UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

GUILHERME WAGNER ECKERT

PROJETO DE HARDWARE E SOFTWARE PARA A PLATAFORMA FRDM-KL25Z

Porto Alegre
2016

GUILHERME WAGNER ECKERT

PROJETO DE HARDWARE E SOFTWARE PARA A PLATAFORMA FRDM-KL25Z

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para Graduação em Engenharia Elétrica.

Orientador: Prof. Me. Alexandre Ambrozi Junqueira

Porto Alegre

2016

GUILHERME WAGNER ECKERT

PROJETO DE HARDWARE E SOFTWARE PARA A PLATAFORMA FRDM-KL25Z

Este projeto foi analisado e julgado adequado para a obtenção do título de Bacharel em Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora designada pelo Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul.

_____
Prof. Me. Alexandre Ambrozi Junqueira

_____
Prof Dr. Ály Ferreira Flores Filho

Aprovado em: ____/____/____

BANCA EXAMINADORA

Prof. Dr. Adalberto Schuck Júnior – UFRGS _____

Prof. Me. Alexandre Ambrozi Junqueira – UFRGS _____

Prof. Dr. Tiago Roberto Balen – UFRGS _____

Dedico este trabalho a Deus, por ter me sustentado até aqui, e aos meus pais, por todo empenho que fizeram para tornar isto uma realidade.

AGRADECIMENTOS

A Deus, que sempre esteve ao meu lado, independentemente da minha fé.

À minha esposa Luana, por ter me dado ânimo e força durante o curso, além de viver as emoções de cada momento juntamente comigo.

Aos meus pais, Francisco e Rosana, que sempre investiram na minha formação e apoiaram as minhas escolhas.

Aos meus irmãos, que proporcionaram descontração e alegria, principalmente nas horas mais difíceis.

Aos amigos que fiz durante a graduação, pelas discussões sobre o conteúdo das aulas, muitas vezes acaloradas, de modo a me fazer refletir profundamente sobre vários temas do curso.

Ao meu orientador, professor Junqueira, pelo companheirismo, dicas e material fornecido para o desenvolvimento deste trabalho.

O que adquire a sabedoria é amigo de si mesmo; o que cultiva a inteligência achará o bem.

Salomão

RESUMO

Este trabalho consiste na elaboração de uma interface de usuário através do projeto de *hardware* e *software* para uso em conjunto com a plataforma FRDM-KL25Z, a qual integra um microcontrolador com núcleo de processador ARM Cortex-M0+. O objetivo é de disponibilizar as ferramentas desenvolvidas para os futuros estudantes da disciplina de Microprocessadores I ministrada na Universidade Federal do Rio Grande do Sul, para que assim possam elaborar as competências de prototipação e programação de sistemas embarcados. O método utilizado para a elaboração do projeto consiste na descrição sobre o *hardware* que integra a FRDM-KL25Z e seus recursos de programação, além da análise dos custos envolvidos e das necessidades de ensino propostas pela disciplina. É possível dizer que se obteve sucesso na execução deste trabalho, visto que a interface de usuário funcionou exatamente conforme o seu projeto.

Palavras-chave: FRDM-KL25Z. Interface de usuário. Microcontrolador.

ABSTRACT

This work consists of the elaboration of a user interface through the design of hardware and software for use in conjunction with the FRDM-KL25Z platform, which integrates a microcontroller with an ARM Cortex-M0+ processor core. The objective is to make available this tools for the future students of the class of Microprocessors I at the Federal University of Rio Grande do Sul, so they can develop their prototyping and programming skills of embedded systems. The design method used consists in the description of the hardware that integrates the FRDM-KL25Z and its programming resources, as well as the analysis of the costs involved and the teaching needs proposed to the class. It can be said that the execution of this work was successful, since the user interface worked exactly according to the project.

Keywords: FRDM-KL25Z. User interface. Microcontroller.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ADC	*Analog-to-Digital Converter*
ARM	*Advanced* RISC *Machine*
CDC	*Communications Device Class*
CI	Circuito Integrado
CISC	*Complex Instruction Set Computer*
CPU	*Central Processing Unit*
CSP	*Chip-Scale Package*
DAC	*Digital-to-Analog Converter*
DIP	*Dual In-line Package*
DMA	*Direct Memory Access*
DNP	*Do Not Place*
DSP	*Digital Signal Processor*
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
FTM	*FlexTimer Module*
GDB	GNU *Debugger*
GPIO	*General-Purpose Input*/*Output*
HAL	*Hardware Abstraction Layer*
IDE	*Integrated Development Environment*
I^2C	*Inter-Integrated Circuit*
I^2S	*Integrated Interchip Sound*
I/O	*Input*/*Output*
IU	Interface de Usuário
JTAG	*Joint Test Action Group*
KDS	Kinetis *Design Studio*
KSDK	Kinetis *Software Development Kit*
LCD	*Liquid Crystal Display*
LED	*Light-Emitting Diode*
LQFP	*Low Profile Quad Flat Package*
MAC	*Multiply-Accumulate*
MCU	*Microcontroller Unit*
MMU	*Memory Management Unit*
MPU	*Memory Protection Unit*

MSD	*Mass Storage Device*
MTB	*Micro Trace Buffer*
NC	*Not Connected*
NMI	*Non-Maskable Interrupt*
NO	*Normally-Open*
NVIC	*Nested Vector Interrupt Controller*
OpenSDA	*Open-standard Serial and Debug Adapter*
OSA	*Operating System Abstraction layer*
OTG	*On-The-Go*
PCI	Placa de Circuito Impresso
PIT	*Periodic Interrupt Timer*
PN	*Part Number*
PWM	*Pulse Width Modulation*
RAM	*Random Access Memory*
RISC	*Reduced Instruction Set Computer*
RGB	*Red-Green-Blue*
RTC	*Real-Time Clock*
SAR	*Successive Aproximation Register*
SCL	*Serial Clock line*
SDA	*Serial Data line*
SPI	*Serial Peripheral Interface*
SPST	*Single Pole*, *Single Throw*
SRAM	*Static Random Access Memory*
SWD	*Serial Wire Debug*
TFS	*Thin Film Storage*
TH	*Through Hole*
TO	*Transistor Outline*
TPM	*Timer*/PWM *Module*
TQFP	*Thin Quad Flat Package*
TSI	*Touch Sensing Input*
TSS	*Touch Sensing Software*
UART	*Universal Asynchronous Receiver*/*Transmitter*
USART	*Universal Synchronous*/*Asynchronous Receiver*/*Transmitter*
USB	*Universal Serial Bus*

SUMÁRIO

1 INTRODUÇÃO

Este trabalho consiste na elaboração de uma interface de usuário através do projeto de *hardware* e *software* para uso em conjunto com a plataforma de desenvolvimento FRDM-KL25Z, produzida e distribuída originalmente pela empresa Freescale Semiconductor e posteriormente (até a atualidade da síntese deste trabalho) pela empresa NXP Semiconductors[1]. A plataforma FRDM-KL25Z é constituída por uma PCI que integra um microcontrolador e outros componentes eletrônicos, a qual pode ser utilizada em conjunto com programas pré-configurados e fornecidos pela NXP que demonstram e aplicam diversas funcionalidades da plataforma.

O objetivo deste projeto é de disponibilizar as ferramentas desenvolvidas para os futuros estudantes da disciplina de Microprocessadores I ministrada na Universidade Federal do Rio Grande do Sul, para que assim possam elaborar suas competências de prototipação e programação de sistemas embarcados. Dessa forma, os requisitos de projeto envolvem principalmente a busca pelos menores custos dos componentes e a simplicidade em sua implementação, visto que serão produzidas oito (ou mais) placas para uso em laboratório. Também se é considerada a utilização de recursos relacionados ao conteúdo da disciplina.

A FRDM-KL25Z integra um microcontrolador da subfamília KL25, de núcleo ARM Cortex-M0+ (muito usado na indústria moderna), o qual contrasta com o microcontrolador que é atualmente utilizado nas atividades de laboratório de Microprocessadores I, o AT89S52 da antiga família Intel 8051. As diferenças, vantagens e desvantagens entre estes microcontroladores são exploradas no capítulo 2.

A FRDM-KL25Z faz parte de um conjunto de plataformas de desenvolvimento da NXP denominado Freedom Development Boards que, segundo a NXP (*homepage* da FRDM-KL25Z, 2016), é perfeito para a rápida prototipação de aplicações e também para a demonstração dos microcontroladores da família Kinetis (desenvolvidos também pela NXP). As características desta família de MCUs estão apresentadas através do capítulo 2, com enfoque dado para a subfamília KL25.

[1]Em Dezembro de 2015 a NXP Semiconductors completou a aquisição da Freescale Semiconductor por aproximadamente $11,8 bilhões. Em Outubro de 2016 a empresa Qualcomm anunciou que compraria a NXP por $39 bilhões em um futuro próximo.

Ainda, segundo a NXP, essas plataformas oferecem baixo custo, forma compacta no padrão industrial, fácil acesso dos pinos do microcontrolador, interfaces de depuração e serial, além da compatibilidade com placas para expansão de *hardware* de empresas terceiras.

No capítulo 2 é abordada toda a descrição sobre o *hardware* que integra a FRDM-KL25Z, a comparação dos atributos de seu MCU em relação a outros disponíveis no mercado e a compatibilidade com as placas Arduino Shields[2] para a expansão de *hardware* (apenas Shields com leiaute de pinos do tipo Arduino R3).

No capítulo 3 são explicados e comparados os recursos de programação disponíveis para a escrita, compilação, ligação, gravação e depuração dos códigos. Pelo fato da IDE KDS da NXP fornecer todas estas ferramentas gratuitamente foi desenvolvido um tutorial para a sua instalação, configuração e uso. Ao final do capítulo, é feita a execução e descrição breve de um programa exemplo fornecido pela NXP para uso direto com a FRDM-KL25Z.

No quarto capítulo é discutida a confecção de uma placa de circuito para uso como interface de usuário quando conectada à FRDM-KL25Z.

No quinto capítulo são explicados os códigos de programas desenvolvidos para a interface de usuário (FRDM-KL25Z utilizada em conjunto com a placa confeccionada). Os programas englobam funcionalidades como o *debounce* de botões, LEDs que piscam através de PWM, interrupção externa, comunicação serial via UART, mensagens mostradas em um LCD e conversões via ADC e DAC.

No capítulo 6 são apresentados os resultados experimentais obtidos e discussões a respeito.

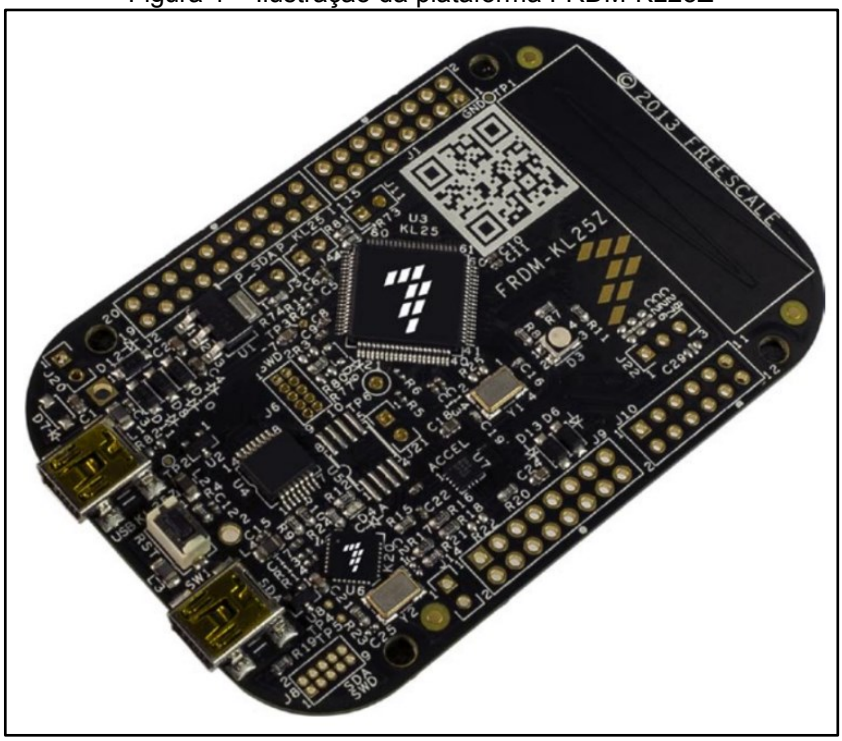
Por fim, na conclusão (capítulo 7), é avaliada a execução do trabalho e discussões sobre a produção do projeto, além de possíveis aperfeiçoamentos.

[2]Arduino Shields são PCIs no estilo *plug and play* compatíveis com as placas distribuídas pela empresa Arduino e usadas para expandir o seu *hardware*.

2 FRDM-KL25Z: ESTRUTURA DE HARDWARE

Neste capítulo é destacada toda a estrutura de *hardware* pertencente à FRDM-KL25Z, com o objetivo de apresentar as suas possibilidades de aplicação. Também são realizadas comparações entre o microcontrolador da plataforma e outros MCUs disponíveis no mercado (incluindo o AT89S52), para assim evidenciar as suas vantagens e desvantagens. A Figura 1 ilustra a FRDM-KL25Z:

Figura 1 – Ilustração da plataforma FRDM-KL25Z



Fonte: NXP. FRDM-KL25Z User's Manual (REV 2.0, 2013, p. 1)

2.1 VISÃO GERAL

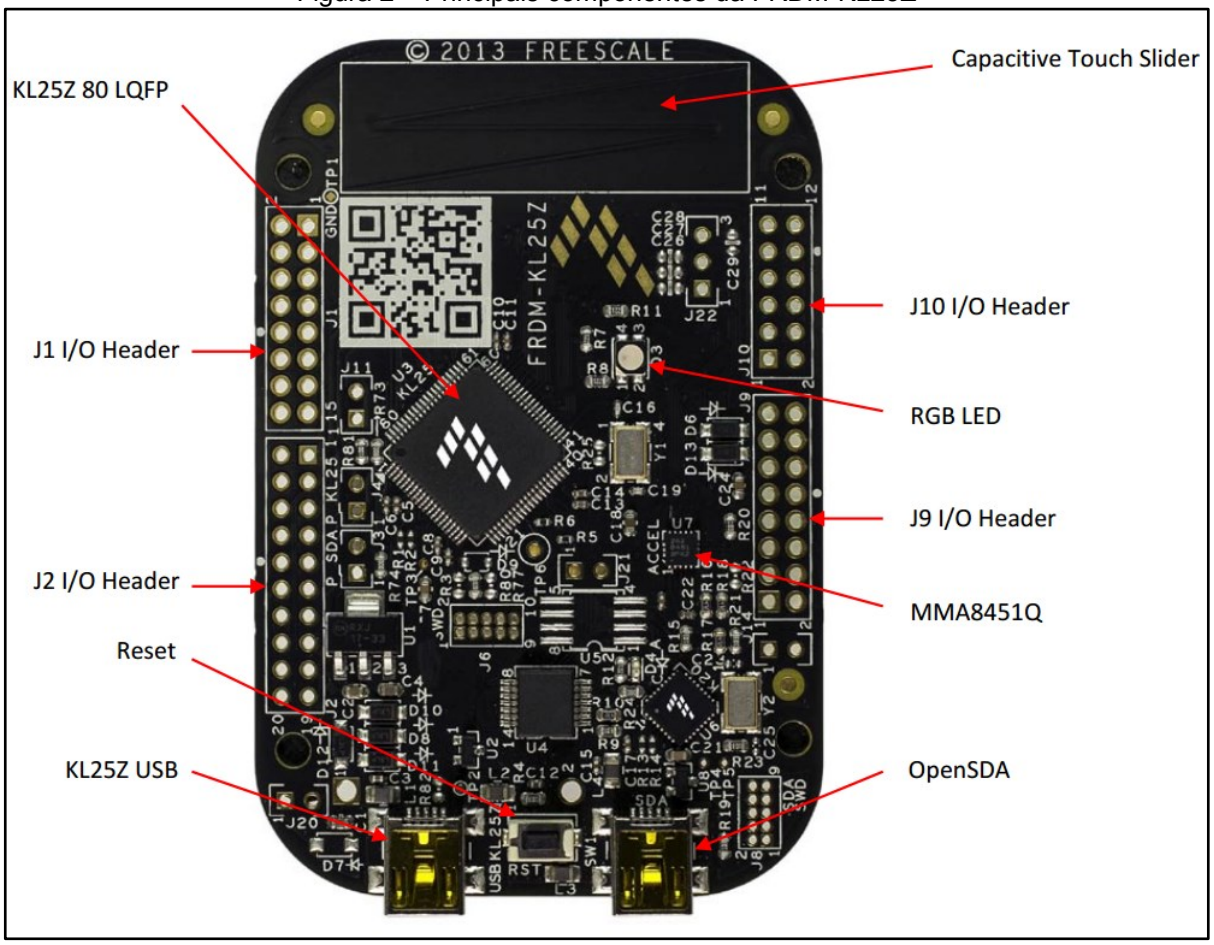
A FRDM-KL25Z inclui os seguintes componentes:

a) o microcontrolador MKL25Z128VLK4 (encapsulamento LQFP de 80 pinos);
b) alimentação flexível: USB, bateria ou fonte externa;
c) interface de toque capacitivo;
d) um acelerômetro (PN do fabricante: MMA8451Q);
e) um LED RGB (PN do fabricante: CLV1A-FKB-CJ1M1F1BB7R4S3);
f) acesso ao MCU através de cabeçalhos de pinos I/O;

g) depuração e programação através da interface OpenSDA, via USB.

A Figura 2 evidencia os componentes citados e a sua localização na PCI:

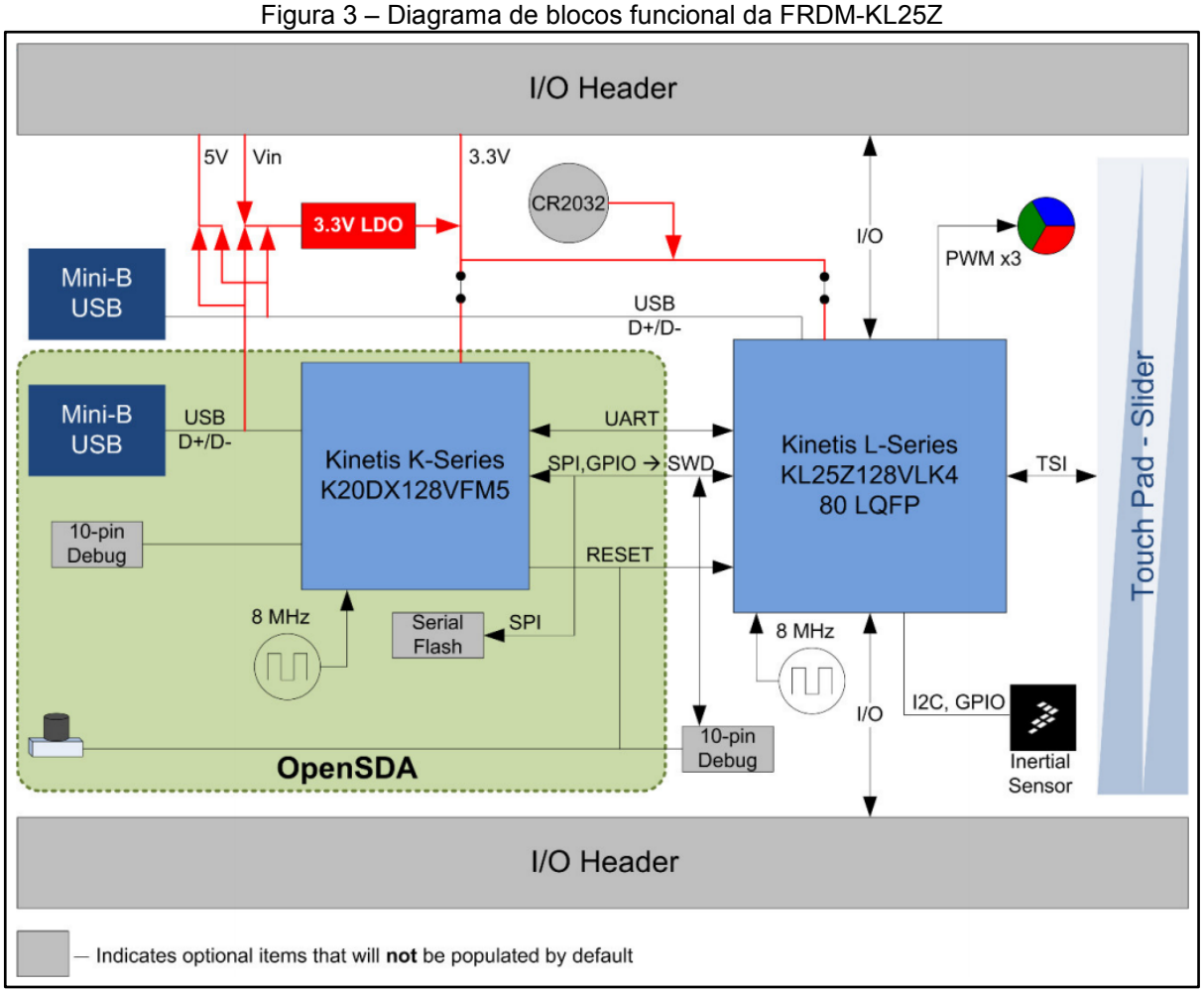
Figura 2 – Principais componentes da FRDM-KL25Z



Fonte: NXP. FRDM-KL25Z User's Manual (REV 2.0, 2013, p. 5)

Nas próximas seções cada um destes componentes é descrito individualmente. A Figura 3 mostra o diagrama de blocos funcional da FRDM-KL25Z:

Figura 3 – Diagrama de blocos funcional da FRDM-KL25Z



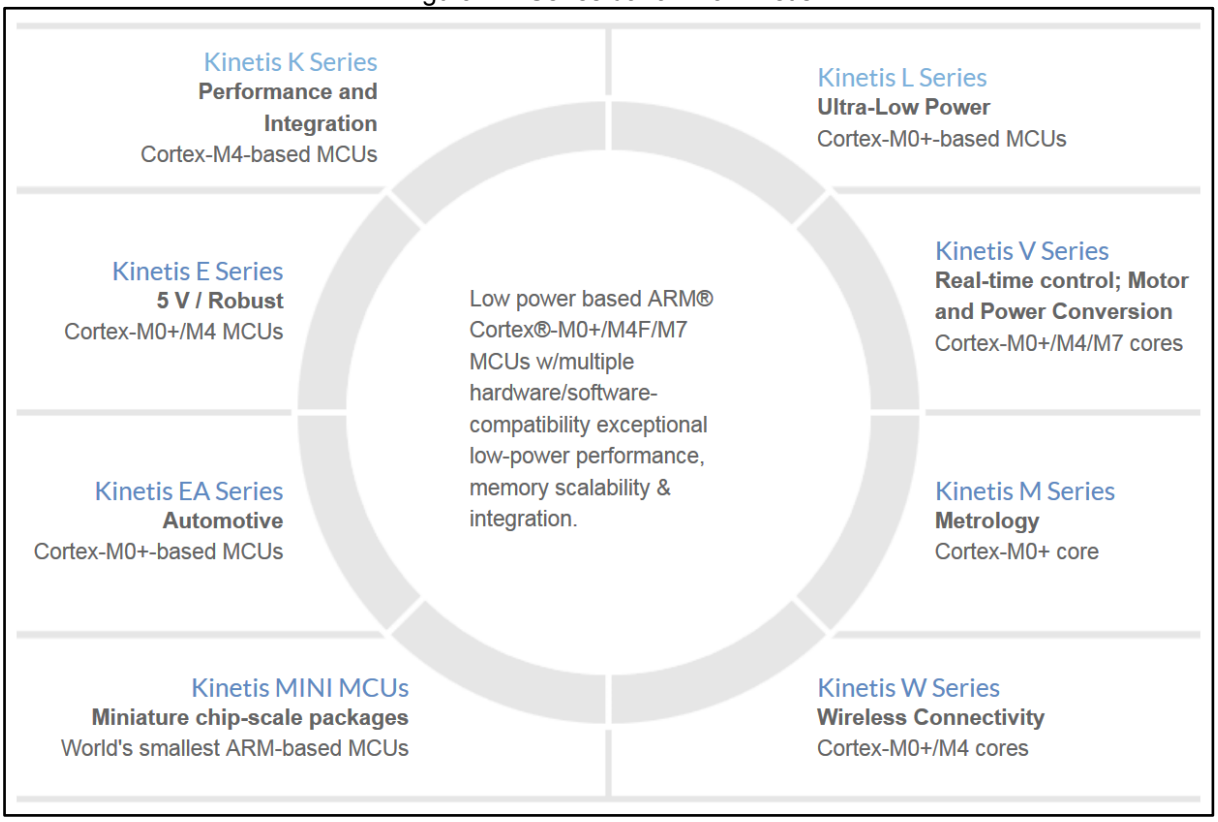
Fonte: NXP. FRDM-KL25Z User's Manual (REV 2.0, 2013, p. 4)

