN;
  EndCritical(sr);
}
```

Códigos para Laboratório 8:

```
// ADCSWTrigger.c

#include <stdint.h>
#include "inc/tm4c123gh6pm.h"


void ADC0_InitSWTriggerSeq3_Ch9(void){ volatile uint32_t delay;
                        // 1) activate clock for Port E
  SYSCTL_RCGCGPIO_R |= 0x10;
  while((SYSCTL_PRGPIO_R&0x10) != 0x10){};
  delay = SYSCTL_RCGCGPIO_R;      //   allow time for clock to stabilize
  delay = SYSCTL_RCGCGPIO_R;
```

```
  GPIO_PORTE_DIR_R &= ~0x10;      // 2) make PE4 input
  GPIO_PORTE_AFSEL_R |= 0x10;     // 3) enable alternate function on PE4
  GPIO_PORTE_DEN_R &= ~0x10;      // 4) disable digital I/O on PE4
  GPIO_PORTE_AMSEL_R |= 0x10;     // 5) enable analog functionality on PE4
  SYSCTL_RCGCADC_R |= 0x0001;   // 7) activate ADC0
//  while((SYSCTL_PRADC_R&0x0001) != 0x0001){};    // good code, but not yet implemented
in simulator
  delay = SYSCTL_RCGCADC_R;        // extra time for clock to stabilize
  delay = SYSCTL_RCGCADC_R;        // extra time for clock to stabilize
  delay = SYSCTL_RCGCADC_R;        // extra time for clock to stabilize
  delay = SYSCTL_RCGCADC_R;

  ADC0_PC_R &= ~0xF;              // 7) clear max sample rate field
  ADC0_PC_R |= 0x1;               //    configure for 125K samples/sec
  ADC0_SSPRI_R = 0x0123;          // 8) Sequencer 3 is highest priority
  ADC0_ACTSS_R &= ~0x0008;        // 9) disable sample sequencer 3
  ADC0_EMUX_R &= ~0xF000;         // 10) seq3 is software trigger
  ADC0_SSMUX3_R &= ~0x000F;       // 11) clear SS3 field
  ADC0_SSMUX3_R += 9;             //    set channel
  ADC0_SSCTL3_R = 0x0006;         // 12) no TS0 D0, yes IE0 END0
  ADC0_IM_R &= ~0x0008;           // 13) disable SS3 interrupts
  ADC0_ACTSS_R |= 0x0008;         // 14) enable sample sequencer 3
}


void ADC0_InitSWTriggerSeq3(uint32_t channelNum){ volatile uint32_t delay;
  switch(channelNum){             // 1) activate clock
    case 0:
    case 1:
    case 2:
    case 3:
    case 8:
    case 9:                       //    these are on GPIO_PORTE
      SYSCTL_RCGCGPIO_R |= 0x10;
      while((SYSCTL_PRGPIO_R&0x10) != 0x10){};
      break;
    case 4:
    case 5:
    case 6:
    case 7:                       //    these are on GPIO_PORTD
      SYSCTL_RCGCGPIO_R |= 0x08;
      while((SYSCTL_PRGPIO_R&0x08) != 0x08){};
      break;
    case 10:
    case 11:                      //    these are on GPIO_PORTB
      SYSCTL_RCGCGPIO_R |= 0x02;
```