2.2 O MICROCONTROLADOR MKL25Z128VLK4

2.2.1 A família Kinetis e a série L

O microcontrolador alvo da FRDM-KL25Z é o MKL25Z128VLK4. Este MCU é pertencente à família Kinetis da NXP, que abrange microcontroladores de núcleos ARM Cortex-M0+, ARM Cortex-M4 e ARM Cortex-M7. Dentro desta família há uma subdivisão entre séries, na qual o MKL25Z128VLK4 insere-se na série Kinetis L. A Figura 4 identifica todas as séries disponibilizadas pela NXP:

Figura 4 – Séries da família Kinetis

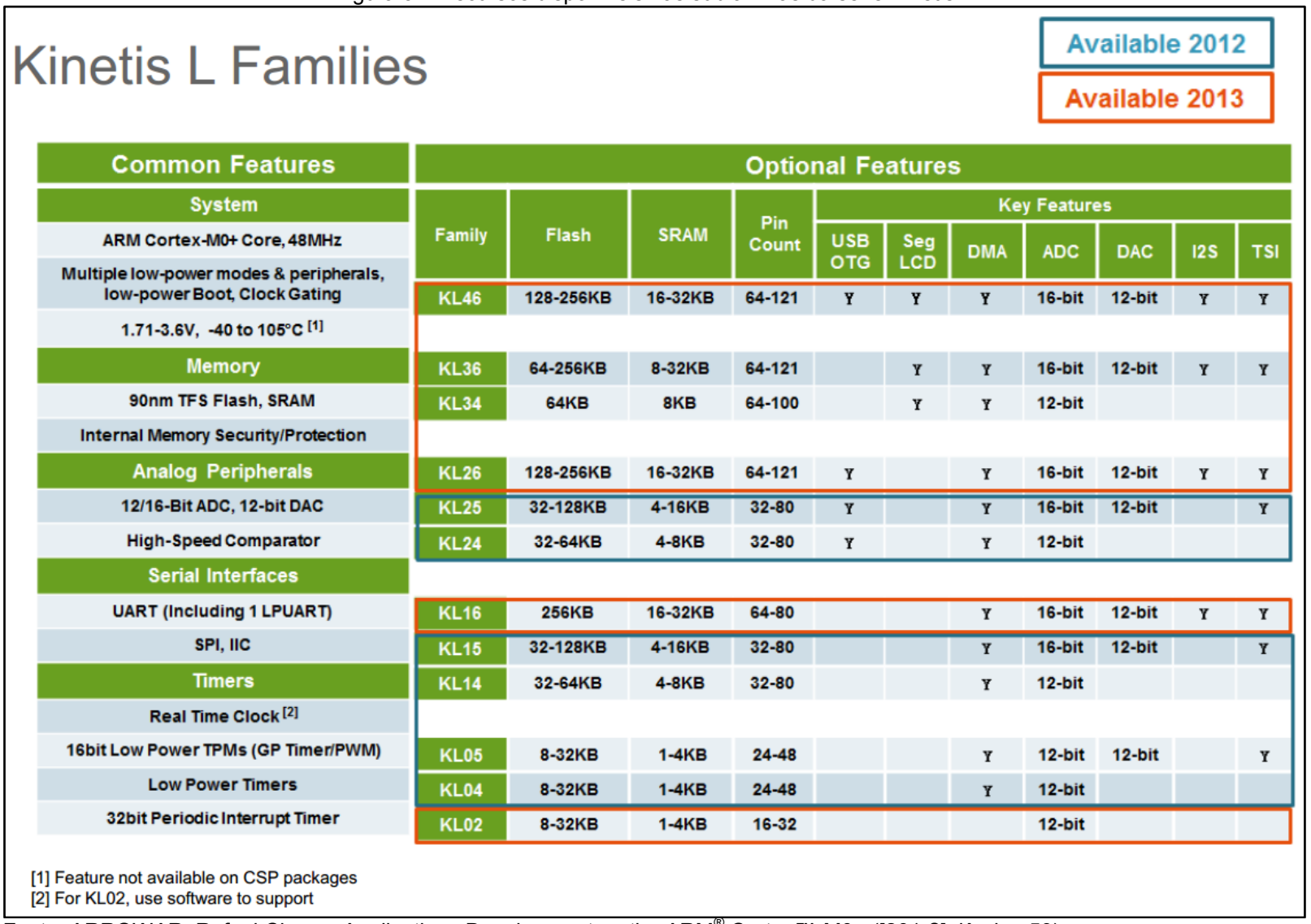


Fonte: NXP. Kinetis Cortex-M MCUs. Disponível em: <http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/kinetis-cortex-m-mcus:KINETIS>. Acesso em: 31 ago. 2016

A série Kinetis L contém apenas microcontroladores de núcleo ARM Cortex-M0+ e tem como principal diferenciação em relação às outras séries a eficiência energética.

Internamente à série Kinetis L, o MKL25Z128VLK4 insere-se na subfamília KL25. A Figura 5 explicita os recursos disponíveis em cada subfamília da série Kinetis L:

Figura 5 – Recursos disponíveis nas subfamílias da série Kinetis L



Fonte: ARROWAR. Rafael Charro. Applications Development on the ARM® Cortex™-M0+ ([201-?], lâmina 59)

2.2.2 A ARM Holdings

Como citado na seção anterior, o MKL25Z128VLK4 é um microcontrolador com núcleo ARM Cortex-M0+, o qual é licenciado pela empresa ARM Holdings. Nesta seção será explorada a história desta empresa, o seu papel e a sua importância na indústria.

A ARM Holdings é uma empresa multinacional japonesa[1] sediada em Cambridge (Inglaterra), que desenvolve primariamente projetos para semicondutores (principalmente processadores). Diferentemente de empresas fornecedoras de microprocessadores tradicionais (como a Intel) a ARM apenas cria e licencia a sua tecnologia como propriedade intelectual, ao invés de manufaturar e vender os dispositivos físicos em si. Além disso, a empresa produz ferramentas, plataformas e sistemas em *software* e *hardware*.

O acrônimo ARM foi usado pela primeira vez em 1983 e originalmente significava "Acorn RISC[2] *Machine*", uma vez que a empresa desenvolvedora desta tecnologia se chamava Acorn Computers. Em 1990 o acrônimo foi alterado para "*Advanced* RISC *Machine*", pois o interesse pela integração destes processadores em outras plataformas resultou em um *joint venture* entre a Acorn Computers, a Apple Computer (hoje Apple Inc.) e a VLSI Technology. A empresa resultante foi nomeada Advanced RISC Machines Ltd.. Em 1998, com a oferta pública inicial, a empresa trocou o seu nome para ARM Holdings e é usualmente chamada apenas por ARM, assim como seus processadores.

Os processadores baseados em projetos licenciados pela ARM permeiam todas as classes de dispositivos computacionais, desde microcontroladores em sistemas embarcados (incluindo sistemas de segurança para carros, *smart* TVs e relógios inteligentes), até *smartphones*, *tablets*, *laptops*, *desktops* e servidores. Uma das principais características dos processadores ARM é seu baixo consumo de energia elétrica, o que os torna particularmente convenientes para uso em dispositivos portáteis. Por consequência, a família de microprocessadores de 32 bits da ARM é a mais utilizada no mundo (ARM. Product Backgrounder, 2005).

---

[1]No dia 5 de Setembro de 2016 a empresa japonesa SoftBank Group completou a aquisição da ARM por aproximadamente $23,4 bilhões. O "coração" da empresa nasceu originalmente em Cambridge.
[2]Na seção 2.2.3 são explicadas as principais características da arquitetura RISC.

2.2.3 Processadores ARM: arquiteturas, núcleos e conjuntos de instruções

Os processadores ARM estão subdivididos por diversas famílias de arquiteturas RISC projetadas pela ARM. RISC é um tipo de arquitetura de processadores que utiliza um conjunto altamente otimizado e pequeno de instruções, o qual contrapõe com as arquiteturas que utilizam um conjunto mais especializado de instruções, como, por exemplo, CISC. A Tabela 1 evidencia as principais diferenças entre as arquiteturas RISC e CISC:

Tabela 1 – Diferenças entre as arquiteturas RISC e CISC

RISC	CISC
Ênfase no *software*	Ênfase no *hardware*
Instruções de apenas um *clock*	Inclui instruções de múltiplos *clocks*
Registrador para registrador: “LOAD” e “STORE” são instruções independentes	Memória para memória: “LOAD” e “STORE” incorporadas nas instruções
Baixos ciclos por segundo, códigos de tamanho grande	Altos ciclos por segundo, códigos de tamanho pequeno

Fonte: Stanford. RISC vs. CISC. Disponível em: <http://cs.stanford.edu/people/eroberts/courses/soco/projects/2000-01/risc/risccisc/index.html>. Acesso em: 01 set. 2016

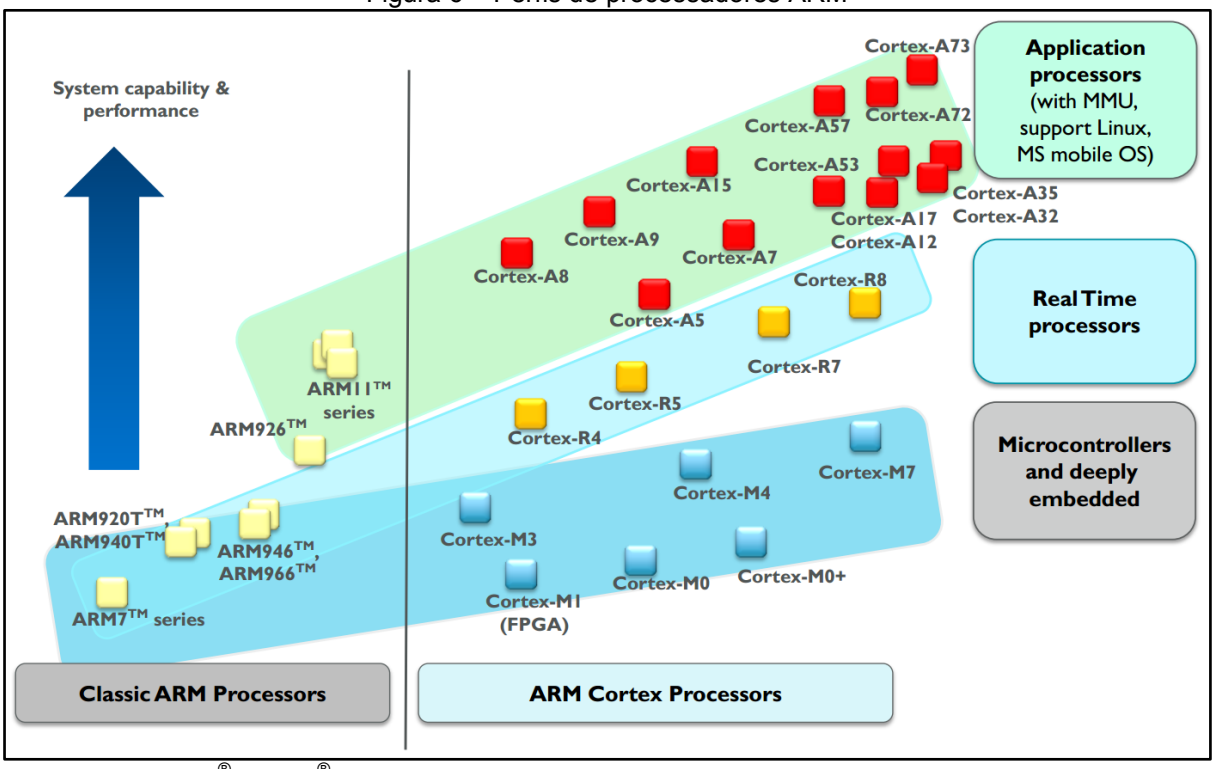
Ao longo de sua história, a ARM desenvolveu muitos processadores, os quais podem ser divididos entre processadores ARM clássicos e os novos processadores ARM Cortex. Além disso, os processadores podem ser divididos por tipos de perfil:

a) processadores para aplicação geral – processadores de mais alto desempenho, que possuem frequência de operação acima de 1GHz e suportam MMU (requerido para embarcar sistemas operacionais como Linux, Android e Windows);
b) processadores de tempo real – processadores para uso em aplicações de tempo real (como controladores de discos rígidos), com frequência de operação entre 200MHz e 800MHz e suporte de MPU, o qual define as permissões no acesso da memória;
c) processadores para microcontroladores – populares no uso em sistemas embarcados, são processadores projetados para alta eficiência energética e área em silício reduzida, possuindo pipelines mais curtos e menores frequências de operação (normalmente abaixo de 200MHz), além de

suporte de MPU.

A Figura 6 relaciona a divisão entre processadores ARM clássicos e ARM Cortex e suas distribuições entre tipos de perfil e desempenho:

Figura 6 – Perfis de processadores ARM



Fonte: ARM. ARM® Cortex®-M for Beginners (final v3, 2016, p. 2)

Os processadores ARM são intitulados pelos nomes de seus núcleos. O núcleo de um processador é a unidade de processamento que lê instruções para executar ações. O Quadro 1 associa os núcleos projetados para cada arquitetura ARM:

Quadro 1 – Arquiteturas ARM e (famílias de) núcleos correspondentes

Arquitetura	Tamanho do núcleo em bits	(Famílias de) Núcleos projetados pela ARM Holdings	Perfil
ARMv1	32	ARM1	-
ARMv2	32	ARM2, ARM250, ARM3	-
ARMv3	32	ARM6, ARM7	-
ARMv4	32	ARM8	-
ARMv4T	32	ARM7TDMI, ARM9TDMI, SecurCore SC100	-
ARMv5TE	32	ARM7EJ, ARM9E, ARM10E	-
ARMv6	32	ARM11	-
ARMv6-M	32	ARM Cortex-M0, ARM Cortex-M0+, ARM Cortex-M1, SecurCore SC000	Microcontrolador
ARMv7-M	32	ARM Cortex-M3, SecurCore SC300	Microcontrolador
ARMv7E-M	32	ARM Cortex-M4, ARM Cortex-M7	Microcontrolador
ARMv7-R	32	ARM Cortex-R4, ARM Cortex-R5, ARM Cortex-R7, ARM Cortex-R8	Tempo real
ARMv7-A	32	ARM Cortex-A5, ARM Cortex-A7, ARM Cortex-A8, ARM Cortex-A9, ARM Cortex-A12, ARM Cortex-A15, ARM Cortex-A17	Aplicação geral
ARMv8-A	32	ARM Cortex-A32	Aplicação geral
ARMv8-A	64	ARM Cortex-A35, ARM Cortex-A53, ARM Cortex-A57, ARM Cortex-A72, ARM Cortex-A73	Aplicação geral
ARMv8-R	32	ARM Cortex-R52	Tempo real

Fonte: WIKIPEDIA. ARM architecture. Disponível em: <https://en.wikipedia.org/wiki/ARM_architecture>. Acesso em: 20 set. 2016

As arquiteturas ARM incluem em seus conjuntos de instruções os seguintes atributos RISC:

a) arquitetura *load*/*store* – somente duas instruções de acesso a memória, não havendo instruções para operações lógicas ou aritméticas com dados colocados na memória, além de resultados das operações sempre serem armazenados em algum registrador;
b) sem suporte para acessos de memória desalinhados (ARMv6 e algumas versões posteriores suportam com algumas limitações);
c) arquivo de registrador uniforme de 16 x 32-bit;
d) tamanho fixo de instrução de 32 bits para facilitar decodificação e *pipelining*, com o custo de perda em densidade de código;
e) maior parte das execuções em apenas um ciclo de *clock*.

2.2.4 Thumb e Thumb-2

Para o aumento da densidade na compilação dos códigos, processadores a partir da arquitetura ARM7TDMI (lançado em 1994) foram equipados com Thumb, uma compacta codificação de 16 bits para um subconjunto do conjunto de instruções ARM. A maioria das instruções Thumb está diretamente mapeada nas instruções ARM comuns. Em 2003, com o lançamento do núcleo ARM1156, foi introduzida a tecnologia Thumb-2, que estende o conjunto de instruções de 16 bits Thumb com instruções adicionais de 32 bits, assim produzindo um conjunto de instruções de tamanho variável. O objetivo do Thumb-2 é de se alcançar uma densidade de código similar ao do Thumb, porém com um desempenho próximo ao conjunto de instruções ARM convencional.

Para demonstrar como as instruções em Thumb e Thumb-2 são programadas, sugere-se a análise de um código exemplo em linguagem C, que em seguida será traduzido para as outras duas linguagens. A Figura 7 ilustra o código exemplo em C:

Figura 7 – Exemplo de código em linguagem C

```
typedef unsigned char uint8;
typedef unsigned short uint16;
/* r0 = x , r1 = a, r2 = b, r3 = c */
uint8 foo(uint8 x, uint8 a, uint16 b, uint16 c)
{
    if (a==2)
    {
        x += (b >> 8);
    }
    else
    {
        x += (c >> 8);
    }
    return x;
}
```

Fonte: INFRASTUCTURE DEVELOPER'S BLOG. ARM/Thumb/Thumb-2. Disponível em: <https://kmittal82.wordpress.com/2012/02/17/armthumbthumb-2/>. Acesso em: 24 nov. 2016

De acordo com o padrão de chamada de procedimento ARM, os argumentos para a função seriam passados para os registradores r0 a r3, com o valor de retorno passado para r0.

No estado Thumb, poucas instruções são condicionais. Além disso, não há

acesso em linha ao *barrel shifter*, portanto instruções separadas são necessárias para deslocar bits. Isso significa na prática que o código Thumb seria geralmente mais lento para ser executado do que o código ARM (já que mais instruções Thumb podem ser necessárias para fazer o trabalho do que o número de instruções ARM), mas pode ajudar a economizar em tamanho de código. O exemplo do código em Thumb está ilustrado na Figura 8:

Figura 8 – Exemplo de código em Thumb

```
1  _Z3foohhtt
2      0x00000000:    2902        .)      CMP     r1,#2
3      0x00000002:    d101        ..      BNE     {pc}+0x6 ; 0x8
4      0x00000004:    0a11        ..      LSRS    r1,r2,#8
5      0x00000006:    e000        ..      B       {pc}+0x4 ; 0xa
6      0x00000008:    0a19        ..      LSRS    r1,r3,#8
7      0x0000000a:    1808        ..      ADDS    r0,r1,r0
8      0x0000000c:    0600        ..      LSLS    r0,r0,#24
9      0x0000000e:    0e00        ..      LSRS    r0,r0,#24
10     0x00000010:    4770        pG      BX      lr
```

Fonte: INFRASTUCTURE DEVELOPER'S BLOG. ARM/Thumb/Thumb-2. Disponível em: <https://kmittal82.wordpress.com/2012/02/17/armthumbthumb-2/>. Acesso em: 24 nov. 2016

Imediatamente, observa-se que todas as instruções são de 16 bits em tamanho, conforme indicado pelas codificações de instrução (segunda coluna da esquerda). Isso representa um tamanho de código total de 18 bytes (dois bytes em cada instrução).

Quando se começa a analisar o código gerado, a desvantagem do estado Thumb é imediatamente aparente. Thumb só tem acesso a ramos condicionais, de modo que o código gerado é feito através de ramos. A segunda instrução é ramificada para o endereço 0x8 caso não haja igualdade na comparação do primeiro passo (isto é, entra-se na parte “else” do código em C). No endereço 0x8, nota-se a segunda desvantagem do estado Thumb: a falta de um *barrel shifter* em linha. Uma instrução LSRS separada desloca os bits em r3 por 8 e os armazena em r1. Em seguida, esse valor em r1 é adicionado a r0. O valor de retorno está quase pronto, mas antes se precisam zerar os 24 bits superiores. O conjunto de instruções Thumb não tem acesso a “AND”, então ele executa dois deslocamentos em r0, primeiro deslocando r0 por 24 bits para a esquerda (zerando os 8 bits inferiores) e então deslocando mais 24 bits para a direita (zerando os 24 bits superiores). A parte do “if” do código em C é semelhante à parte que fora detalhada.

O exemplo do mesmo código em Thumb-2 está ilustrado na Figura 9:

Figura 9 – Exemplo de código em Thumb-2

```
1  _Z3foohhtt
2      0x00000000:    2902        .)      CMP       r1,#2
3      0x00000002:    bf14        ..      ITE       NE
4      0x00000004:    eb002013    ...     ADDNE     r0,r0,r3,LSR #8
5      0x00000008:    eb002012    ...     ADDEQ     r0,r0,r2,LSR #8
6      0x0000000c:    b2c0        ..      UXTB      r0,r0
7      0x0000000e:    4770        pG      BX        lr
```

Fonte: INFRASTUCTURE DEVELOPER'S BLOG. ARM/Thumb/Thumb-2. Disponível em: <https://kmittal82.wordpress.com/2012/02/17/armthumbthumb-2/>. Acesso em: 24 nov. 2016

Nota-se é que existe uma mistura de 32 bits e 16 bits nas instruções através de suas codificações (segunda coluna da esquerda). Ao invés de todas as instruções serem da mesma largura, têm-se quatro instruções que são de 16 bits e duas instruções que são 32 bits, representando um tamanho de código de 16 bytes.

Como se pode ver, a segunda instrução é uma construção ITE NE. Esta não é uma instrução propriamente dita, mas um aviso ao processador, instruindo-o que algumas instruções condicionais precisam ser executadas e a primeira será baseada na condição NE (não igual). A terceira instrução informa ao processador: "se o resultado da comparação anterior era NE, então desloque r3 por 8 bits para a direita, adicione-o a r0 e armazene o resultado de volta em r0". Isso corresponde à parte "else" do laço em C. Não só se conseguiu evitar um ramo, como também se foi capaz de mudar os bits e fazer a adição, tudo em uma simples instrução. A terceira instrução é exatamente a mesma, mas trata da parte "if" do laço. Finalmente, a instrução UXTB é uma instrução que estende o byte a uma *word*, que neste caso zera os 24 bits superiores.

2.2.5 Os processadores ARM Cortex-M e o núcleo ARM Cortex-M0+

O ARM Cortex-M0+ é um núcleo de processador de arquitetura ARM ARMv6-M, de 32 bits, pertencente à família de processadores ARM Cortex-M.