```
    while((SYSCTL_PRGPIO_R&0x02) != 0x02){};
    break;
  default: return;              //    0 to 11 are valid channels on the LM4F120/TM4C123
}
delay = SYSCTL_RCGCGPIO_R;      // extra time for clock to stabilize
delay = SYSCTL_RCGCGPIO_R;
switch(channelNum){
  case 0:                       //      Ain0 is on PE3
    GPIO_PORTE_DIR_R &= ~0x08;  // 3.0) make PE3 input
    GPIO_PORTE_AFSEL_R |= 0x08; // 4.0) enable alternate function on PE3
    GPIO_PORTE_DEN_R &= ~0x08;  // 5.0) disable digital I/O on PE3
    GPIO_PORTE_AMSEL_R |= 0x08; // 6.0) enable analog functionality on PE3
    break;
  case 1:                       //      Ain1 is on PE2
    GPIO_PORTE_DIR_R &= ~0x04;  // 3.1) make PE2 input
    GPIO_PORTE_AFSEL_R |= 0x04; // 4.1) enable alternate function on PE2
    GPIO_PORTE_DEN_R &= ~0x04;  // 5.1) disable digital I/O on PE2
    GPIO_PORTE_AMSEL_R |= 0x04; // 6.1) enable analog functionality on PE2
    break;
  case 2:                       //      Ain2 is on PE1
    GPIO_PORTE_DIR_R &= ~0x02;  // 3.2) make PE1 input
    GPIO_PORTE_AFSEL_R |= 0x02; // 4.2) enable alternate function on PE1
    GPIO_PORTE_DEN_R &= ~0x02;  // 5.2) disable digital I/O on PE1
    GPIO_PORTE_AMSEL_R |= 0x02; // 6.2) enable analog functionality on PE1
    break;
  case 3:                       //      Ain3 is on PE0
    GPIO_PORTE_DIR_R &= ~0x01;  // 3.3) make PE0 input
    GPIO_PORTE_AFSEL_R |= 0x01; // 4.3) enable alternate function on PE0
    GPIO_PORTE_DEN_R &= ~0x01;  // 5.3) disable digital I/O on PE0
    GPIO_PORTE_AMSEL_R |= 0x01; // 6.3) enable analog functionality on PE0
    break;
  case 4:                       //      Ain4 is on PD3
    GPIO_PORTD_DIR_R &= ~0x08;  // 3.4) make PD3 input
    GPIO_PORTD_AFSEL_R |= 0x08; // 4.4) enable alternate function on PD3
    GPIO_PORTD_DEN_R &= ~0x08;  // 5.4) disable digital I/O on PD3
    GPIO_PORTD_AMSEL_R |= 0x08; // 6.4) enable analog functionality on PD3
    break;
  case 5:                       //      Ain5 is on PD2
    GPIO_PORTD_DIR_R &= ~0x04;  // 3.5) make PD2 input
    GPIO_PORTD_AFSEL_R |= 0x04; // 4.5) enable alternate function on PD2
    GPIO_PORTD_DEN_R &= ~0x04;  // 5.5) disable digital I/O on PD2
    GPIO_PORTD_AMSEL_R |= 0x04; // 6.5) enable analog functionality on PD2
    break;
  case 6:                       //      Ain6 is on PD1
    GPIO_PORTD_DIR_R &= ~0x02;  // 3.6) make PD1 input
    GPIO_PORTD_AFSEL_R |= 0x02; // 4.6) enable alternate function on PD1
```

```
    GPIO_PORTD_DEN_R &= ~0x02;  // 5.6) disable digital I/O on PD1
    GPIO_PORTD_AMSEL_R |= 0x02; // 6.6) enable analog functionality on PD1
    break;
  case 7:                   //      Ain7 is on PD0
    GPIO_PORTD_DIR_R &= ~0x01;  // 3.7) make PD0 input
    GPIO_PORTD_AFSEL_R |= 0x01; // 4.7) enable alternate function on PD0
    GPIO_PORTD_DEN_R &= ~0x01;  // 5.7) disable digital I/O on PD0
    GPIO_PORTD_AMSEL_R |= 0x01; // 6.7) enable analog functionality on PD0
    break;
  case 8:                   //      Ain8 is on PE5
    GPIO_PORTE_DIR_R &= ~0x20;  // 3.8) make PE5 input
    GPIO_PORTE_AFSEL_R |= 0x20; // 4.8) enable alternate function on PE5
    GPIO_PORTE_DEN_R &= ~0x20;  // 5.8) disable digital I/O on PE5
    GPIO_PORTE_AMSEL_R |= 0x20; // 6.8) enable analog functionality on PE5
    break;
  case 9:                   //      Ain9 is on PE4
    GPIO_PORTE_DIR_R &= ~0x10;  // 3.9) make PE4 input
    GPIO_PORTE_AFSEL_R |= 0x10; // 4.9) enable alternate function on PE4
    GPIO_PORTE_DEN_R &= ~0x10;  // 5.9) disable digital I/O on PE4
    GPIO_PORTE_AMSEL_R |= 0x10; // 6.9) enable analog functionality on PE4
    break;
  case 10:                  //       Ain10 is on PB4
    GPIO_PORTB_DIR_R &= ~0x10;  // 3.10) make PB4 input
    GPIO_PORTB_AFSEL_R |= 0x10; // 4.10) enable alternate function on PB4
    GPIO_PORTB_DEN_R &= ~0x10;  // 5.10) disable digital I/O on PB4
    GPIO_PORTB_AMSEL_R |= 0x10; // 6.10) enable analog functionality on PB4
    break;
  case 11:                  //       Ain11 is on PB5
    GPIO_PORTB_DIR_R &= ~0x20;  // 3.11) make PB5 input
    GPIO_PORTB_AFSEL_R |= 0x20; // 4.11) enable alternate function on PB5
    GPIO_PORTB_DEN_R &= ~0x20;  // 5.11) disable digital I/O on PB5
    GPIO_PORTB_AMSEL_R |= 0x20; // 6.11) enable analog functionality on PB5
    break;
 }
 SYSCTL_RCGCADC_R |= 0x0001;   // 7) activate ADC0
//  while((SYSCTL_PRADC_R&0x0001) != 0x0001){};   // good code, but not yet implemented
in simulator
 delay = SYSCTL_RCGCADC_R;        // extra time for clock to stabilize
 delay = SYSCTL_RCGCADC_R;        // extra time for clock to stabilize
 delay = SYSCTL_RCGCADC_R;        // extra time for clock to stabilize
 delay = SYSCTL_RCGCADC_R;
 ADC0_PC_R &= ~0xF;              // 9) clear max sample rate field
 ADC0_PC_R |= 0x1;               //    configure for 125K samples/sec
 ADC0_SSPRI_R = 0x3210;          // 10) Sequencer 3 is lowest priority
 ADC0_ACTSS_R &= ~0x0008;        // 11) disable sample sequencer 3
 ADC0_EMUX_R &= ~0xF000;         // 12) seq3 is software trigger
```

```
  ADC0_SSMUX3_R &= ~0x000F;       // 13) clear SS3 field
  ADC0_SSMUX3_R += channelNum;    //     set channel
  ADC0_SSCTL3_R = 0x0006;         // 14) no TS0 D0, yes IE0 END0
  ADC0_IM_R &= ~0x0008;           // 15) disable SS3 interrupts
  ADC0_ACTSS_R |= 0x0008;         // 16) enable sample sequencer 3
}