A linha de processadores ARM da família Cortex-M está voltada para o baixo custo e eficiência energética. No entanto, se comparados a processadores típicos integrados à maioria dos microcontroladores, este processadores ainda são bastante poderosos. A Tabela 2 descreve as principais características de cada núcleo da

família Cortex-M:

Tabela 2 – Principais características dos núcleos da família ARM Cortex-M

Núcleo	Descrição
Cortex-M0	Um processador pequeno (a partir de 12 mil *gates*) com baixo custo, baixo consumo de energia e para aplicações embarcadas
Cortex-M0+	O processador mais eficiente para pequenos sistemas embarcados. Tamanho e modelo de programação similar ao do Cortex-M0, porém com opções adicionais
Cortex-M1	Pequeno processador otimizado para projetos de FPGA. Mesmo conjunto de instruções que o Cortex-M0
Cortex-M3	Pequeno e potente processador embarcado para microcontroladores de baixo consumo, com um rico conjunto de instruções
Cortex-M4	Provê todas as características do Cortex-M3, com instruções adicionais para tarefas de DSP. Além disso, possui uma unidade opcional para ponto flutuante de precisão compatível com o padrão IEEE 754
Cortex-M7	Processador de alto desempenho para microcontroladores de alta qualidade e aplicações que requerem processamento intenso

Fonte: ARM. ARM® Cortex®-M for Beginners (final v3, 2016, p. 3)

Algumas das funções padrões da família Cortex-M seguem listadas abaixo:

a) apenas suporte para instruções ARM Thumb e Thumb-2;
b) manuseio das interrupções é gerenciado por um controlador interno denominado *Nested Vector Interrupt Controller* (NVIC), que provê a priorização automática, mascaramento e *nesting* de interrupções além de captura de exceções do sistema;
c) tratamento das interrupções podem ser escritas como funções normais na linguagem C;
d) modo de “descanso” embutido na arquitetura;
e) suporte para embarcar sistemas operacionais voltados para microcontroladores;
f) suporte para depuração de código.

O controlador NVIC suporta os seguintes meios de interrupção:

a) um número (dependendo do núcleo) de interrupções através de periféricos;
b) uma requisição de interrupção por hardware não-mascarável pelo sistema,

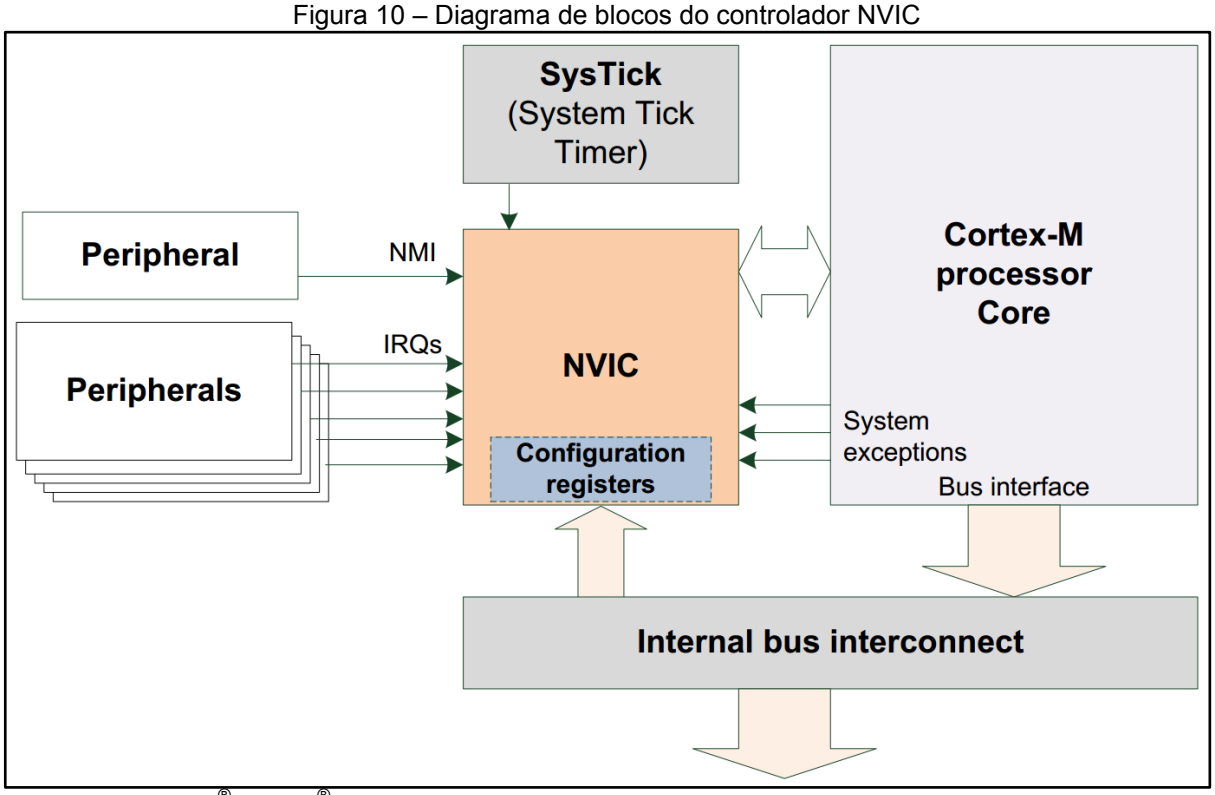
denominada de *Non-Maskable Interrupt* (NMI);

c) uma requisição de interrupção por um *timer* embutido ao sistema, denominado de SysTick;

d) um número (dependendo do núcleo) de exceções do sistema, como por exemplo, falhas no gerenciamento da memória do sistema.

A Figura 10 apresenta o diagrama de blocos do controlador NVIC em um processador da família Cortex-M:

Figura 10 – Diagrama de blocos do controlador NVIC



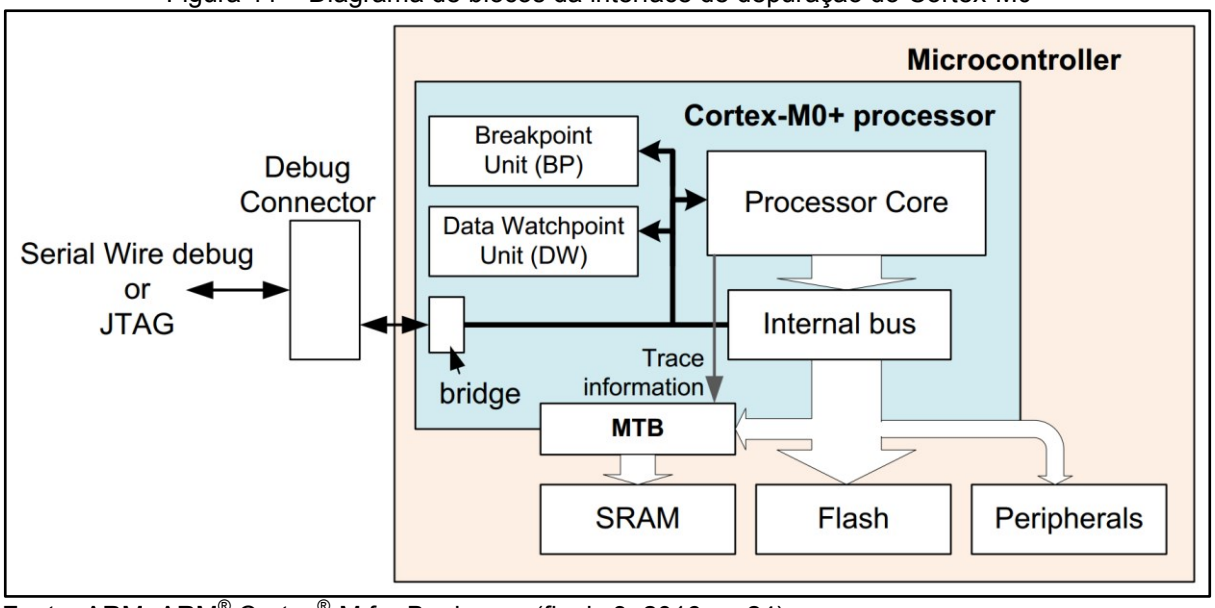
Fonte: ARM. ARM® Cortex®-M for Beginners (final v3, 2016, p. 5)

A programação e depuração dos códigos executados nos processadores Cortex-M pode ser realizada via a tradicional conexão JTAG, que requer de 4 a 5 pinos (TDI, TDO, TCK, TMS e opcionalmente nTRST), ou o novo protocolo SWD que necessita de apenas dois pinos (SWDIO e SWCLK). Em ambos os casos é necessário que seja conectado um adaptador para depuração ao microcontrolador através da interface de conexão JTAG ou SWD.

Os processadores Cortex-M0 e Cortex-M0+ não possuem a ferramenta *Trace Interface*, disponível nos outros processadores Cortex-M, para coleta de informação

em tempo real durante a execução do programa. No entanto, pode-se adicionar uma opção ao silício do Cortex-M0+, chamada de *Micro Trace Buffer* (MTB), que possibilita o usuário a alocar uma pequena parte da SRAM do sistema como um *buffer* para armazenar instruções. Assim sendo, o histórico da execução de instruções recentes pode ser capturado pelo adaptador para depuração. A Figura 11 mostra o diagrama de blocos da interface de depuração do Cortex-M0+:

Figura 11 – Diagrama de blocos da interface de depuração do Cortex-M0+



Fonte: ARM. ARM® Cortex®-M for Beginners (final v3, 2016, p. 24)

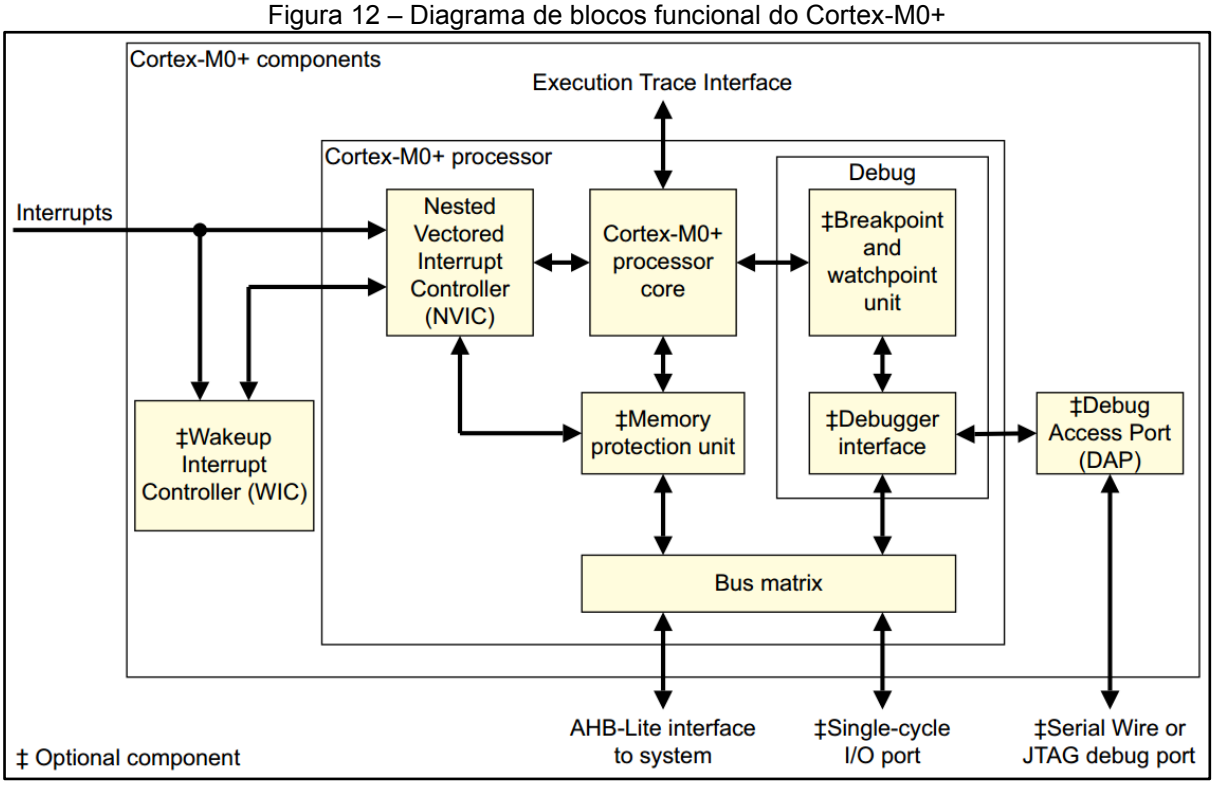
O Cortex-M0+ também recebeu recursos do Cortex-M3 e Cortex-M4, que podem ser adicionados opcionalmente ao seu silício, tais como o MPU.

As principais características do Cortex-M0+ são:

a) arquitetura ARM ARMv6-M (32 bits);
b) dois estágios de *pipeline*;
c) conjunto de instruções igual ao do Cortex-M0, com Thumb e Thumb-2;
d) até 32 interrupções através de periféricos, além da NMI.

A Figura 12 ilustra o diagrama de blocos funcional do Cortex-M0+:

Figura 12 – Diagrama de blocos funcional do Cortex-M0+



Fonte: ARM. Cortex-M0+ Technical Reference Manual (Revision: r0p1, 2012, p. 20)

2.2.6 A subfamília KL25

Dentro da série Kinetis L, o MKL25Z128VLK4 está inserido na subfamília KL25, a qual abrange doze MCUs que apenas diferem nos tamanhos de memória flash e SRAM e na quantidade de pinos I/O. O MKL25Z128VLK4 especificamente possui 128kB (quilobyte) de memória flash, 16kB de SRAM e 66 pinos I/O.

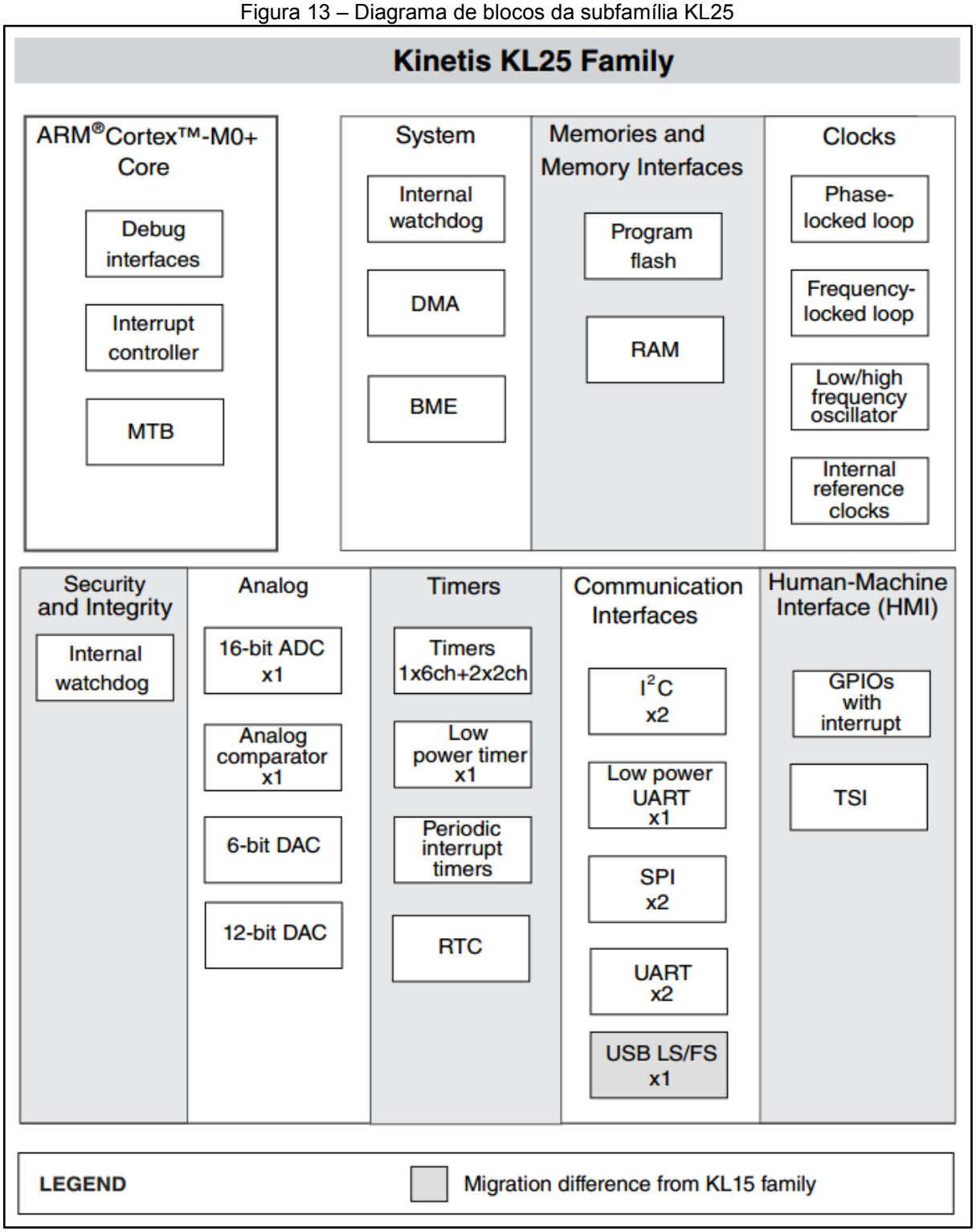
As principais características da subfamília KL25 são citadas a seguir:

a) mínimo consumo de 47μA/MHz no modo muito baixo consumo de energia;
b) frequência de operação até 48MHz;
c) oito modos de operação de consumo de energia;
d) controlador DMA com 4 canais, suportando até 63 fontes de requisição;
e) interface de depuração SWD e *Micro Trace Buffer*;
f) tensão de operação entre 1,71V e 3,6V;
g) 66 pinos I/O e interface TSI disponível;
h) controlador USB On-the-Go com *tranceiver* e regulador de 5V para 3,3V;
i) dois módulos SPI;

j) três módulos UART (um podendo operar em modo de baixa energia);
k) dois módulos I^2C;
l) um ADC SAR de 16 bits (possível multiplexar 14 pinos);
m) um DAC de 12 bits;
n) um módulo de *timer*/PWM (TPM) com 6 canais e dois módulos com 2 canais;
o) *timers* de interrupção periódica;
p) relógio de tempo real.

A Figura 13 mostra o diagrama de blocos da subfamília KL25:

Figura 13 – Diagrama de blocos da subfamília KL25



Fonte: NXP. Kinetis KL25 Sub-Family (REV 5, 2014, p. 3)

A Tabela 3 indica as características de operação de tensão e corrente da subfamília KL25:

Tabela 3 – Características de operação de tensão e corrente da subfamília KL25

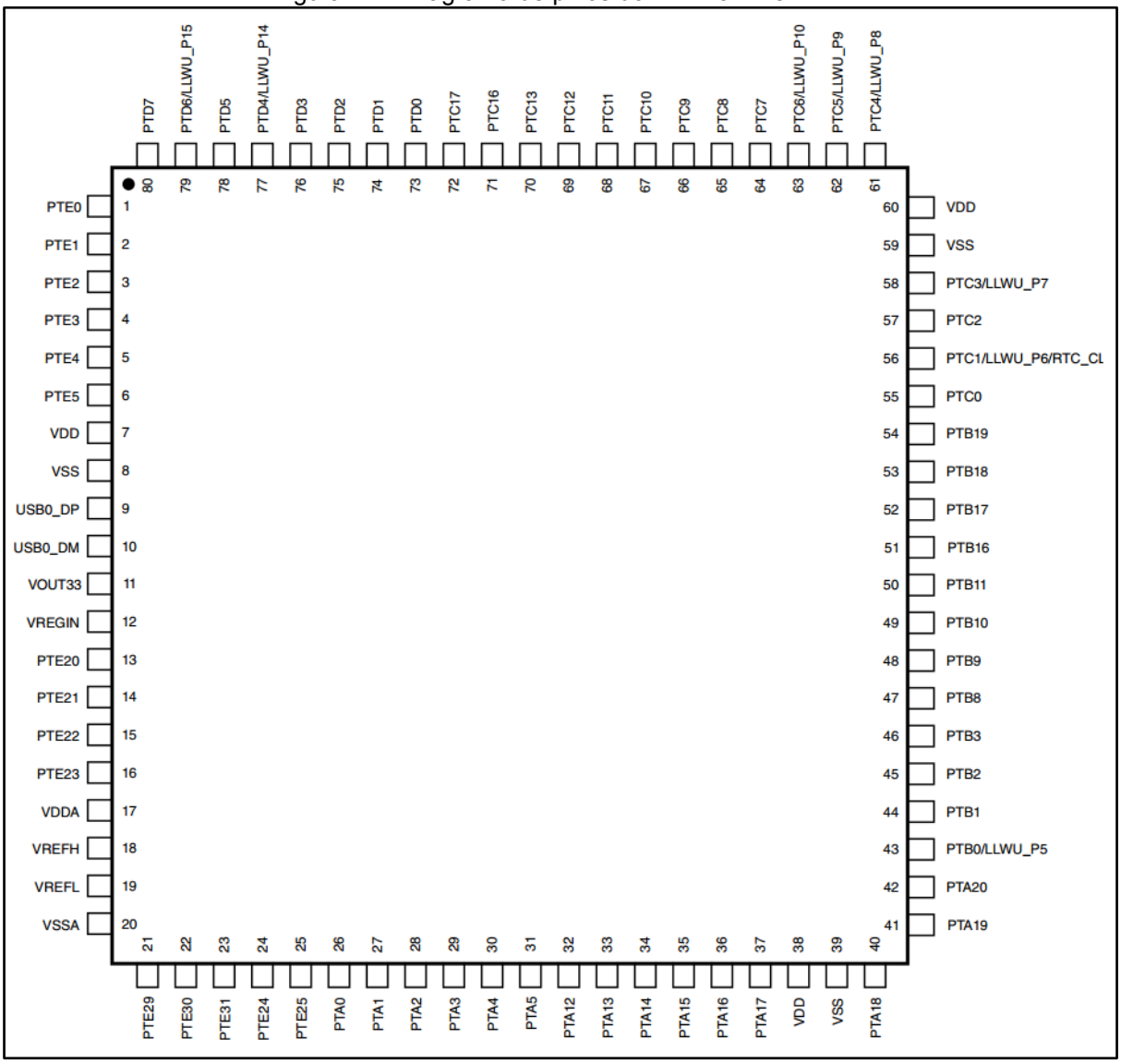
Descrição	Mínimo	Máximo	Unidade
V_{DD}: Tensão de alimentação (limites máx.)	-0,3	3,8	V
V_{DD} (limites de funcionamento)	1,71	3,6	V
I_{DD}: Corrente de alimentação (limites máx.)	–	120	mA
Tensão de entrada em um pino I/O	-0,3	V_{DD}+0,3	V
Corrente máxima instantânea em um único pino	-25	25	mA
V_{DDA}: Tensão de alimentação do ADC (limites máx.)	V_{DD}-0,3	V_{DD}+0,3	V
V_{DDA} (limites de funcionamento)	1,71	3,6	V

Fonte: NXP. Kinetis KL25 Sub-Family (REV 5, 2014)

2.2.7 Especificações de hardware e conexões

O diagrama de pinos do MKL25Z128VLK4 está representado pela Figura 14:

Figura 14 – Diagrama de pinos do MKL25Z128VLK4



Fonte: NXP. Kinetis KL25 Sub-Family (REV 5, 2014, p. 48)

Dos 80 pinos disponíveis no encapsulamento LQFP que envolve o microcontrolador, 66 são classificados como I/O e estão disponíveis para programação pelo usuário. Os pinos I/O tem nomeação iniciada pelas letras “PT”, seguidas pela letra da respectiva porta (A, B, C, D ou E) e finalizando com o número do pino da respectiva porta. Além disso, para cada pino I/O pode ser programada alguma função específica, dentre as quais as mais comuns são funcionalidades relativas aos módulos de comunicação disponíveis (UART, SPI, I^2C e TSI), interrupção externa, entradas para o ADC e saídas dos *timers*/PWM. A lista completa das possíveis funções de cada pino I/O pode ser encontrada na tabela da página 44 do *datasheet* da subfamília Kinetis KL25 (REV 5, 2014).