void ADC0_InitAllTriggerSeq3(uint32_t channelNum){ volatile uint32_t delay;
  switch(channelNum){             // 1) activate clock
    case 0:
    case 1:
    case 2:
    case 3:
    case 8:
    case 9:                       //    these are on GPIO_PORTE
      SYSCTL_RCGCGPIO_R |= 0x10;
      while((SYSCTL_PRGPIO_R&0x10) != 0x10){};
      break;
    case 4:
    case 5:
    case 6:
    case 7:                       //    these are on GPIO_PORTD
      SYSCTL_RCGCGPIO_R |= 0x08;
      while((SYSCTL_PRGPIO_R&0x08) != 0x08){};
      break;
    case 10:
    case 11:                      //    these are on GPIO_PORTB
      SYSCTL_RCGCGPIO_R |= 0x02;
      while((SYSCTL_PRGPIO_R&0x02) != 0x02){};
      break;
    default: return;              //    0 to 11 are valid channels on the LM4F120/TM4C123
  }
  delay = SYSCTL_RCGCGPIO_R;      // 2) extra time for clock to stabilize
  delay = SYSCTL_RCGCGPIO_R;
  switch(channelNum){
    case 0:                       //      Ain0 is on PE3
      GPIO_PORTE_DIR_R &= ~0x08;  // 3.0) make PE3 input
      GPIO_PORTE_AFSEL_R |= 0x08; // 4.0) enable alternate function on PE3
      GPIO_PORTE_DEN_R &= ~0x08;  // 5.0) disable digital I/O on PE3
      GPIO_PORTE_AMSEL_R |= 0x08; // 6.0) enable analog functionality on PE3
      break;
    case 1:                       //      Ain1 is on PE2
      GPIO_PORTE_DIR_R &= ~0x04;  // 3.1) make PE2 input
      GPIO_PORTE_AFSEL_R |= 0x04; // 4.1) enable alternate function on PE2
      GPIO_PORTE_DEN_R &= ~0x04;  // 5.1) disable digital I/O on PE2
      GPIO_PORTE_AMSEL_R |= 0x04; // 6.1) enable analog functionality on PE2
```

```
  break;
case 2:                 //      Ain2 is on PE1
  GPIO_PORTE_DIR_R &= ~0x02;  // 3.2) make PE1 input
  GPIO_PORTE_AFSEL_R |= 0x02; // 4.2) enable alternate function on PE1
  GPIO_PORTE_DEN_R &= ~0x02;  // 5.2) disable digital I/O on PE1
  GPIO_PORTE_AMSEL_R |= 0x02; // 6.2) enable analog functionality on PE1
  break;
case 3:                 //      Ain3 is on PE0
  GPIO_PORTE_DIR_R &= ~0x01;  // 3.3) make PE0 input
  GPIO_PORTE_AFSEL_R |= 0x01; // 4.3) enable alternate function on PE0
  GPIO_PORTE_DEN_R &= ~0x01;  // 5.3) disable digital I/O on PE0
  GPIO_PORTE_AMSEL_R |= 0x01; // 6.3) enable analog functionality on PE0
  break;
case 4:                 //      Ain4 is on PD3
  GPIO_PORTD_DIR_R &= ~0x08;  // 3.4) make PD3 input
  GPIO_PORTD_AFSEL_R |= 0x08; // 4.4) enable alternate function on PD3
  GPIO_PORTD_DEN_R &= ~0x08;  // 5.4) disable digital I/O on PD3
  GPIO_PORTD_AMSEL_R |= 0x08; // 6.4) enable analog functionality on PD3
  break;
case 5:                 //      Ain5 is on PD2
  GPIO_PORTD_DIR_R &= ~0x04;  // 3.5) make PD2 input
  GPIO_PORTD_AFSEL_R |= 0x04; // 4.5) enable alternate function on PD2
  GPIO_PORTD_DEN_R &= ~0x04;  // 5.5) disable digital I/O on PD2
  GPIO_PORTD_AMSEL_R |= 0x04; // 6.5) enable analog functionality on PD2
  break;
case 6:                 //      Ain6 is on PD1
  GPIO_PORTD_DIR_R &= ~0x02;  // 3.6) make PD1 input
  GPIO_PORTD_AFSEL_R |= 0x02; // 4.6) enable alternate function on PD1
  GPIO_PORTD_DEN_R &= ~0x02;  // 5.6) disable digital I/O on PD1
  GPIO_PORTD_AMSEL_R |= 0x02; // 6.6) enable analog functionality on PD1
  break;
case 7:                 //      Ain7 is on PD0
  GPIO_PORTD_DIR_R &= ~0x01;  // 3.7) make PD0 input
  GPIO_PORTD_AFSEL_R |= 0x01; // 4.7) enable alternate function on PD0
  GPIO_PORTD_DEN_R &= ~0x01;  // 5.7) disable digital I/O on PD0
  GPIO_PORTD_AMSEL_R |= 0x01; // 6.7) enable analog functionality on PD0
  break;
case 8:                 //      Ain8 is on PE5
  GPIO_PORTE_DIR_R &= ~0x20;  // 3.8) make PE5 input
  GPIO_PORTE_AFSEL_R |= 0x20; // 4.8) enable alternate function on PE5
  GPIO_PORTE_DEN_R &= ~0x20;  // 5.8) disable digital I/O on PE5
  GPIO_PORTE_AMSEL_R |= 0x20; // 6.8) enable analog functionality on PE5
  break;
case 9:                 //      Ain9 is on PE4
  GPIO_PORTE_DIR_R &= ~0x10;  // 3.9) make PE4 input
  GPIO_PORTE_AFSEL_R |= 0x10; // 4.9) enable alternate function on PE4
```

```
    GPIO_PORTE_DEN_R &= ~0x10;  // 5.9) disable digital I/O on PE4
    GPIO_PORTE_AMSEL_R |= 0x10; // 6.9) enable analog functionality on PE4
    break;
  case 10:                  //       Ain10 is on PB4
    GPIO_PORTB_DIR_R &= ~0x10;  // 3.10) make PB4 input
    GPIO_PORTB_AFSEL_R |= 0x10; // 4.10) enable alternate function on PB4
    GPIO_PORTB_DEN_R &= ~0x10;  // 5.10) disable digital I/O on PB4
    GPIO_PORTB_AMSEL_R |= 0x10; // 6.10) enable analog functionality on PB4
    break;
  case 11:                  //       Ain11 is on PB5
    GPIO_PORTB_DIR_R &= ~0x20;  // 3.11) make PB5 input
    GPIO_PORTB_AFSEL_R |= 0x20; // 4.11) enable alternate function on PB5
    GPIO_PORTB_DEN_R &= ~0x20;  // 5.11) disable digital I/O on PB5
    GPIO_PORTB_AMSEL_R |= 0x20; // 6.11) enable analog functionality on PB5
    break;
 }
 SYSCTL_RCGCADC_R |= 0x0001;   // 7) activate ADC0
//  while((SYSCTL_PRADC_R&0x0001) != 0x0001){};   // good code, but not yet implemented
in simulator
 delay = SYSCTL_RCGCADC_R;        // extra time for clock to stabilize
 delay = SYSCTL_RCGCADC_R;        // extra time for clock to stabilize
 delay = SYSCTL_RCGCADC_R;        // extra time for clock to stabilize
 delay = SYSCTL_RCGCADC_R;
 ADC0_PC_R &= ~0xF;              // 9) clear max sample rate field
 ADC0_PC_R |= 0x1;               //    configure for 125K samples/sec
 ADC0_SSPRI_R = 0x3210;          // 10) Sequencer 3 is lowest priority
 ADC0_ACTSS_R &= ~0x0008;        // 11) disable sample sequencer 3
 ADC0_EMUX_R |= 0xF000;          // 12) seq3 is continuous trigger
 ADC0_SSMUX3_R &= ~0x000F;       // 13) clear SS3 field
 ADC0_SSMUX3_R += channelNum;    //     set channel
 ADC0_SSCTL3_R = 0x0006;         // 14) no TS0 D0, yes IE0 END0
 ADC0_IM_R &= ~0x0008;           // 15) disable SS3 interrupts
 ADC0_ACTSS_R |= 0x0008;         // 16) enable sample sequencer 3
}

//------------ADC0_InSeq3------------
// Busy-wait Analog to digital conversion
// Input: none
// Output: 12-bit result of ADC conversion
uint32_t ADC0_InSeq3(void){  uint32_t result;
 ADC0_PSSI_R = 0x0008;           // 1) initiate SS3
 while((ADC0_RIS_R&0x08)==0){};   // 2) wait for conversion done
   // if you have an A0-A3 revision number, you need to add an 8 usec wait here
 result = ADC0_SSFIFO3_R&0xFFF;   // 3) read result
 ADC0_ISC_R = 0x0008;            // 4) acknowledge completion
 return result;}
```