O *clock* do MCU é gerado a partir de um cristal oscilador externo. O MKL25Z128VLK4 possui um circuito oscilador interno que é compatível com três faixas de frequências relativas ao cristal conectado: 32 a 40kHz (modo de baixa frequência), 3 a 8MHz (modo de alta frequência, faixa curta) e 8 a 32MHz (modo de alta frequência, faixa larga). O microcontrolador na plataforma FRDM-KL25Z tem seu *clock* gerado através de um cristal de 8MHz.

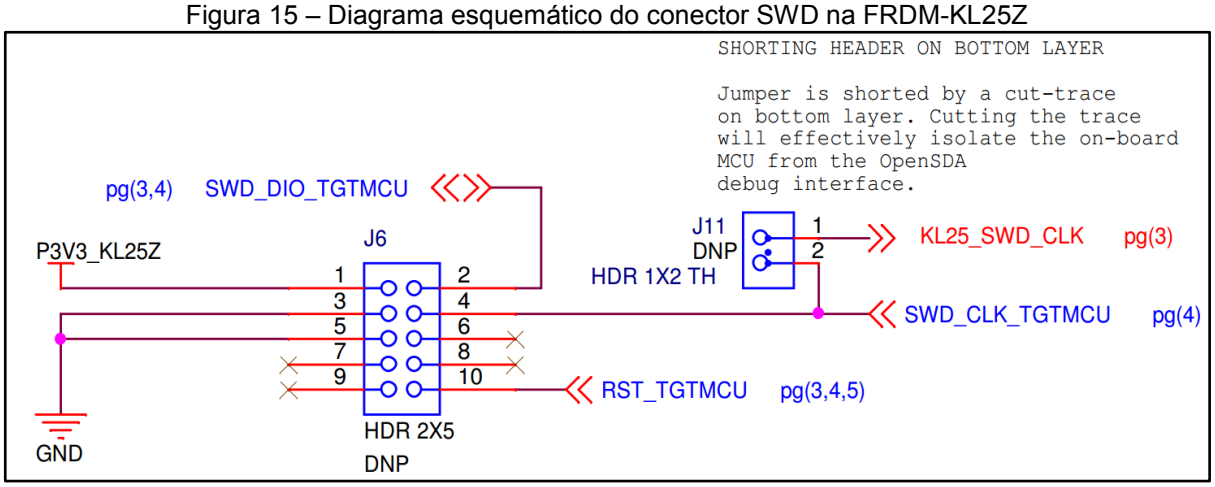
A plataforma FRDM-KL25Z oferece uma forma pronta de comunicação serial via UART, através da conexão entre os pinos PTA1 e PTA2 (UART0_RX e UART0_TX, respectivamente) do MCU e a interface OpenSDA, que por sua vez pode ser acessada pela porta USB OpenSDA. Assim, o usuário pode receber e enviar informações ao MCU através de um emulador terminal instalado em um computador. Os pinos PTA1 e PTA2 também estão conectados respectivamente aos pinos 2 e 4 do cabeçalho de pinos J1. Dessa forma, pode-se também usar estes pinos para embarcar algum outro dispositivo que utiliza o módulo UART para comunicação.

A plataforma FRDM-KL25Z permite reiniciar o microcontrolador através de três formas:

a) pressionando o botão SW1;
b) utilizando a interface OpenSDA;
c) fornecendo uma tensão de 0V em qualquer um dos pinos do cabeçalho J14.

Há duas maneiras de programar/depurar os códigos no MCU: primariamente através da interface OpenSDA, ou alternativamente através da conexão de um adaptador para depuração ao cabeçalho de pinos J6, que provê acesso aos sinais SWD do microcontrolador. A Figura 15 ilustra o diagrama esquemático do conector SWD:

Figura 15 – Diagrama esquemático do conector SWD na FRDM-KL25Z



Fonte: NXP. FRDM-KL25Z Schematics (REV E, 2013. p. 3)

2.2.8 Comparações com outros MCUs

Para serem esclarecidas as vantagens e desvantagens no uso do MKL25Z128VLK4 e a plataforma de desenvolvimento FRDM-KL25Z nas aulas de laboratório da disciplina de Microprocessadores I, foi elaborado um quadro comparativo entre estes *hardwares* e três outros bastante utilizados na indústria e por hobistas. O Quadro 2 evidencia as principais características:

Quadro 2 – Comparação entre o MKL25Z128VLK4 e outros MCUs

MCU	MKL25Z128VLK4	AT89S52-24PU	ATMEGA328P-AUR	PIC18F4520-I/P
Desenvolvedor	NXP	Atmel	Atmel	Microchip Technology
Preço unitário do MCU (Digi-Key)	$5,54	$4,42	$3,81	$4,15
PN (Digi-Key)	MKL25Z128VLK4-ND	AT89S52-24PU-ND	ATMEGA328P-AURCT-ND	PIC18F4520-I/P-ND
Plataforma comercial disponível	FRDM-KL25Z	MIKROE-455	Arduino Uno R3 (A000066)	Open18F4520
Fornecedor comercial da plataforma	Digi-Key	Digi-Key	Digi-Key	Waveshare
Preço unitário da plataforma	$14,00	$141,78	$21,49	$29,99
PN da plataforma (fornecedor)	FRDM-KL25Z-ND	1471-1471-ND	1050-1024-ND	Open18F4520
Núcleo do processador	ARM Cortex-M0+	8051	AVR	PIC
Tamanho do núcleo	32 bits	8 bits	8 bits	8 bits
Máxima frequência de *clock*	48MHz	33MHz	20MHz	40MHz
Conectividade	I²C,SPI,UART/ USART,USB	UART/ USART	I²C,SPI, UART/USART	I²C, SPI, UART/ USART
Quantidade de pinos I/O	66	32	23	36
Tamanho de memória de programa	128kB	8kB	32kB	32kB
Tipo de memória de programa	Flash	Flash	Flash	Flash
Tamanho da EEPROM	Indisponível	Indisponível	1kB	256B
Tamanho da RAM	16kB	256B	2kB	1,5kB
Tensão de alimentação	1,71V ~ 3,6V	4V ~ 5,5V	1,8V ~5,5V	4,2V ~ 5,5V
Conversores de dados	ADC 14x16b, DAC 1x12b	Indisponível	ADC 8x10b	ADC 13x10b
Encapsulam.	80-LQFP	40-DIP	32-TQFP	40-DIP

Fonte: Elaborado pelo autor

Como se pode observar, o MKL25Z128VLK4 é em média $1,41 mais caro do que outros MCUs analisados, porém a plataforma de desenvolvimento que o utiliza, a FRDM-KL25Z, é a mais barata. Além disso, as especificações do MKL25Z128VLK4 são superiores em todos os quesitos, com exceção de não possuir memória EEPROM separada da memória de programa (Flash). Uma vantagem muito importante observada é que o MKL25Z128VLK4 é o único que tem processador com núcleo de 32 bits, o que possibilita um maior armazenamento de dados nas variáveis de programação e por consequência maior precisão nas operações. Outra característica relevante do MKL25Z128VLK4 é que a frequência do seu *clock* é de 48MHz, sendo isto um fator bastante atrativo para aplicações que demandam maior rapidez na execução das instruções.

2.3 A ALIMENTAÇÃO

Para a alimentação da plataforma FRDM-KL25Z dispõe-se de uma das seguintes opções:

a) conector USB OpenSDA (identificado no diagrama esquemático da plataforma como J7);
b) conector USB KL25Z (identificado no diagrama esquemático da plataforma como J5);
c) pino V_{IN} (pino 16 do cabeçalho de pinos identificado como J9);
d) pino de 3,3V (pino 8 do cabeçalho de pinos identificado como J9);
e) bateria de célula tipo moeda (espaço identificado como BT1).

A alimentação sendo provida por um dos conectores USB ou pelo pino V_{IN} é regulada por um regulador de tensão integrado à placa (identificado como U1). A saída de tensão desse regulador é fixa em 3,3V (ideal para a alimentação do MCU) e suporta uma corrente de saída de até 800mA.

Uma observação importante é que para realizar a depuração/programação do microcontrolador via interface OpenSDA é obrigatório que a alimentação seja feita através do conector USB OpenSDA, caso contrário o MCU apenas será depurável/programável através de um adaptador para depuração conectado à interface SWD (identificado como J6). Vale ressaltar que se não há interesse em

gravar ou depurar um programa no microcontrolador, sendo a intenção apenas fazer com este execute o seu programa já previamente gravado, pode-se utilizar qualquer um dos meios de alimentação.

A Tabela 4 identifica as opções de alimentação da plataforma e suas especificações:

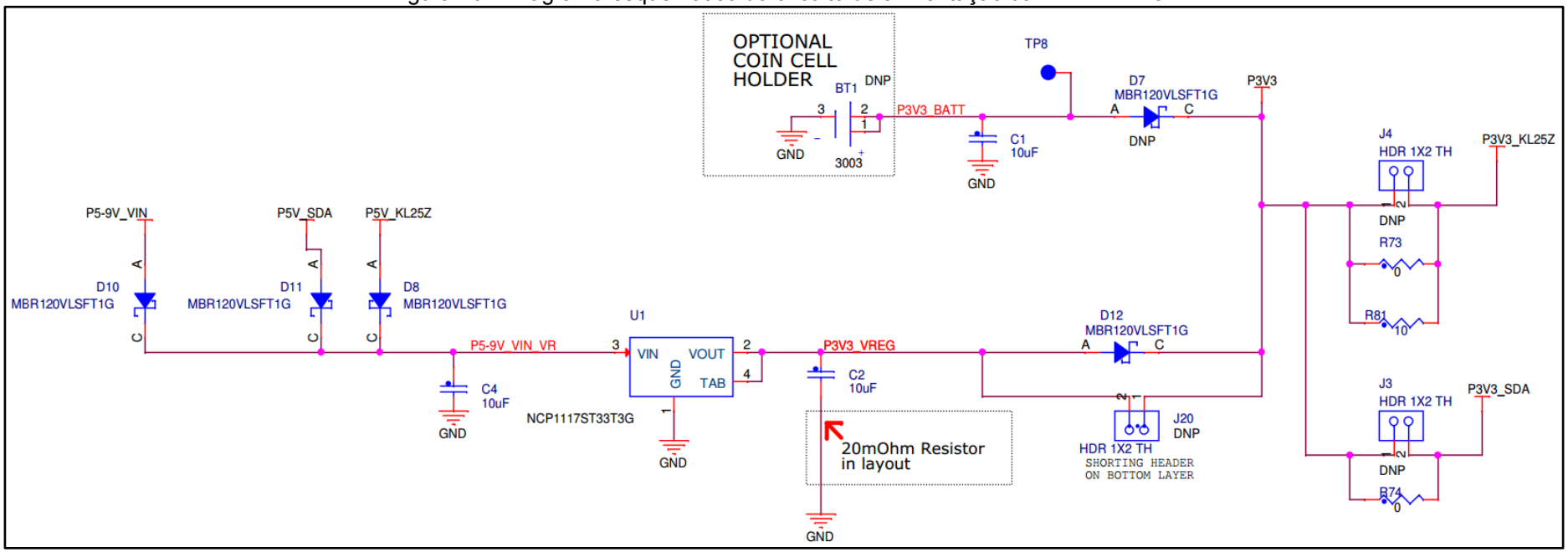
Tabela 4 – Opções de alimentação da FRDM-KL25Z

Fonte de Alimentação	Faixa da tensão de entrada	OpenSDA é utilizável?	Tensão de entrada é regulada a 3,3V?
Conector USB OpenSDA	5V	Sim	Sim
Conector USB KL25Z	5V	Não	Sim
Pino V_{IN}	4,3V ~ 9V	Não	Sim
Pino P3V3	1,71V ~ 3,6V	Não	Não
Bateria de célula tipo moeda	1,71V ~ 3,6V	Não	Não

Fonte: NXP. FRDM-KL25Z User's Manual (REV 2.0, 2013, p. 5)

Não ocorrerá dano na plataforma caso o usuário alimente a placa através de mais de um dos meios listados, pois diodos Schottky protegem cada uma das entradas de uma possível inversão de corrente. A Figura 16 evidencia os componentes utilizados através do diagrama esquemático do circuito de alimentação da plataforma:

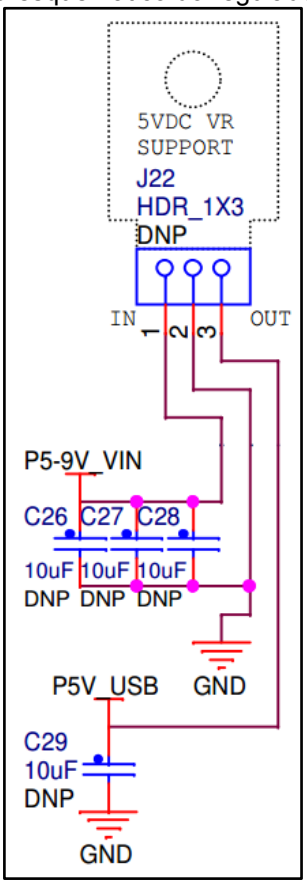
Figura 16 – Diagrama esquemático do circuito de alimentação da FRDM-KL25Z



Fonte: NXP. FRDM-KL25Z Schematics (REV E, 2013. p. 5)

Se a alimentação escolhida for através do pino V_{IN}, o usuário tem a opção de soldar um regulador de tensão (com saída de tensão fixa em 5V e no encapsulamento TO-220) no cabeçalho de pinos J22. Com a adição deste regulador é possível fornecer uma tensão de 5V para o pino 10 do cabeçalho de pinos J9. Sendo a alimentação realizada via qualquer um dos conectores USB, não há necessidade do uso do regulador para o fornecimento de 5V. A Figura 17 ilustra o diagrama esquemático do circuito do regulador:

Figura 17 – Diagrama esquemático do regulador de tensão opcional



Fonte: NXP. FRDM-KL25Z Schematics (REV E, 2013. p. 5)

Na adição deste regulador, é recomendável que os capacitores C26, C27, C28 e C29 sejam também soldados para prevenir o decaimento da tensão de saída sob cargas elevadas. Os valores das capacitâncias devem condizer com as especificações dadas pelo *datasheet* do regulador escolhido (valores ilustrados na Figura 17 são sugeridos pela NXP para o regulador de tensão 7805).

Através das ultimas duas figuras, pode-se observar todos os pontos de tensão na alimentação da FRDM-KL25Z. A Tabela 5 explica a função de cada um desses pontos:

Tabela 5 – Descrição dos pontos de tensão da alimentação da FRDM-KL25Z

Ponto de tensão	Descrição
P5-9V_VIN	Tensão de alimentação de entrada pelo pino V_{IN}
P5V_SDA	Tensão de alimentação de entrada pelo conector USB OpenSDA
P5V_KL25Z	Tensão de alimentação de entrada pelo conector USB KL25Z
P3V3_VREG	Tensão de 3,3V fornecida pelo regulador de tensão U1

Conclusão da Tabela 5 – Descrição dos pontos de tensão da alimentação da FRDM-KL25Z

P3V3_BATT	Tensão de 3,3V fornecida pela bateria de célula tipo moeda
P3V3	Tensão de alimentação principal na plataforma (pode prover de P3V3_VREG, P3V3_BATT ou do direto fornecimento pelo pino 8 do cabeçalho J9)
P3V3_KL25Z	Tensão de alimentação do microcontrolador
P3V3_SDA	Tensão de alimentação da interface OpenSDA
P5V_USB	Tensão de 5V fornecida ao pino 10 do cabeçalho J9 para o usuário alimentar alguma carga extra (pode prover de P5V_SDA, P5V_KL25Z ou do pino 3 do cabeçalho J22 relativo ao regulador de tensão opcional)

Fonte: NXP. FRDM-KL25Z User's Manual (REV 2.0, 2013, p. 7)

2.3.1 Tensão de referência analógica

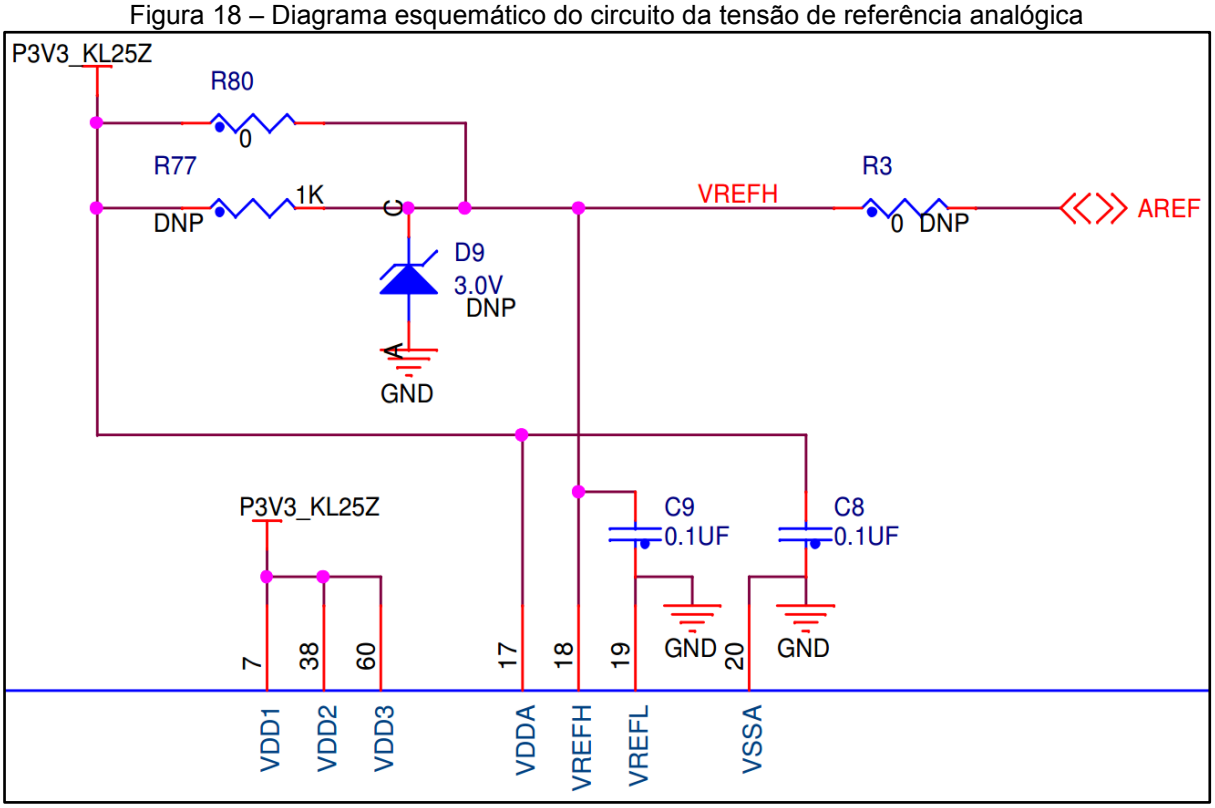
Para o correto uso do ADC do microcontrolador MKL25Z128VLK4 é necessário determinar quais os valores de tensão que serão aplicados aos sinais VREFH e VREFL. Estes sinais representam as tensões de referência analógica alta e baixa, respectivamente. A função deles é estabelecer quais são os limites da faixa de tensão do sinal analógico a ser lido. Através desses limites é possível aumentar ou diminuir a resolução em tensão do ADC. Como o ADC do MKL25Z128VLK4 possui 16 bits para a interpretação do sinal analógico, existem 2^{16} (ou 65536) valores associados à faixa de tensão dada por VREFH e VREFL. Por exemplo, se VREFL for 0V e VREFH for 3,3V, o ADC terá uma resolução em tensão de 50,354μV. A Equação (1) explica como é realizado este cálculo:

$$\text{Resolução em tensão} = \frac{\text{VREFH - VREFL}}{\text{Resolução em valores discretos}} = \frac{3,3V-0V}{2^{16}} = 50,354\mu V \quad (1)$$

Isso quer dizer que, no caso em que a faixa de tensão estabelecida está entre 0V e 3,3V, a mínima variação em tensão interpretável pelo ADC é de 50,354μV. Logicamente, se a faixa de tensão fosse reduzida, a resolução em tensão do ADC diminuiria, tornando a leitura mais precisa.

Por padrão de fábrica VREFH está conectado em P3V3_KL25Z (alimentação de 3,3V) e VREFL está aterrado (conectado em GND). A Figura 18 mostra o diagrama esquemático do circuito da tensão de referência analógica:

Figura 18 – Diagrama esquemático do circuito da tensão de referência analógica



Fonte: NXP. FRDM-KL25Z Schematics (REV E, 2013. p. 3)

Se o usuário deseja utilizar outra tensão para VREFH é possível remover o resistor R80 e soldar um resistor de 0Ω em R3. A alimentação de VREFH pode então ser realizada pelo pino 16 do cabeçalho J2 (sinal nomeado como AREF no diagrama esquemático da FRDM-KL25Z).

Na Figura 18 é possível verificar também que a alimentação para o funcionamento do ADC, representada pelas entradas VDDA e VSSA, é efetuada diretamente por P3V3_KL25Z (3,3V) e GND (terra), respectivamente.

2.4 A INTERFACE DE TOQUE CAPACITIVO

A plataforma FRDM-KL25Z está equipada com uma interface de toque capacitivo que utiliza como protocolo de comunicação dois sinais do tipo *Touch Sensing Input* (TSI). O TSI é um método de escaneamento capacitivo de eletrodos baseado em *hardware* desenvolvido pela NXP para algumas famílias de MCUs. A NXP também oferece suporte em *software* para o rápido desenvolvimento de programas que utilizam o protocolo TSI através de uma biblioteca denominada por *Touch Sensing Software* (TSS).

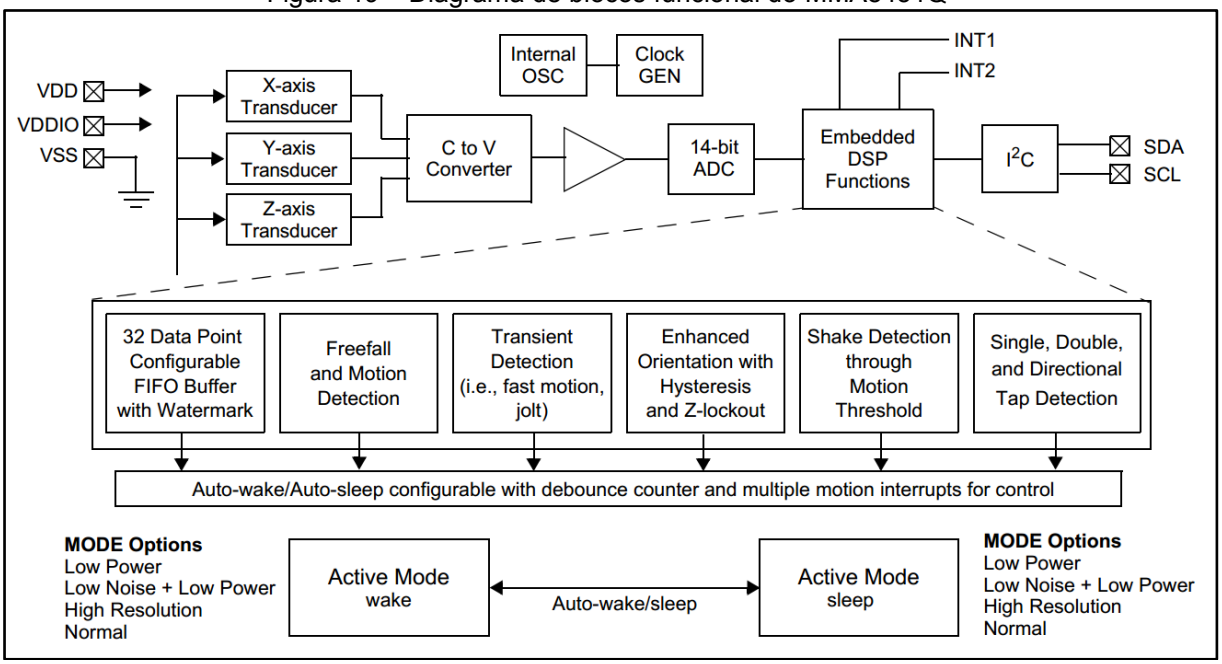
O TSI utiliza dois sinais oscilantes, um interno e outro externo. Quando o toque na interface altera a capacitância, o sinal interno permanece oscilando na mesma frequência, porém o sinal externo diminui sua frequência. Para a interpretação do toque uma contagem é realizada sobre a diferença nos números de períodos passados entre os dois sinais após certo tempo.

A conexão entre a interface de toque capacitivo e o MCU na plataforma FRDM-KL25Z é realizada através dos sinais TSI0_CH9 e TSI0_CH10 (pinos 51 e 52 do microcontrolador).

2.5 O ACELERÔMETRO MMA8451Q

A plataforma FRDM-KL25Z está equipada com o acelerômetro MMA8451Q, um CI fabricado pela NXP. O seu diagrama de blocos funcional está ilustrado na Figura 19:

Figura 19 – Diagrama de blocos funcional do MMA8451Q



Fonte: NXP. *Datasheet* do circuito integrado MMA8451Q (Rev. 10.1, 2016. p. 3).

Como se pode observar, este acelerômetro possui três transdutores: um para o eixo X, um para o eixo Y e outro para o eixo Z. A aceleração de cada eixo é convertida para tensão através dos transdutores. Logo após esse processo, as tensões dos eixos são filtradas através de filtros anti-*aliasing* e digitalizadas através

de ADCs (resolução de 14 bits). Os sinais digitais resultantes são processados através de um módulo DSP (para a serialização dos dados e detecção dos diversos modos de aceleração) e finalmente transmitidos ao microcontrolador através de um barramento I^2C.

O barramento I^2C emprega dois sinais, um para a transmissão de dados (o sinal SDA, bidirecional) e outro que estabelece a taxa de transmissão (o sinal SCL). O barramento propõe que os dispositivos operem em dois modos:

a) modo mestre – dispositivo que gera o sinal SCL e inicia a comunicação;
b) modo escravo – dispositivo que recebe o sinal SCL e responde.

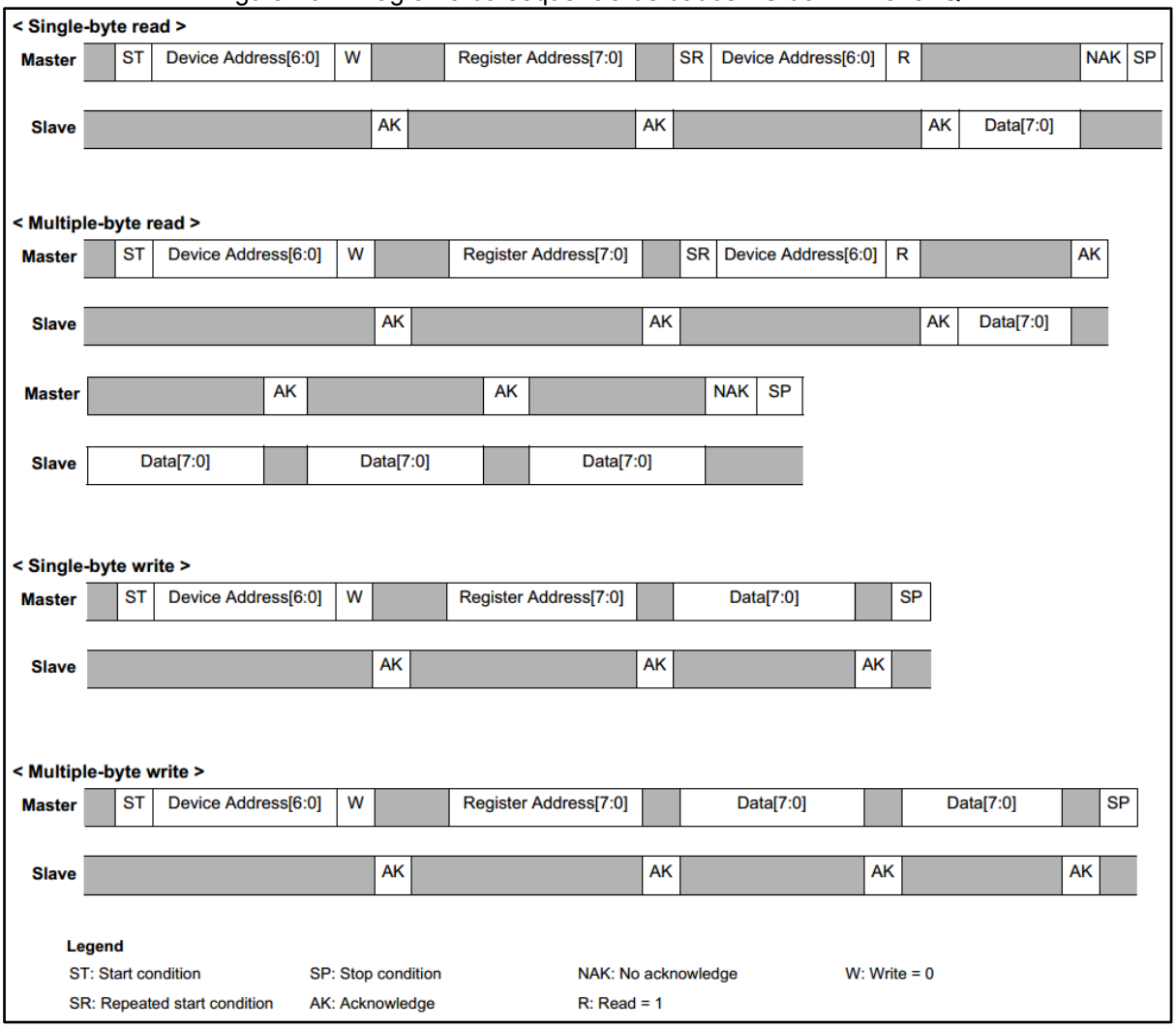
O MMA8451Q suporta frequências de até 400kHz pela linha SCL. Para a configuração do acelerômetro como mestre ou escravo é necessário especificar o nível de tensão no pino denominado como SA0. Tal pino representa o dígito menos significativo do endereço sequencial do acelerômetro, que pode ser 0011100 (0x1C em hexadecimal) ou 0011101 (0x1D em hexadecimal). Para modo escravo deve-se fornecer a SA0 tensão em nível lógico alto, e para modo mestre tensão em nível lógico baixo. Por padrão de fábrica a FRDM-KL25Z conecta SA0 em nível alto, ou seja, o acelerômetro opera em modo escravo, com endereço 0x1D.

Assim sendo, o microcontrolador envia por primeiro dados ao acelerômetro que fazem a requisição de escrita ou de leitura de algum registrador interno. Dentro das opções de escrita é possível determinar a função dos pinos INT1 e INT2 do acelerômetro. Estes pinos podem atuar como sinais de interrupção externa ao microcontrolador, indicando quando uma nova leitura de aceleração foi efetuada ou quando acorreu alguma ação específica (movimento, queda livre, transiente, orientação e “batida”). Na leitura dos registradores, o microcontrolador obtém informações relativas à aceleração do dispositivo. Cada registrador de leitura possui uma informação específica, como por exemplo, a última aceleração amostrada no eixo X. A lista completa das descrições e endereços dos registradores pode ser encontrada no *datasheet* do circuito integrado MMA8451Q.

Além disso, sendo o I^2C um barramento que utiliza dados em série, para o acesso a um registrador do acelerômetro é necessário enviar uma sequência de dados específicos. A Figura 20 evidencia como os dados devem ser transmitidos pelo mestre (neste caso o MCU) e como o escravo (neste caso o acelerômetro)

responde:

Figura 20 – Diagrama da sequência de dados I^2C do MMA8451Q

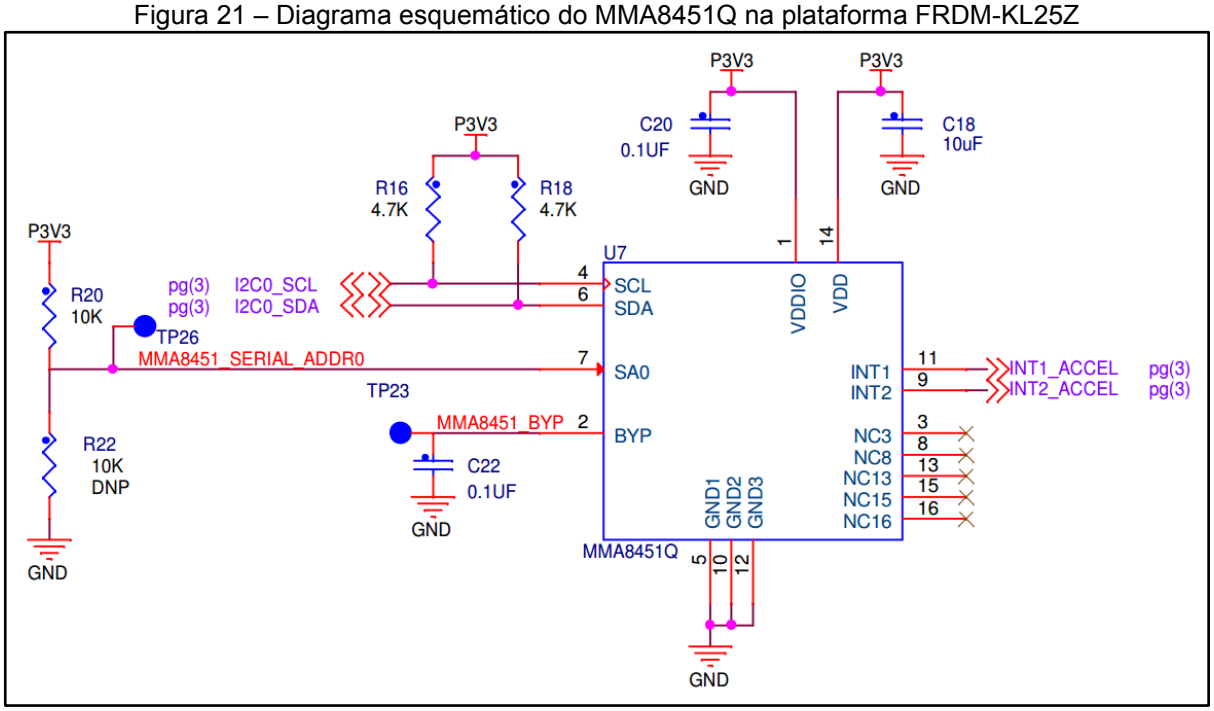


Fonte: NXP. *Datasheet* do circuito integrado MMA8451Q (Rev. 10.1, 2016. p. 19).

O sinal representado como ST é definido como uma transição de 1 para 0 na trilha SDA, enquanto SCL permanece em 1. SP é definido como uma transição de 0 para 1 na trilha SDA, enquanto SCL permanece em 1. SR é apenas uma repetição de ST.

Na Figura 21 é possível visualizar o diagrama esquemático do acelerômetro MMA8451Q na plataforma FRDM-KL25Z:

Figura 21 – Diagrama esquemático do MMA8451Q na plataforma FRDM-KL25Z



Fonte: NXP. FRDM-KL25Z Schematics (REV E, 2013. p. 5)

A Tabela 6 especifica como estão feitas as conexões entre os sinais do acelerômetro e os pinos do microcontrolador na plataforma FRDM-KL25Z:

Tabela 6 – Conexões entre sinais do acelerômetro e MCU

Sinal do MMA8451Q	Pino do MKL25Z128VLK4
SCL	PTE24
SDA	PTE25
INT1	PTA14
INT2	PTA15

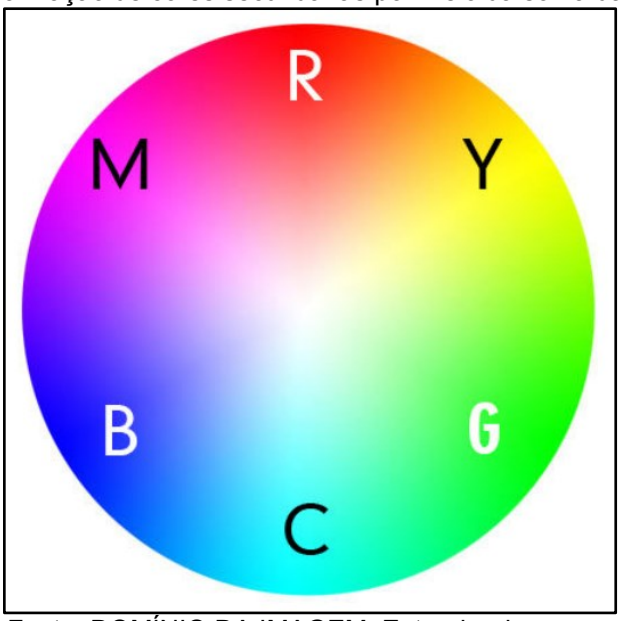
Fonte: NXP. FRDM-KL25Z User's Manual (REV 2.0, 2013, p. 12)

2.6 O LED RGB CLV1A-FKB-CJ1M1F1BB7R4S3

A plataforma FRDM-KL25Z dispõe de um LED RGB de PN CLV1A-FKB-CJ1M1F1BB7R4S3, fabricado pela empresa Cree Inc.. Este componente é composto por três diodos emissores de luz: um que emite luz vermelha (R), um que emite luz verde (G) e outro que emite luz azul (B). Estas três cores são chamadas de cores primárias do sistema aditivo, pois podem ser somadas para formar qualquer outra cor, dependendo apenas da intensidade da emissão de luz de cada uma. Para formar luz branca, por exemplo, é necessário que os três LEDs estejam emitindo a mesma intensidade de luz. Através da Figura 22 é possível verificar a relação de

intensidade entre as cores RGB para formar as outras cores:

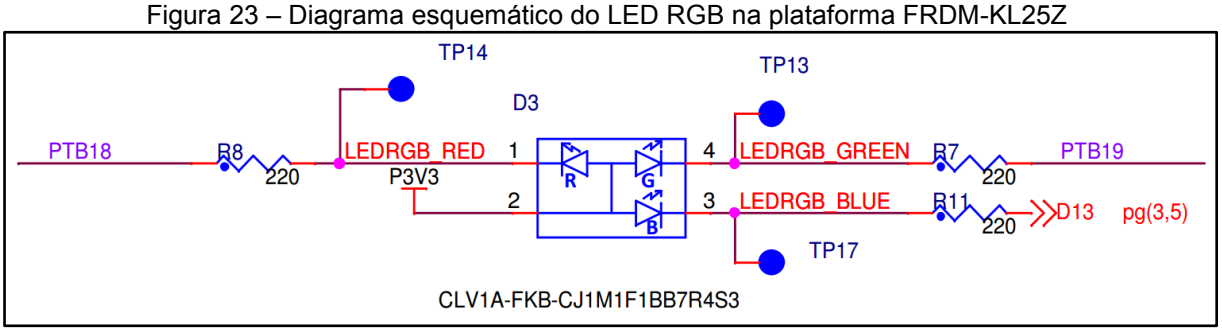
Figura 22 – Formação de cores secundárias por meio da soma das cores RGB



Fonte: DOMÍNIO DA IMAGEM. Entendendo os Espaços de Cores. Disponível em: <http://www.escoladominiodaimagem.com/single-post/2015/02/20/Entendendo-os-Espa%C3%A7os-de-Cores->. Acesso em: 28 set. 2016

Para regular a intensidade de luz emitida por cada diodo do LED RGB é possível aplicar sinais PWM através dos pinos do MCU, uma vez que a luminosidade compreendida pelo olho humano é correspondente ao valor eficaz da corrente que atravessa o diodo. No entanto, essa técnica é válida apenas a partir de frequências acima de 200Hz, pois o olho humano é capaz de perceber oscilações na iluminação quando a frequência está abaixo desse valor. Partindo dessa premissa, a plataforma FRDM-KL25Z conecta os diodos do LED RGB aos pinos PTB18 (LED vermelho), PTB19 (LED verde) e PTD1 (LED azul), visto que estes pinos podem exercer a funcionalidade PWM. A Figura 23 ilustra o diagrama esquemático do LED RGB na plataforma:

Figura 23 – Diagrama esquemático do LED RGB na plataforma FRDM-KL25Z



Fonte: NXP. FRDM-KL25Z Schematics (REV E, 2013. p. 3)

Verifica-se que para “acender” qualquer um dos LEDs é necessário fornecer uma tensão de 0V através do MCU, uma vez que os pinos estão conectados nos terminais de catodo. Logicamente, para “apagá-los” deve-se fornecer uma tensão de 3,3V. É importante observar que o mesmo pino do MCU que está conectado ao LED azul (PTD1), está também conectado ao pino 10 do cabeçalho J2 (chamado também de D13).

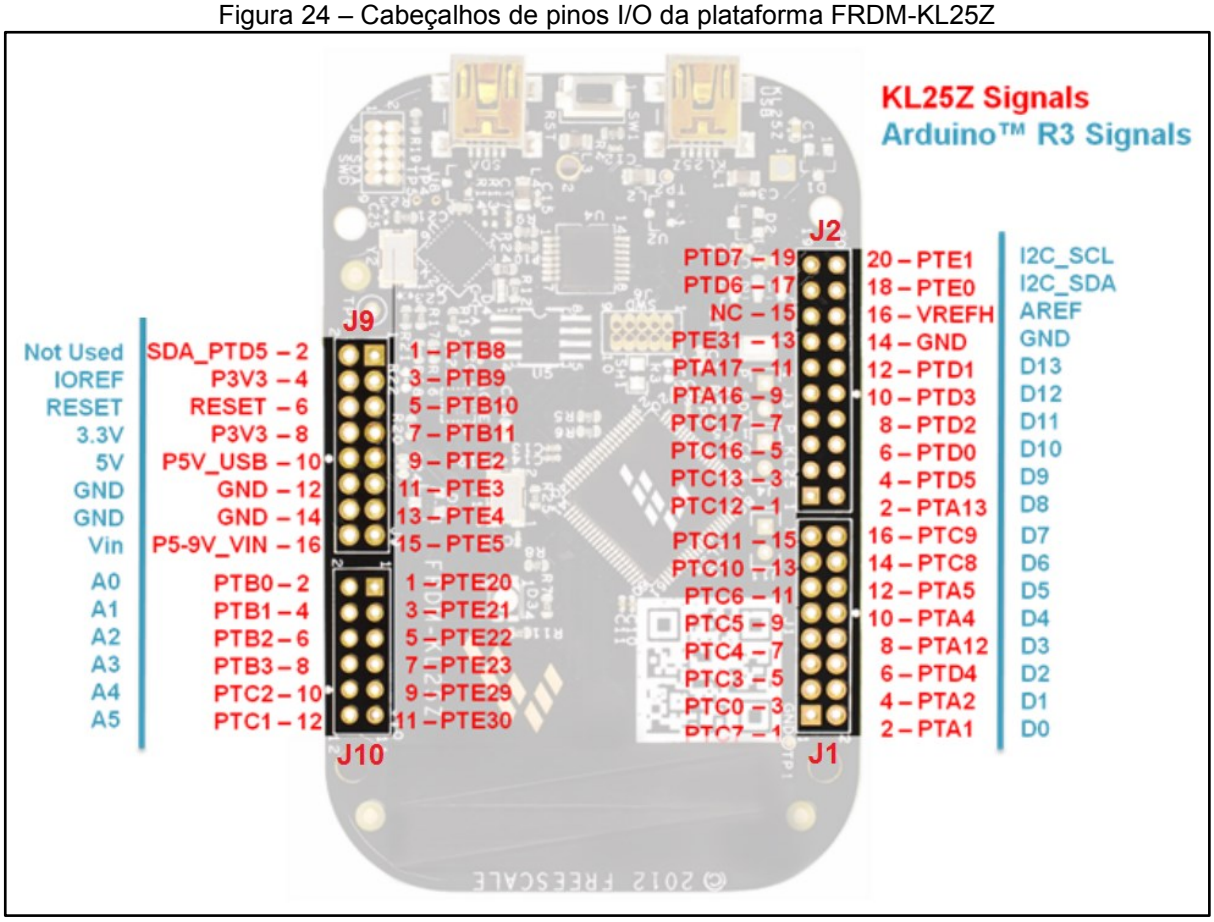
2.7 OS CABEÇALHOS DE PINOS I/O

Os cabeçalhos de pinos I/O da plataforma FRDM-KL25Z são os cabeçalhos denominados por J1, J2, J9 e J10. Apesar do nome, nem todos os pinos destes cabeçalhos estão conectados a pinos I/O do microcontrolador, sendo alguns conectados a sinais de alimentação da plataforma e outros em funcionalidades específicas do MCU.

Além disso, as colunas de pinos mais externas dos cabeçalhos (pinos de numeração par) oferecem compatibilidade mecânica e elétrica com Arduino Shields de leiaute no padrão Arduino *Revision* 3 (R3).

A Figura 24 exibe a relação entre a numeração dos pinos dos cabeçalhos I/O, nomes dos pinos no MCU, aplicação dos pinos na plataforma e nomes dos pinos no padrão Arduino R3:

Figura 24 – Cabeçalhos de pinos I/O da plataforma FRDM-KL25Z



Fonte: NXP. FRDM-KL25Z User's Manual (REV 2.0, 2013, p. 16)

É importante ressaltar que cada I/O suporta uma corrente de até 25mA e uma tensão máxima de 3,6V. De maneira geral, os pinos de 5V fornecem uma corrente máxima que depende da fonte de alimentação da plataforma (USB ou pino V_{IN}). Através do regulador de tensão o pino de 3,3V fornece uma corrente de 800mA menos a corrente consumida pelos dispositivos que estão em operação (varia conforme a aplicação).

Em uma aplicação em que o MCU esteja em operação com frequência de 48MHz, OpenSDA executando instruções via UART, LED RGB piscando com frequência próxima a 1Hz e acelerômetro ligado, mensura-se uma corrente total de 19,58mA consumida pela plataforma[1], restando 780,42mA para outras aplicações.

As principais funcionalidades disponibilizadas pelo MCU para os pinos dos cabeçalhos I/O da plataforma FRDM-KL25Z são evidenciadas pela Tabela 7:

[1]Informação fornecida por Erich Styger no artigo: **Tutorial: Using the FRDM-KL25Z as Low Power Board**. Disponível em: <https://mcuoneclipse.com/2013/10/20/tutorial-using-the-frdm-kl25z-as-low-power-board/>. Acesso em: 17 nov. 2016

Tabela 7 – Principais funcionalidades dos pinos dos cabeçalhos I/O

Cabeçalho e pino	Aplicação padrão na plataforma	Nome do pino no MCU	ALT0 (Padrão)	ALT1	ALT2	ALT3	ALT4	ALT5
J1 01	—	PTC7	CMP0_IN1	PTC7	SPI0_MISO	—	—	SPI0_MOSI
J1 02	—	PTA1	TSI0_CH2	PTA1	UART0_RX	FTM2_CH0	—	—
J1 03	—	PTC0	ADC0_SE14 / TSI0_CH13	PTC0	—	EXTRG_IN	—	CMP0_OUT
J1 04	—	PTA2	TSI0_CH3	PTA2	UART0_TX	FTM2_CH1	—	—
J1 05	—	PTC3	—	PTC3 / LLWU_P7	—	UART1_RX	FTM0_CH2	CLKOUT
J1 06	—	PTD4	—	PTD4 / LLWU_P14	SPI1_PCS0	UART2_RX	FTM0_CH4	—
J1 07	—	PTC4	—	PTC4 / LLWU_P8	SPI0_PCS0	UART1_TX	FTM0_CH3	—
J1 08	—	PTA12	—	PTA12	—	FTM1_CH0	—	—
J1 09	—	PTC5	—	PTC5 / LLWU_P9	SPI0_SCK	LPTMR0_ALT2	—	—
J1 10	—	PTA4	TSI0_CH5	PTA4	I2C1_SDA	FTM0_CH1	—	—
J1 11	—	PTC6	CMP0_IN0	PTC6 / LLWU_P10	SPI0_MOSI	EXTRG_IN	—	SPI0_MISO
J1 12	—	PTA5	—	PTA5	USB_CLKIN	FTM0_CH2	—	—
J1 13	—	PTC10	—	PTC10	I2C1_SCL	—	—	—
J1 14	—	PTC8	CMP0_IN2	PTC8	I2C0_SCL	FTM0_CH4	—	—
J1 15	—	PTC11	—	PTC11	I2C1_SDA	—	—	—
J1 16	—	PTC9	CMP0_IN3	PTC9	I2C0_SDA	FTM0_CH5	—	—
J2 01	—	PTC12	—	PTC12	—	—	FTM_CLKIN0	—
J2 02	—	PTA13	—	PTA13	—	FTM1_CH1	—	—
J2 03	—	PTC13	—	PTC13	—	—	FTM_CLKIN1	—
J2 04	—	PTD5	ADC0_SE6b	PTD5	SPI1_SCK	UART2_TX	FTM0_CH5	—
J2 05	—	PTC16	—	PTC16	—	—	—	—
J2 06	—	PTD0	—	PTD0	SPI0_PCS0	—	FTM0_CH0	—

Continuação da Tabela 7 – Principais funcionalidades dos pinos dos cabeçalhos I/O

J2 07	—	PTC17	—	PTC17	—	—	—	—
J2 08	—	PTD2	—	PTD2	SPI0_MOSI	UART2_RX	FTM0_CH2	SPI0_MISO
J2 09	—	PTA16	—	PTA16	SPI0_MOSI	—	—	SPI0_MISO
J2 10	—	PTD3	—	PTD3	SPI0_MISO	UART2_TX	FTM0_CH3	SPI0_MOSI
J2 11	—	PTA17	—	PTA17	SPI0_MISO	—	—	SPI0_MOSI
J2 12	LED Azul	PTD1	ADC0_SE5b	PTD1	SPI0_SCK	—	FTM0_CH1	—
J2 13	—	PTE31	—	PTE31	—	FTM0_CH4	—	—
J2 14	Alimentação	GND	—	—	—	—	—	—
J2 15	—	—	—	—	—	—	—	—
J2 16	Alimentação	VREFH	VREFH	—	—	—	—	—
J2 17	—	PTD6	ADC0_SE7b	PTD6 / LLWU_P15	SPI1_MOSI	UART0_RX	—	SPI1_MISO
J2 18	—	PTE1	—	PTE1	SPI1_MOSI	UART1_RX	—	SPI1_MISO
J2 19	—	PTD7	—	PTD7	SPI1_MISO	UART0_TX	—	SPI1_MOSI
J2 20	—	PTE0	—	PTE0	—	UART1_TX	RTC_CLKOUT	CMP0_OUT
J9 01	—	PTB8	—	PTB8	—	EXTRG_IN	—	—
J9 02	—	SDA_PTD5	—	—	—	—	—	—
J9 03	—	PTB9	—	PTB9	—	—	—	—
J9 04	Alimentação	P3V3	—	—	—	—	—	—
J9 05	—	PTB10	—	PTB10	SPI1_PCS0	—	—	—
J9 06	Reset	PTA20	—	PTA20	—	—	—	—
J9 07	—	PTB11	—	PTB11	SPI1_SCK	—	—	—
J9 08	Alimentação	P3V3	—	—	—	—	—	—
J9 09	—	PTE2	—	PTE2	SPI1_SCK	—	—	—
J9 10	Alimentação	P5V_USB	—	—	—	—	—	—
J9 11	—	PTE3	—	PTE3	SPI1_MISO	—	—	SPI1_MOSI
J9 12	Alimentação	GND	—	—	—	—	—	—
J9 13	—	PTE4	—	PTE4	SPI1_PCS0	—	—	—
J9 14	Alimentação	GND	—	—	—	—	—	—
J9 15	—	PTE5	—	PTE5	—	—	—	—
J9 16	Alimentação	P5-9V_VIN	—	—	—	—	—	—

Conclusão da Tabela 7 – Principais funcionalidades dos pinos dos cabeçalhos I/O

J10 01	—	PTE20	ADC0_DP0 / ADC0_SE0	PTE20	—	FTM1_CH0	UART0_TX	—
J10 02	—	PTB0	ADC0_SE8 / TSI0_CH0	PTB0 / LLWU_P5	I2C0_SCL	FTM1_CH0	—	—
J10 03	—	PTE21	ADC0_DM0 / ADC0_SE4a	PTE21	—	FTM1_CH1	UART0_RX	—
J10 04	—	PTB1	ADC0_SE9 / TSI0_CH6	PTB1	I2C0_SDA	FTM1_CH1	—	—
J10 05	—	PTE22	ADC0_DP3 / ADC0_SE3	PTE22	—	FTM2_CH0	UART2_TX	—
J10 06	—	PTB2	ADC0_SE12 / TSI0_CH7	PTB2	I2C0_SCL	FTM2_CH0	—	—
J10 07	—	PTE23	ADC0_DM3 / ADC0_SE7a	PTE23	—	FTM2_CH1	UART2_RX	—
J10 08	—	PTB3	ADC0_SE13 / TSI0_CH8	PTB3	I2C0_SDA	FTM2_CH1	—	—
J10 09	—	PTE29	CMP0_IN5 / ADC0_SE4b	PTE29	—	FTM0_CH2	FTM_CLKIN0	—
J10 10	—	PTC2	ADC0_SE11 / TSI0_CH15	PTC2	I2C1_SDA	—	FTM0_CH1	—
J10 11	—	PTE30	DAC0_OUT / ADC0_SE23 / CMP0_IN4	PTE30	—	FTM0_CH3	FTM_CLKIN1	—
J10 12	—	PTC1	ADC0_SE15 / TSI0_CH14	PTC1 / LLWU_P6 / RTC_CLKIN	I2C1_SCL	—	FTM0_CH0	—

Fonte: NXP

O Quadro 3 mostra a compatibilidade de funcionalidades entre pinos Arduino R3 e pinos da plataforma FRDM-KL25Z:

Quadro 3 – Compatibilidade de funcionalidades dos pinos Arduino R3

Plataforma	Pino	UART	PWM	I/O	Interrup.	I^2C	SPI	ADC
Arduino R3	D0	RX	Não	Sim	Sim	—	—	—
FRDM-KL25Z	PTA1	RX0	Sim	Sim	Sim	—	—	—
Arduino R3	D1	TX	Não	Sim	Sim	—	—	—
FRDM-KL25Z	PTA2	TX0	Sim	Sim	Sim	—	—	—
Arduino R3	D2	—	Não	Sim	Sim	—	—	—
FRDM-KL25Z	PTD4	—	Sim	Sim	Sim	—	—	—
Arduino R3	D3	—	Sim	Sim	Sim	—	—	—
FRDM-KL25Z	PTA12	—	Sim	Sim	Sim	—	—	—
Arduino R3	D4	—	Não	Sim	Sim	—	—	—
FRDM-KL25Z	PTA4	—	Sim	Sim	Sim	—	—	—
Arduino R3	D5	—	Sim	Sim	Sim	—	—	—
FRDM-KL25Z	PTA5	—	Sim	Sim	Sim	—	—	—
Arduino R3	D6	—	Sim	Sim	Sim	—	—	—
FRDM-KL25Z	PTC8	—	Sim	Sim	Não	SCL0	—	—
Arduino R3	D7	—	Não	Sim	Sim	—	—	—
FRDM-KL25Z	PTC9	—	Sim	Sim	Não	SDA0	—	—
Arduino R3	D8	—	Não	Sim	Sim	—	—	—
FRDM-KL25Z	PTA13	—	Sim	Sim	Sim	—	—	—
Arduino R3	D9	—	Sim	Sim	Sim	—	—	—
FRDM-KL25Z	PTD5	—	Sim	Sim	Sim	—	—	—
Arduino R3	D10	—	Sim	Sim	Sim	—	CS	—
FRDM-KL25Z	PTD0	—	Sim	Sim	Sim	—	CS0	—
Arduino R3	D11	—	Não	Sim	Sim	—	MOSI	—
FRDM-KL25Z	PTD2	RX2	Sim	Sim	Sim	—	MOSI0	—
Arduino R3	D12	—	Não	Sim	Sim	—	MISO	—
FRDM-KL25Z	PTD3	TX2	Sim	Sim	Sim	—	MISO0	—
Arduino R3	D13	—	Não	Sim	Sim	—	SCK	—
FRDM-KL25Z	PTD1	—	Sim	Sim	Sim	—	SCK0	ADC0_SE5b
Arduino R3	D14	—	—	Sim	Sim	SDA	—	A4
FRDM-KL25Z	PTE0	—	—	Sim	Não	SDA1	—	—
Arduino R3	D15	—	—	Sim	Sim	SCL	—	A5
FRDM-KL25Z	PTE1	—	—	Sim	Não	SCL1	—	—
Arduino R3	A0	—	—	Sim	Sim	—	—	A0
FRDM-KL25Z	PTB0	—	—	Sim	Não	SCL0	—	ADC0_SE8
Arduino R3	A1	—	—	Sim	Sim	—	—	A1
FRDM-KL25Z	PTB1	—	—	Sim	Não	SDA0	—	ADC0_SE9
Arduino R3	A2	—	—	Sim	Sim	—	—	A2
FRDM-KL25Z	PTB2	—	—	Sim	Não	SCL0	—	ADC0_SE12
Arduino R3	A3	—	—	Sim	Sim	—	—	A3
FRDM-KL25Z	PTB3	—	—	Sim	Não	SDA0	—	ADC0_SE13
Arduino R3	A4	—	—	Sim	Sim	SDA	—	A4
FRDM-KL25Z	PTC2	—	—	Sim	Não	SDA1	—	ADC0_SE11
Arduino R3	A5	—	—	Sim	Sim	SCL	—	A5
FRDM-KL25Z	PTC1	—	—	Sim	Não	SCL1	—	ADC0_SE15

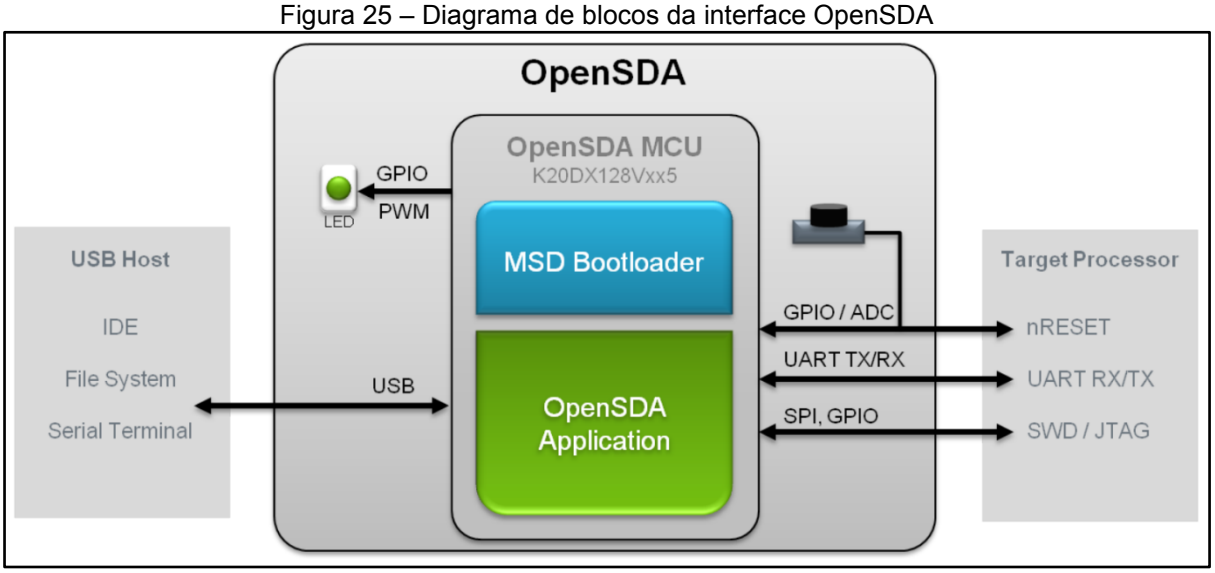
Fonte: NXP

Observa-se que, com exceção de algumas funções de interrupção e ADC, a plataforma FRDM-KL25Z dispõe das mesmas funcionalidades (ou mais) requisitadas pelos pinos do padrão Arduino R3.

2.8 A INTERFACE OPENSDA

A interface OpenSDA é um adaptador serial e de depuração. Ela realiza papel de ponte na comunicação entre o MCU da plataforma FRDM-KL25Z e um computador conectado via USB. A depuração é comandada através dos sinais da interface SWD do microcontrolador. Para a comunicação serial o OpenSDA implementa uma porta serial virtual através de uma interface USB CDC, tendo como conexão primária os pinos PTA1 e PTA2 (UART0_RX e UART0_TX, respectivamente) do MCU. A Figura 25 ilustra o diagrama de blocos da interface:

Figura 25 – Diagrama de blocos da interface OpenSDA



Fonte: NXP. FRDM-KL25Z User's Manual (REV 2.0, 2013, p. 8)

Seu circuito tem como base um microcontrolador da família Kinetis K20 (núcleo ARM Cortex-M4), que possui 128kB de memória flash e um controlador USB integrado. Além disso, a interface dispõe de um *bootloader* MSD, que provê um mecanismo rápido e fácil para carregar as suas funções. O circuito inclui um LED (D4) de status e um botão (SW1) para o reset do MCU MKL25Z128VLK. O botão também pode ser usado para fazer a interface OpenSDA entrar no modo de *bootloader* (pressionando-o durante a conexão com o computador via USB).

3 FRDM-KL25Z: PROGRAMAÇÃO, CONFIGURAÇÃO E USO

Neste capítulo é discutido o uso dos recursos de *software* disponibilizados para a plataforma FRDM-KL25Z, com o objetivo de apresentar e comparar os meios de desenvolvimento para a programação do MCU. Também será apresentado um tutorial de instalação do pacote KSDK e a IDE KDS, com os passos de configurações necessários para a operação correta da plataforma. Por último será executado e brevemente descrito o programa “bubble level” (fornecido como exemplo pela NXP), que implementa funções de uso do acelerômetro, LED RGB e UART via OpenSDA.

3.1 MEIOS DE DESENVOLVIMENTO

De maneira geral o processo de programação de um *software* para um microcontrolador envolve quatro etapas: primeiramente deve ser escrito o código desejado através de instruções em alguma linguagem de programação; em seguida é necessário realizar a compilação do código escrito; após isso é preciso ligar os arquivos objetos gerados na compilação para formar o arquivo executável final; por último deve-se gravar/depurar o arquivo executável no MCU. Existem diversas formas de realizar esses quatro passos, no entanto *softwares* denominados pelo acrônimo de IDE (em português: Ambiente de Desenvolvimento Integrado) possibilitam a execução de todos (ou quase todos) os passos em um só ambiente. Nos parágrafos que seguem são explicadas algumas formas possíveis para a elaboração de cada passo, evidenciando as formas mais usuais, eficientes e recomendadas pela NXP para o microcontrolador da FRDM-KL25Z.

Para a escrita dos códigos é possível utilizar qualquer linguagem de programação. No entanto, geralmente as aplicações para microcontroladores requerem o uso direto e específico de seus registradores e endereços internos, além do manuseio dos níveis de tensão. Assim, é mais fácil utilizar uma linguagem de programação de mais baixo nível para escrever os códigos. Três linguagens muito usadas para a escrita de programas para MCU hoje são Assembly, C e C++. Neste sentido, C é uma linguagem intermediária, pois ainda que seja fácil a manipulação dos registradores (como em Assembly), todavia é possível utilizar funções que realizam operações de mais alto nível (bastante usais em C++).

Através desse raciocínio, muitas empresas adotam a linguagem C na implementação dos *drivers* para os periféricos de seus MCUs. *Driver*, neste contexto, é um conjunto de instruções que permite a configuração, inicialização e manipulação de forma abstrata sobre algum periférico. Os periféricos são circuitos do microcontrolador que realizam determinadas tarefas e podem ser utilizados diretamente pelo usuário (exemplos: pinos I/O, ADC, DAC, RTC, *timers*, interface UART, interface SPI, interface I²C, etc). Assim sendo, a escrita dos códigos é facilitada através do uso dos *drivers* disponibilizados pelas empresas. Normalmente os *drivers* na linguagem C são funções pré-definidas que proporcionam o direcionamento do nível de tensão em registradores ou endereços específicos do MCU. Dessa forma, não á necessidade do usuário ler o manual do microcontrolador para saber qual o endereço a se referenciar para realizar alguma operação em um periférico.

No caso do MCU da plataforma FRDM-KL25Z a NXP recomenda dois possíveis caminhos como facilitadores para a escrita dos códigos: KSDK ou ARM mbed. O KSDK é um pacote de arquivos desenvolvido pela NXP que disponibiliza todos os *drivers* em linguagem C para os microcontroladores da família Kinetis, além de códigos exemplos (como é o caso do programa “bubble level”) que utilizam estes *drivers*. O pacote KSDK pode ser baixado pelo *website* da NXP gratuitamente. Por outro lado, o ARM mbed é uma IDE gratuita, de uso *online* (apenas via *website*), desenvolvida pela ARM e que desempenha várias tarefas. Dentre estas tarefas estão: um editor de texto para a escrita dos códigos; disponibilização de *drivers* para certos microcontroladores com núcleo ARM (incluindo o MCU da FRDM-KL25Z), facilitando a escrita; compilação do código e ligação dos arquivos objetos, gerando um arquivo executável final; uma vasta gama de exemplos de programas prontos, visto que o objetivo de esta IDE ser *online* é justamente despertar uma comunidade desenvolvedora ativa que compartilha seus códigos. No entanto, apesar do ARM mbed demonstrar muitas vantagens no seu uso, possui algumas deficiências, como a limitação de *drivers* (não são todos que estão implementados) e a falta da parte de gravação e depuração dos programas. Por este motivo, escolheu-se utilizar como referência neste trabalho o pacote KSDK.

Como explicado anteriormente, o pacote KSDK apenas fornece arquivos contendo *drivers* e exemplos de códigos. Para as demais atividades no processo de programação (escrita dos códigos, compilação, ligação, gravação e depuração) é

necessário utilizar um (ou mais de um) programa(s). A NXP recomenda que se utilize o KSDK em conjunto com alguma IDE que tenha todas estas capacidades. Uma opção de IDE de uso gratuito com estes requisitos é o KDS, desenvolvido e distribuído pela própria NXP. O KDS está construído sobre *softwares* livres de código aberto, incluindo o Eclipse, o GNU *Compiler Collection* (GCC) e o GNU *Debugger* (GDB). Além disso, não possui limitações de tamanho de código. Assim sendo, optou-se utilizar o KDS por ser uma IDE pronta a atender gratuitamente todas as necessidades no desenvolvimento de *software* para microcontroladores da família Kinetis.

Um exemplo de IDE alternativa ao KDS seria a IDE µVision, desenvolvida e distribuída pela ARM. O µVision faz parte do conjunto de ferramentas Keil da ARM que incluí: compiladores, IDEs, sistemas operacionais para embarcados, adaptadores para depuração e plataformas de desenvolvimento. Apesar de o µVision ser uma excelente IDE, há uma limitação no tamanho de código (32 kB) na sua versão gratuita. Por este motivo, não foram usadas as ferramentas da ARM.

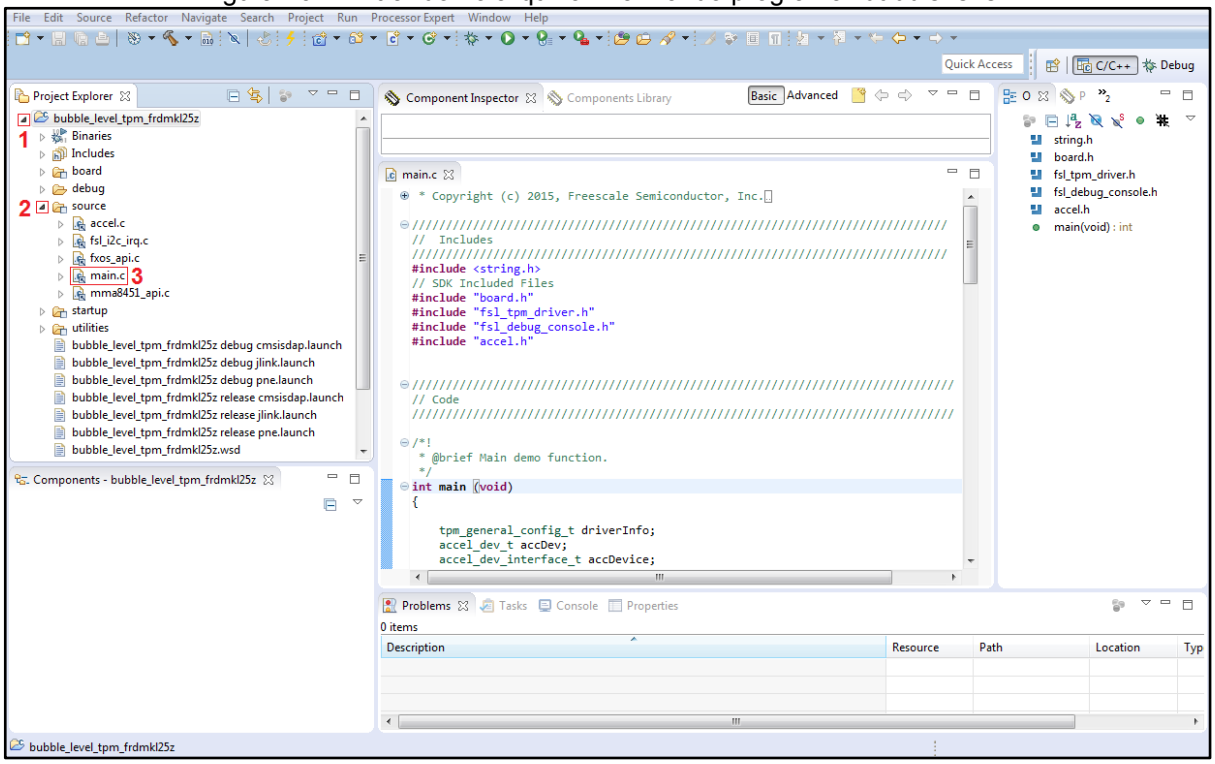
3.2 TUTORIAL DE DESENVOLVIMENTO

Esta sessão do trabalho está toda descrita no Apêndice A, por ser bastante extensa e conter muitas ilustrações.

3.3 BREVE DESCRIÇÃO DO PROGRAMA “BUBBLE LEVEL”

Clicando sobre a seta localizada ao lado esquerdo da pasta que representa o projeto “bubble_level_tpm_frdmkl25z” (dentro da *view* Project Explorer) é possível visualizar os arquivos de código pertencentes ao projeto. Após isso, clique também na seta da pasta “source” e depois dê duplo clique no arquivo “main.c”. A Figura 26 destaca estes passos:

Figura 26 – Entrando no arquivo “main.c” do programa “bubble level”



Fonte: Elaborado pelo autor

O “main.c” é o arquivo principal na linguagem C e é onde o usuário começa a escrever os seu próprios códigos. O usuário também tem a possibilidade de alterar e escrever códigos em todos os arquivos do projeto, porém é dentro do “main.c” que o microcontrolador inicia as atividades após ter sido configurado. Esta configuração inicial (antes do programa entrar no arquivo main.c”) é realizada através dos arquivos que estão dentro da pasta “startup”. Esses arquivos introduzem ao microcontrolador todas as informações necessárias para que ele funcione adequadamente. Eles também informam ao microcontrolador que é necessário fazer a execução da função “main” dentro do arquivo “main.c” após as configurações terem ocorrido. O microcontrolador “saberá” que os arquivos dentro da pasta “startup” devem ser os primeiros a serem executados, pois a IDE passa essa referência a alguns endereços de sua memória (e possível verificar todas essas especificações nas propriedades do projeto). Esta é uma das maiores vantagens na utilização de uma IDE sugerida pela empresa que fabrica o microcontrolador. Sempre que se criar um novo projeto no KDS, será perguntado ao usuário qual é o microcontrolador alvo do programa, justamente para que a IDE realize todas essas configurações automaticamente.

Voltando ao programa “bubble level”, o arquivo “main.c” declara nas suas

primeira linhas alguns arquivos do tipo “header” (nome_do_arquivo.h) para utilizar funções que estão declaradas em outros arquivos. Este é o princípio de como são referenciados os *drivers* do KSDK. Os *headers* no “main.c” do “bubble level” foram declarados no intuito de utilizar funções que inicializam alguns componentes da plataforma, *drivers* dos periféricos TPM e UART e o acelerômetro.

A função “main” é responsável por todas as operações descritas a seguir. Primeiramente são definidas e inicializadas variáveis que serão utilizadas na manipulação do módulo TPM, do acelerômetro e do módulo UART (comunicação serial).

Logo após isso, a função “hardware_init” habilita o *clock* nas portas do MCU, configura os pinos I²C para manuseio do acelerômetro, configura os pinos e o *clock* dos dois módulos TPM para o manuseio do LED RGB (apenas vermelho e verde), habilita o *clock* na plataforma e inicializa o módulo UART. É possível encontrar as ações dessas operações internamente à função “hardware_init” clicando com o botão direito do *mouse* sobre ela e depois não opção “Open Declaration”. Assim, o usuário é redirecionado para o arquivo onde ocorre a declaração da função (neste caso dentro do arquivo “hardware_init.c”). Sempre que necessário entender o que alguma função faz pode-se utilizar a opção “Open Declaration”.

Em seguida, a função “OSA_Init” inicializa uma camada de abstração de serviços que são necessários para o uso do acelerômetro, a função “PRINTF” faz a transmissão via UART da frase "Bubble Level Demo!”, a função “memset” prepara a memória do MCU para a inicialização do módulo TPM, a função “TPM_DRV_Init” inicializa o módulo e finalmente a função “TPM_DRV_SetClock” especifica a fonte de *clock* para o módulo.

Finalmente, após todas essas inicializações e configurações o programa entra no laço principal, definido pela função “while(1)”. A função “while” escrita dessa forma especifica que “enquanto verdade” o programa deve permanecer dentro do laço, ou seja, toda vez que a última operação for executada o programa volta para a primeira operação do laço. Assim, o programa repete sequencialmente as operações do laço até o microcontrolador ser desligado. Dentro do laço principal do “bubble level” as seguintes operações são executadas em sequencia: função de espera de 5ms; busca de novos dados do acelerômetro, inicialização do módulo TPM com as últimas configurações de *duty cycle* (para alterar a intensidade luminosa dos LEDs do LED RGB); decodificação do último dado recebido pelo acelerômetro em dados

de rotação nos eixos X e Y; conversão dos dados de rotação em ângulos de 0º a 90º; tratamento dos dados dos ângulos; atualização do *duty cycle* dos canais do módulo TPM através de uma operação matemática com os dados dos ângulos; transmissão via UART dos dados de rotação do acelerômetro.

4 PROJETO E PRODUÇÃO DO HARDWARE

Neste capítulo são explicados todos os procedimentos de projeto e produção de um *hardware* para uso como interface de usuário compatível com a plataforma FRDM-KL25Z. Primeiramente são destacados os componentes propostos no desenvolvimento da interface e as discussões que envolvem o custo-benefício e a correta implementação das partes, de forma que atendam as especificações da plataforma. Após isso, é apresentado o diagrama esquemático do circuito proposto e o leiaute da placa através do *software* Altium Designer. A localização escolhida para cada componente e as características das trilhas na placa também são elucidadas. Após isso, é exposta a lista dos componentes necessários e os custos envolvidos. Por último é comentada a produção da PCI.

4.1 DISCUSSÕES SOBRE OS COMPONENTES

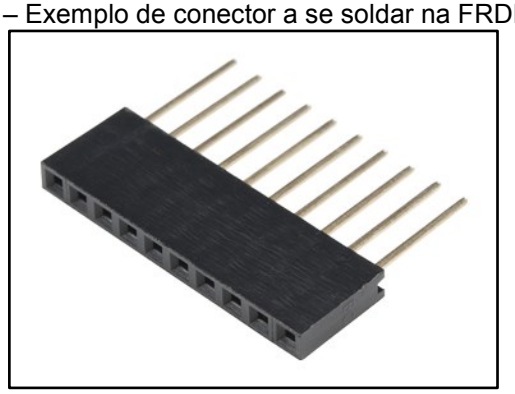
Para que a interface de usuário seja proveitosa no aprendizado dos estudantes de Microprocessadores I, pensou-se em utilizar componentes relevantes para a didática da disciplina. Esses componentes, segundo o professor orientador deste trabalho, devem implementar funções como: conectividade com a plataforma FRDM-KL25Z, leitura de tensão elétrica em pinos I/O, uso de módulos TPM (*Timer*/PWM), interrupção externa, conversão de dados via ADC e DAC, comunicação via UART, além do controle de um display LCD de duas linhas por 16 caracteres. A condição de escolha dos componentes envolveu principalmente a busca pelos menores custos e a simplicidade de implementação, pois, como já mencionado anteriormente, é pretendida a produção de oito (ou mais) placas para uso em laboratório. As subseções a seguir explanam sobre os componentes escolhidos para cada uma das funcionalidades citadas.

4.1.1 Conectividade com a plataforma FRDM-KL25Z

Pensou-se em uma solução na qual a plataforma poderia ser facilmente desconectada da placa de interface de usuário e que, ao mesmo tempo, possibilitasse a conexão de Arduino Shields sobre ela. Também foi considerada a importância de se poder utilizar e visualizar os dispositivos da FRDM-KL25Z (como o

botão de *reset*, a interface de toque capacitivo e o LED RGB). Levando em conta esses quesitos, propôs-se a conectividade da plataforma por cima da placa de interface. Ademais, para propiciar a facilidade de conexão e desconexão da interface e a possibilidade de conexões de Arduino Shield sobre a FRDM-KL25Z, optou-se por soldar na plataforma conectores do tipo barra de pinos fêmea em 180º, com pinos inferiores longos. Esse é o mesmo tipo de conector utilizado em Arduino Shields, e pode ser visualizado na Figura 27:

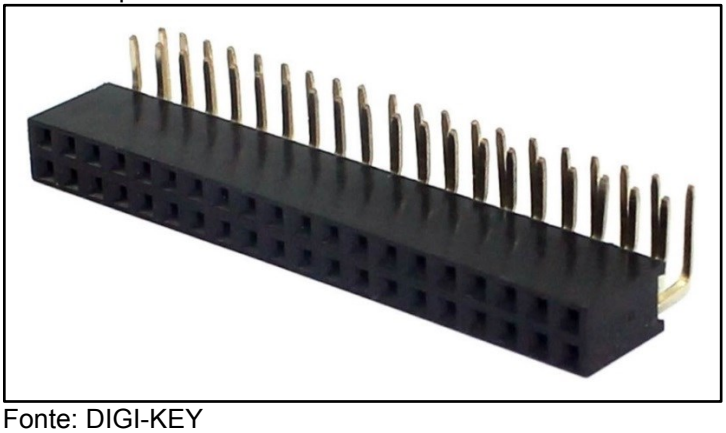
Figura 27 – Exemplo de conector a se soldar na FRDM-KL25Z



Fonte: ROBOCORE

Na data 21/11/2016 foi possível encontrar esses componentes no *website* da ROBOCORE, uma empresa brasileira de bastante renome na venda via internet de componentes relativos a Arduino. Infelizmente, por não se ter encontrado esse tipo de conector nas lojas locais de Porto Alegre, utilizou-se provisoriamente um tipo de conector alternativo: conector barra de pinos fêmea em 90º. Este conector está ilustrado na Figura 28:

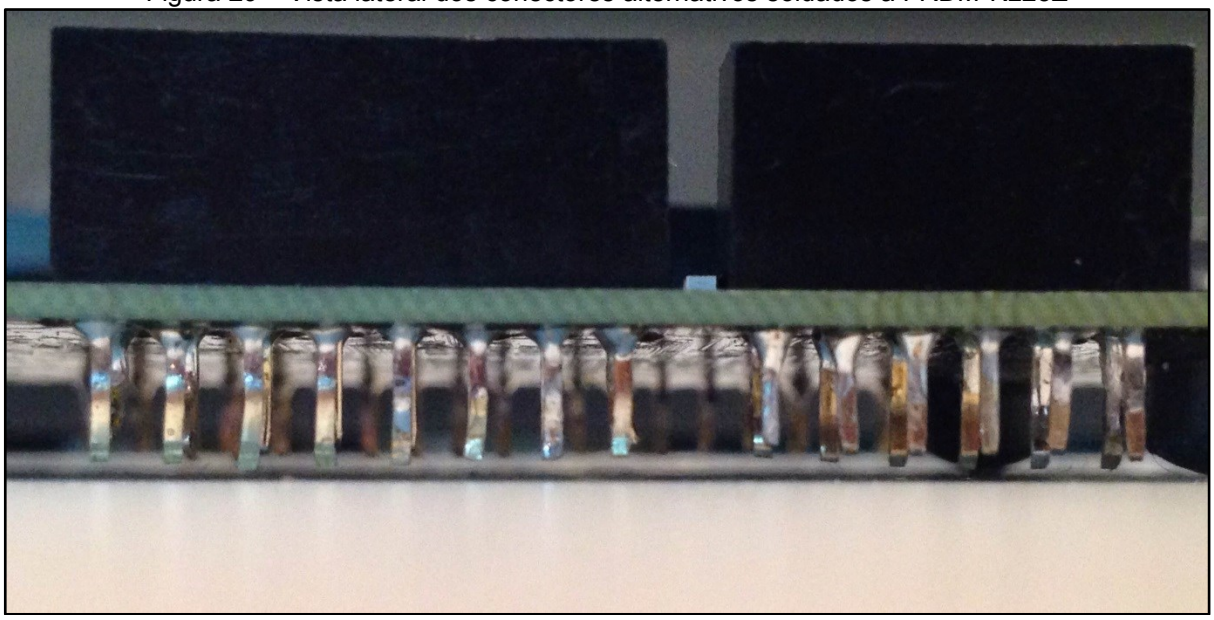
Figura 28 – Exemplo de alternativa de conector a se soldar na FRDM-KL25Z



Fonte: DIGI-KEY

Esse tipo de conector é de fácil localização comercial e com ele foi possível obter pinos inferiores longos ao se “endireitar” os pinos até o ângulo de 180º com um alicate do tipo bico fino. Após isso, emparelhou-se a altura dos pinos com um alicate de corte. O resultado, após soldar os conectores na plataforma, observa-se na Figura 29:

Figura 29 – Vista lateral dos conectores alternativos soldados à FRDM-KL25Z

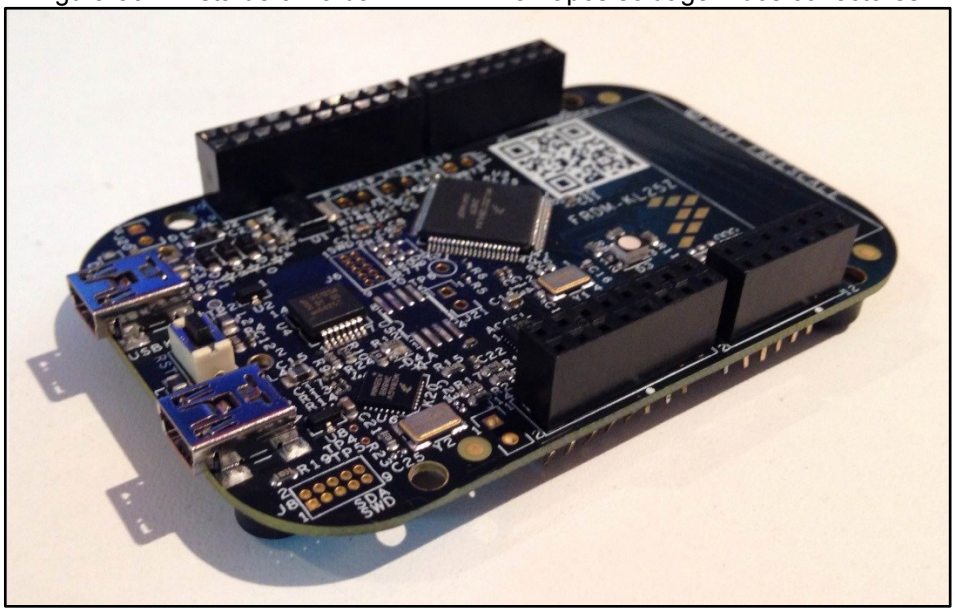


Fonte: Elaborado pelo autor

Essa alteração do conector foi realizada apenas para ser possível prototipar e testar a interface de usuário. Enfatiza-se que o tipo de conector correto para uso no projeto é o da Figura 27, e não o da Figura 28.

É muito importante observar que os pinos não encostam na mesa. Isso ocorre graças aos pés de apoio da plataforma, pois são mais altos que os pinos. É importante cortar os pinos em uma altura menor que os pés de apoio, de forma que não encostem na mesa. Esse corte (para o ajuste da altura) deve ser sempre realizado, mesmo quando o conector utilizado for o correto (Figura 27). A Figura 30 exibe a vista de cima da plataforma, após a soldagem dos conectores:

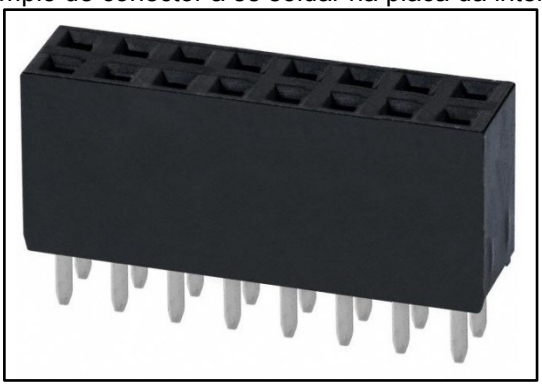
Figura 30 – Vista de cima da FRDM-KL25Z após soldagem dos conectores



Fonte: Elaborado pelo autor

Finalmente, o tipo de conector a ser usado na placa da interface de usuário deve proporcionar uma forma em que a FRDM-KL25Z seja conectada através dos conectores já soldados. Para tal, é possível utilizar conectores do tipo barra de pinos fêmea em 180º, com pinos inferiores curtos. Esse tipo de conector é de fácil localização comercial e está ilustrado na Figura 31:

Figura 31 – Exemplo de conector a se soldar na placa da interface de usuário



Fonte: DIGI-KEY

Uma observação importante é que a placa a ser utilizada na conexão com a plataforma não pode ser do tipo padrão de prototipação (somente com ilhas). Isso se deve ao fato de que as distancias entre as ilhas nesse tipo de placa não obedecem às distâncias exigidas pelos terminais dos cabeçalhos da FRDM-KL25Z. Essa constatação foi realizada após uma vasta pesquisa de placas disponíveis para

compra na *web*. Por esse motivo, realizou-se o leiaute de uma PCI, para posterior confecção.

4.1.2 Leitura de tensão elétrica em pinos I/O

Uma das maneiras mais simples de alterar a tensão elétrica que é direcionada a um pino I/O (de forma que o MCU consiga identificar a mudança) é através de uma chave. Existem diversos tipos de chaves elétricas no mercado, no entanto uma chave de fácil manuseio e compra é a tecla táctil. A Figura 32 exemplifica a aparência de uma tecla táctil:

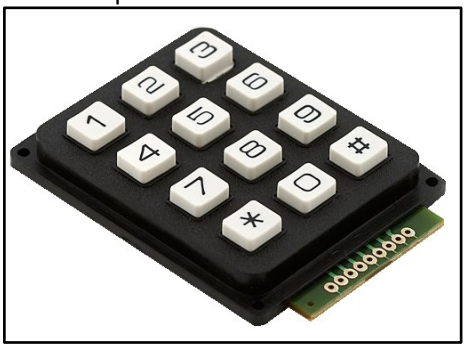
Figura 32 – Exemplo de tecla táctil disponível no mercado

Fonte: DIGI-KEY

Na data 20/11/2016 foi possível encontrar esse componente pelo preço de R$ 0,17 no *website* da PROESI, outra empresa brasileira de bastante renome na venda de componentes pela internet.

Outro tipo de “chave” interessante para uso em sistemas embarcados é o conjunto de chaves dado por um teclado numérico de 12 botões. Esse teclado está exemplificado na Figura 33:

Figura 33 – Exemplo de teclado numérico de 12 botões



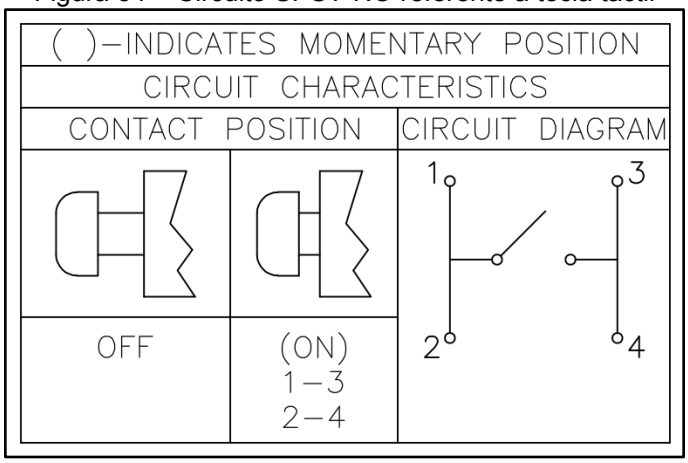
Fonte: ROBOCORE

Na data 20/11/2016 foi possível encontrar esse componente pelo preço de R$ 35,00 no *website* da ROBOCORE.

Por causa de o teclado ser bem mais caro que a tecla táctil, além de ocupar muito mais espaço em placa, optou-se pela utilização da tecla. No intuito de que o usuário tivesse oportunidade de realizar vários tipos de comandos via pinos I/O, foi projetado uma espécie de *joystick* com cinco teclas tácteis, uma tecla para cada direção (cima, baixo, esquerda e direita) e uma central.

A tecla táctil que se escolheu usar corresponde eletricamente a uma chave no estilo SPST-NO (*Single Pole*, *Single Throw* – *Normally-Open*). A Figura 34 ilustra esse circuito:

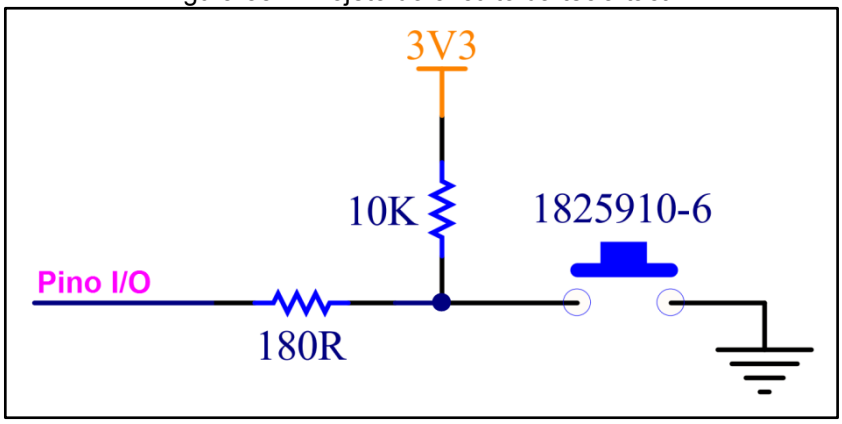
Figura 34 – Circuito SPST-NO referente à tecla táctil



Fonte: TE CONNECTIVITY. *Datasheet* da tecla táctil 1825910-6. Disponível em: <http://www.te.com/commerce/DocumentDelivery/DDEController?Action=srchrtrv&DocNm=1825910&DocType=Customer+Drawing&DocLang=English>. Acesso em: 20 nov. 2016.

Assim sendo, o projeto do circuito da tecla táctil (que se repete cinco vezes) para a interface de usuário está representado pela Figura 35:

Figura 35 – Projeto do circuito da tecla táctil



Fonte: Elaborado pelo autor

Este circuito, como se pode observar, fornece uma tensão de 3,3V ao pino I/O com a tecla solta. Ao se apertar a tecla, é imposto ao pino I/O uma tensão de 0V. O resistor de 180Ω serve para impedir que um curto-circuito ocorra caso a tecla esteja pressionada em um momento de *reset* do MCU, visto que os pinos I/O são levados a 3,3V pelo MCU nesse instante. Ele também impede que aconteça um curto-circuito (com a tecla pressionada) caso o pino I/O tenha sido configurado como pino de saída e a tensão imposta seja de 0V. O valor de 180Ω foi calculado de forma a impedir que a corrente ultrapasse o valor máximo em um pino I/O (25mA, segundo a Tabela 3, no capítulo 2). A Equação (2) calcula a resistência mínima:

$$R_{min} = \frac{V_{máx\_pino} - V_{min\_tecla}}{I_{máx\_pino}} = \frac{3,3V - 0V}{25mA} = 132\Omega \qquad (2)$$

No intuito de providenciar uma “folga” e de usar um resistor com valor no padrão comercial, utilizou-se um resistor no valor de 180Ω ao invés dos 132Ω calculados. A corrente máxima que passará no pino I/O, neste caso, é calculada através da Equação (3):

$$I_{máx\_pino} = \frac{V_{máx\_pino} - V_{min\_tecla}}{R_{utilizado}} = \frac{3,3V - 0V}{180\Omega} \cong 18,33mA \qquad (3)$$

O resistor de 10kΩ é um resistor de *pull-up*, que neste circuito impede que um curto-circuito ocorra entre o terminal de 3,3V e o terminal de terra no momento em que a tecla é pressionada.

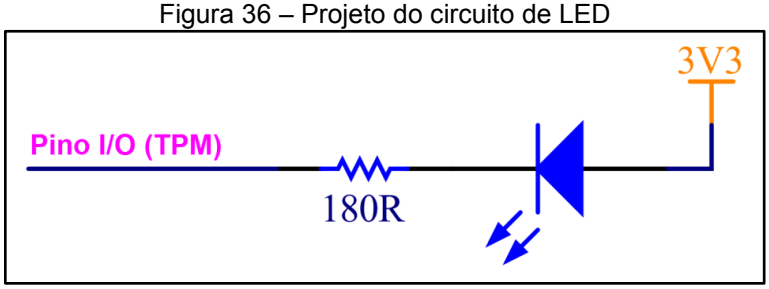
Pode-se observar que o circuito projetado não apresenta um capacitor para a

suavização da transição na curva de tensão formada com o pressionamento do botão. Essa ausência foi proposital, para que os estudantes aprendam a implementar o *debounce* via *software*.

4.1.3 Módulos TPM

Os módulos TPM do MCU da FRDM-KL25Z têm dupla funcionalidade, pois podem operar tanto como *timer*, quanto como PWM. No caso de operar como PWM, que é um dos interesses de aplicação deste trabalho, o MCU pode ser configurado de forma a direcionar o sinal gerado diretamente para certos pinos I/O (ver Quadro 3, no capítulo 2). Assim sendo, para se observar os sinais gerados pelos módulos TPM, utilizaram-se quatro LEDs conectados a pinos que providenciam tais sinais. O circuito projetado para cada LED é exemplificado na Figura 36:

Figura 36 – Projeto do circuito de LED



Fonte: Elaborado pelo autor

Por questões de preço e facilidade de localização no mercado, utilizaram-se LEDs alto brilho na cor vermelha, com *pads* no estilo TH e corpo circular com 5mm de diâmetro. Esse tipo de LED está ilustrado na Figura 37:

Figura 37 – Exemplo de LED utilizado no projeto

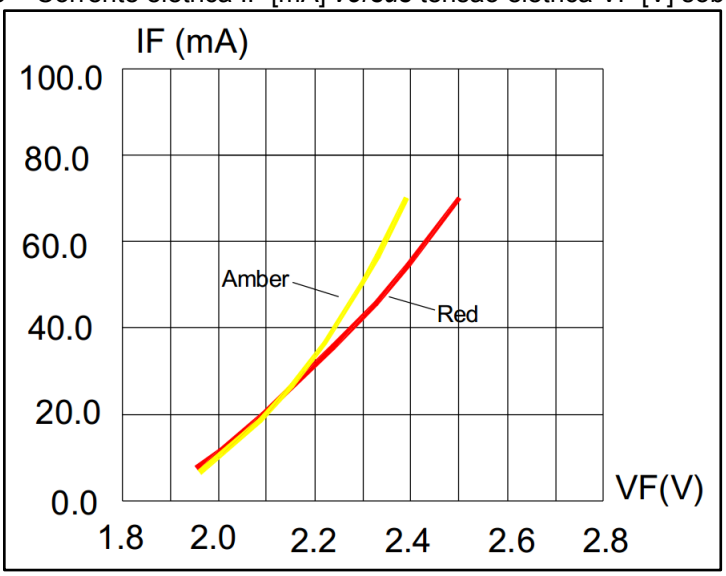
Fonte: DIGI-KEY

Através da Figura 36 é possível observar que quando é fornecida ao pino do MCU uma tensão de 3,3V, não há corrente sobre o LED e, consequentemente, não há iluminação sendo gerada. No entanto, quando é fornecida ao pino uma tensão de 0V, há corrente e o LED “acende”. Essa lógica de tensão é dita inversa, pois a tensão “alta” no pino faz o LED “apagar”. O circuito foi projetado dessa forma, pois, como explicado anteriormente, a tensão nos pinos durante a inicialização do MCU é de 3,3V, ou seja, não há desperdício de energia pelos LEDs. O valor da corrente no LED quando “aceso” é calculado pela Equação (4):

$$I_{max\_LED} = \frac{V_{anodo\_LED} - V_{min\_pino}}{R_{utilizado}} = \frac{3,3V - 0V}{180\Omega} \cong 18,33mA \qquad (4)$$

Da mesma forma que no circuito da tecla táctil, o valor do resistor utilizado foi projetado de tal maneira que a corrente no pino I/O não ultrapassasse o máximo recomendado pela NXP. A Figura 38 exibe o gráfico da corrente *versus* tensão sobre o LED (observar apenas a curva vermelha):

Figura 38 – Corrente elétrica IF [mA] *versus* tensão elétrica VF [V] sobre o LED



Fonte: CREE INC. *Datasheet* do LED C503B-RCN-CW0Z0AA1. Disponível em: <http://www.cree.com/~/media/Files/Cree/LED%20Components%20and%20Modules/HB/Data%20Sheets/C503B%20RAS%20RAN%20AAS%20AAN%20RBS%20RBN%20ABS%20ABN%20RCS%20RCN%20ACS%20ACN%201079.pdf>. Acesso em: 21 nov. 2016.

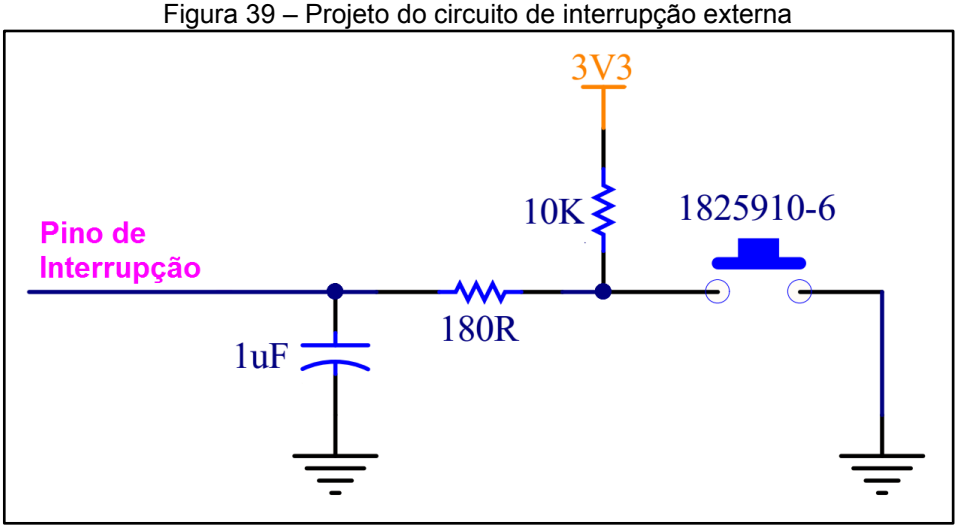
Como se pode observar, a tensão que permanece sobre o LED, no caso de

uma corrente igual a 18,33mA, é de aproximadamente 2,08V.

4.1.4 Interrupção externa

O circuito de interrupção externa é bastante semelhante aos dos botões apresentado na seção 4.1.2. A diferença neste circuito é que o *debounce* já está implementado via *hardware* através de um capacitor, como exibido na Figura 39:

Figura 39 – Projeto do circuito de interrupção externa



Fonte: Elaborado pelo autor

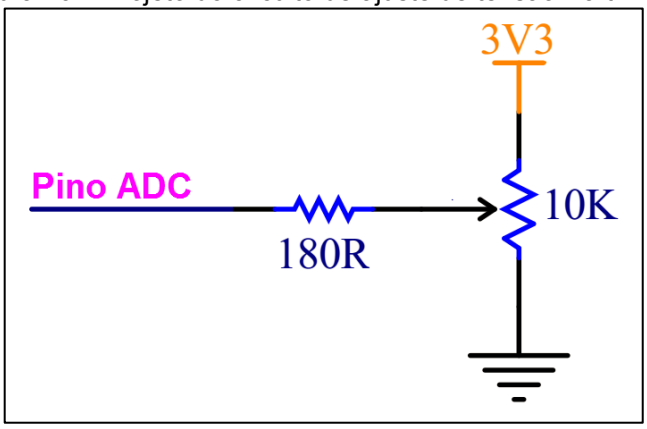
Este circuito é repetido duas vezes no projeto, de forma que os estudantes possam aprender a tratar mais de uma interrupção via *software*. O tempo que o circuito RC leva para estabilizar a tensão no pino após o botão ter sido pressionado é aproximadamente igual a cinco vezes a constante de tempo e está calculado através da Equação (5):

$$Tempo_{RC} = 5.\tau = 5.R.C = 5.180\Omega.1\mu F = 0{,}9ms \quad (5)$$

4.1.5 Conversão de dados via ADC e DAC

Foram projetados dois circuitos para aquisição de dados via ADC, de forma a utilizar dois canais do conversor. No primeiro utilizou-se um trimpot para o ajuste da tensão pelo usuário. A Figura 40 mostra o circuito:

Figura 40 – Projeto do circuito de ajuste de tensão via trimpot



Fonte: Elaborado pelo autor

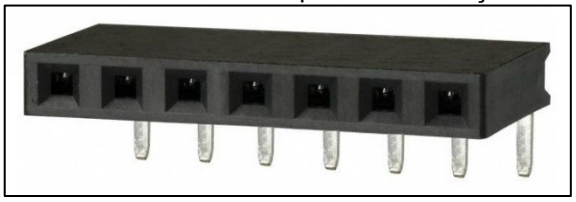
Com o trimpot disposto desta maneira, o usuário pode manualmente excursionar a tensão de entrada no pino ADC entre 0V e 3,3V. O resistor de 180Ω serve apenas para prevenir a ocorrência de um curto-circuito, assim como já mencionado na sessão 4.1.2.

O segundo circuito implementado para aquisição de dados via ADC é simplesmente um curto-circuito entre o pino referente à DAC e o pino de um canal do ADC. Assim, é possível programar uma função de teste simultâneo entre os conversores. Um exemplo prático de aplicação desse circuito seria a geração de uma onda senoidal entre 0V e 3,3V (nível DC de 1,65V) pela DAC e a captação e tratamento desse sinal via ADC.

4.1.6 Comunicação via UART

O projeto do circuito de comunicação via UART é bastante simples, pois envolve apenas a ligação de um conector aos pinos de um dos módulos UART do MCU. O conector utilizado foi do tipo barra de pinos fêmea em 90º, com sete pinos e uma fileira. A Figura 41 exibe esse conector:

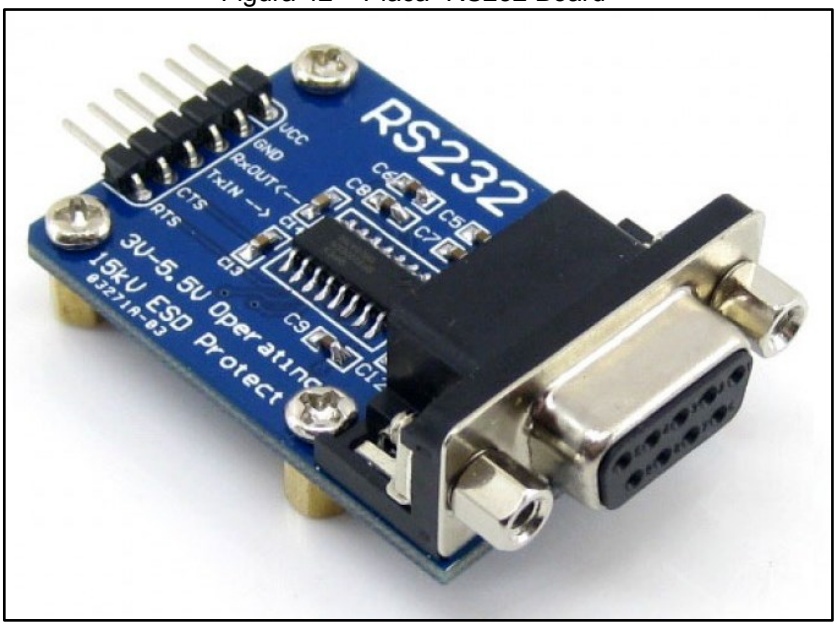
Figura 41 – Conector utilizado para comunicação via UART



Fonte: DIGI-KEY

O circuito foi projetado de forma a ser compatível com o padrão de conexão exigido por placas disponíveis no mercado, as quais contêm dispositivos de comunicação serial UART. Uma dessas placas é a conhecida “RS232 Board”, que proporciona a comunicação entre o microcontrolador e um computador. A Figura 42 a ilustra:

Figura 42 – Placa “RS232 Board”



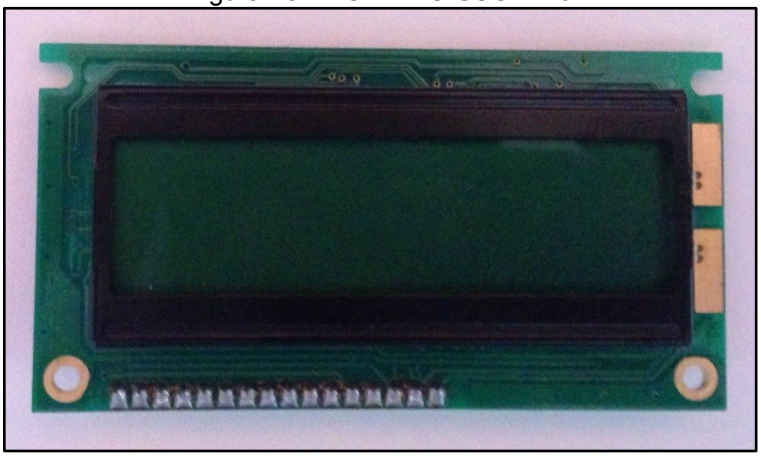
Fonte: WAVESHARE

É possível obter todas as informações sobre essa placa (diagrama esquemático, *datasheets* e *softwares*) através do link: “http://www.waveshare.com/wiki/RS232_Board”. Seu preço unitário pelo *website* da Waveshare no dia 24/11/2016 era de $ 3,99.

4.1.7 Controle de um display LCD 16x2

O *display* sugerido para uso na interface de usuário é o LMC-SSC2E16, uma vez que o Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul já possui certa quantidade desse item. Esse *display* é atualmente utilizado na disciplina de Microprocessadores I. A Figura 43 o ilustra:

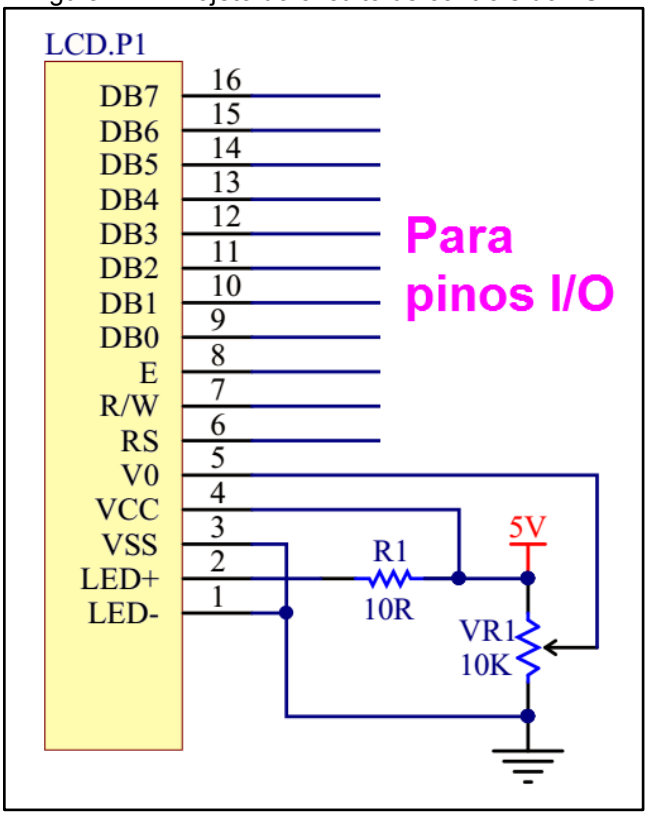
Figura 43 – LCD LMC-SSC2E16



Fonte: Elaborado pelo autor

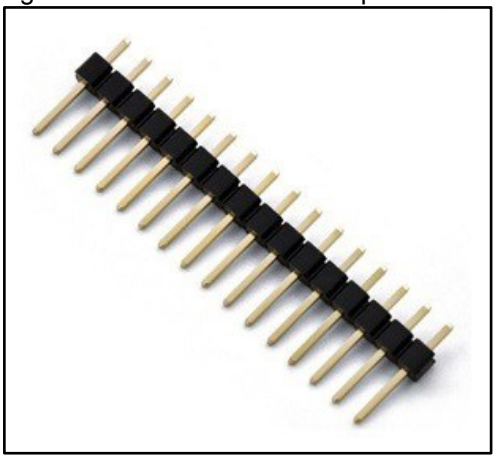
O circuito projetado para operação deste LCD contém um conector do tipo barra de pinos macho em 180º, com dezesseis pinos e uma fileira, além de um trimpot para o ajuste do contraste e um resistor para limitar a corrente no LED de iluminação. Os valores das resistências do trimpot e do resistor foram projetados de forma a atender as necessidades impostas pelo *datasheet* do LCD. Além disso, segundo o seu *datasheet*, o *display* deve ser alimentado com 5V, porém é capaz de interpretar dados de entrada com uma tensão mínima de 2,2V. Assim sendo, não há problema de compatibilidade com a FRDM-KL25Z, pois a tensão de nível lógico alto de saída do MCU é de 3,3V. A Figura 44 exibe o circuito projetado e a Figura 45 o conector utilizado:

Figura 44 – Projeto do circuito de controle do LCD



Fonte: Elaborado pelo autor

Figura 45 – Conector utilizado para o LCD

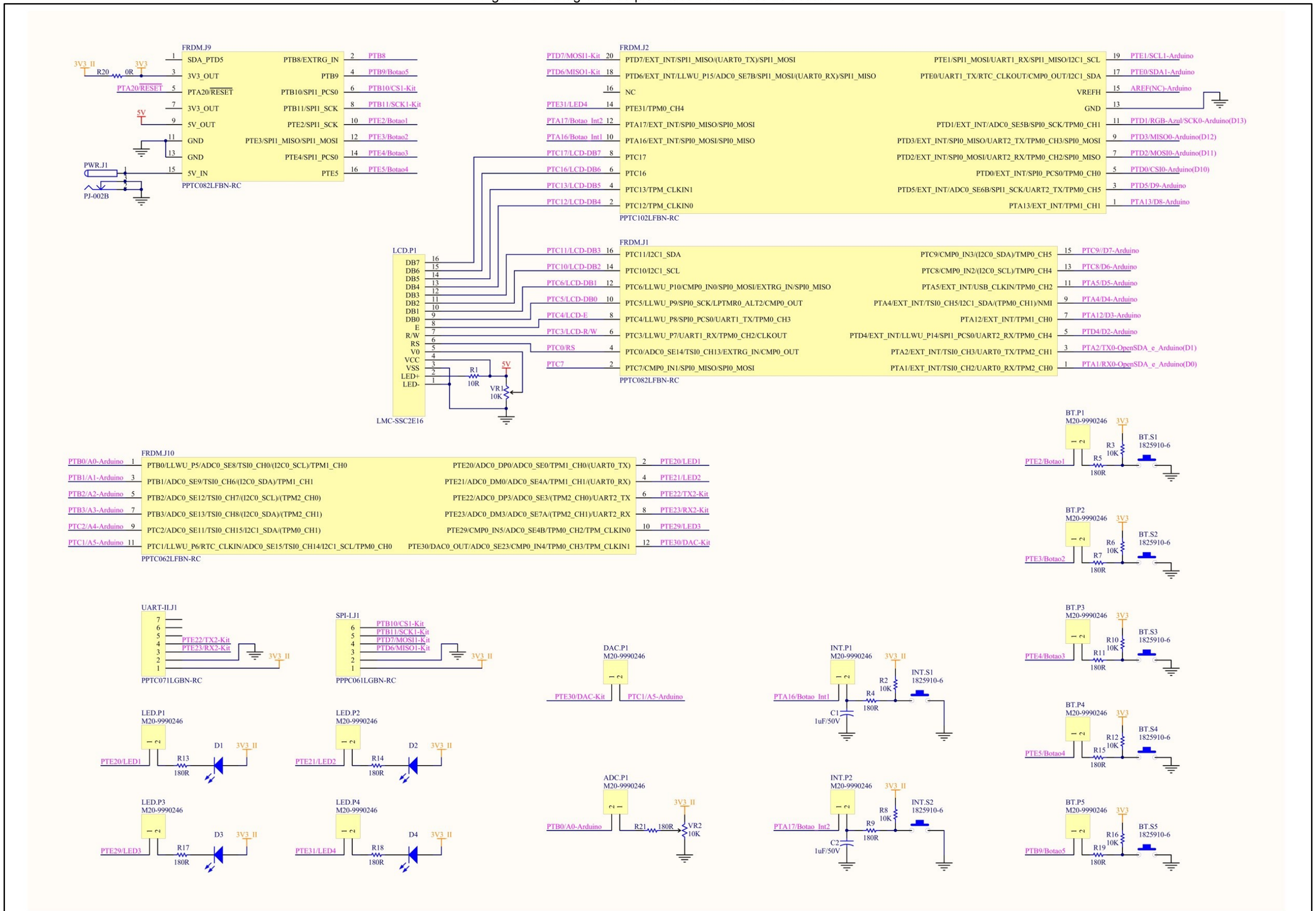


Fonte: DIGI-KEY

4.2 DIAGRAMA ESQUEMÁTICO

A Figura 46 ilustra o diagrama esquemático da interface de usuário em sua totalidade:

Figura 46 – Diagrama esquemático da interface de usuário



Fonte: Elaborado pelo autor

Os cabeçalhos “FRDM.J1”, “FRDM.J2”, “FRDM.J9” e “FRDM.J10” correspondem respectivamente à conexões estabelecidas com os cabeçalhos J1, J2, J9 e J10 da plataforma FRDM-KL25Z. Uma observação importante é que no diagrama da interface de usuário os números dos cabeçalhos estão espelhados em relação aos da FRDM-KL25Z, ou seja, o pino 1 do cabeçalho “FRDM.J9” na interface corresponde ao pino 2 do cabeçalho J9 na FRDM-KL25Z. Isso se deve ao *software* Altium Designer (utilizado para projetar a interface) designar os pinos dos cabeçalhos diferentemente do *software* que foi utilizado pela NXP para projetar a FRDM-KL25Z.

Outra informação relevante é que foram projetados *jumpers* para o desacoplamento opcional dos circuitos dos LEDs, botões, ADC e DAC, visto que o usuário pode não querer utilizar algum desses circuitos, mas apenas o pino que está conectado a ele. Dessa forma, o usuário pode desconectar os circuitos da interface para uso individual das ilhas, conforme a necessidade.

O plugue “PWR.J1” foi adicionado ao projeto em caso de o usuário querer alimentar a interface com uma fonte externa que não seja USB. A tensão aceitável nesse plugue está dentro da faixa de 5V a 9V. Porém, sendo a interface alimentada pelo plugue, o pino 9 do cabeçalho “FRDM.J9” somente poderá fornecer 5V ao LCD (e a outras cargas adicionadas pelo usuário) quando o regulador de tensão citado na Figura 17 do capítulo 2 estiver soldado na FRDM-KL25Z. Caso a alimentação seja provida via um dos conectores USB da plataforma, todos os componentes operam normalmente.

Além do conector UART, citado na sessão anterior, foi adicionado um conector do mesmo estilo para SPI, referenciado pelo cabeçalho “SPI-I.J1”. Assim, além da possibilidade de se conectar uma placa à interface de usuário que comunica via UART, tem-se como também conectar uma placa que comunica via SPI.

O resistor R20, de 0Ω, está presente no projeto, pois foi de interesse que no leiaute houvesse um meio de passagem extra para a trilha de 3,3V. Dessa forma, esse resistor funciona como um *jumper* permanente.

A escolha dos pinos para cada sinal empregado foi realizada da seguinte forma e sequência:

a) não utilizar os pinos mais externos dos cabeçalhos da FRDM-KL25Z, para que se possam conectar Arduino Shields sem interferir nos circuitos da

interface;

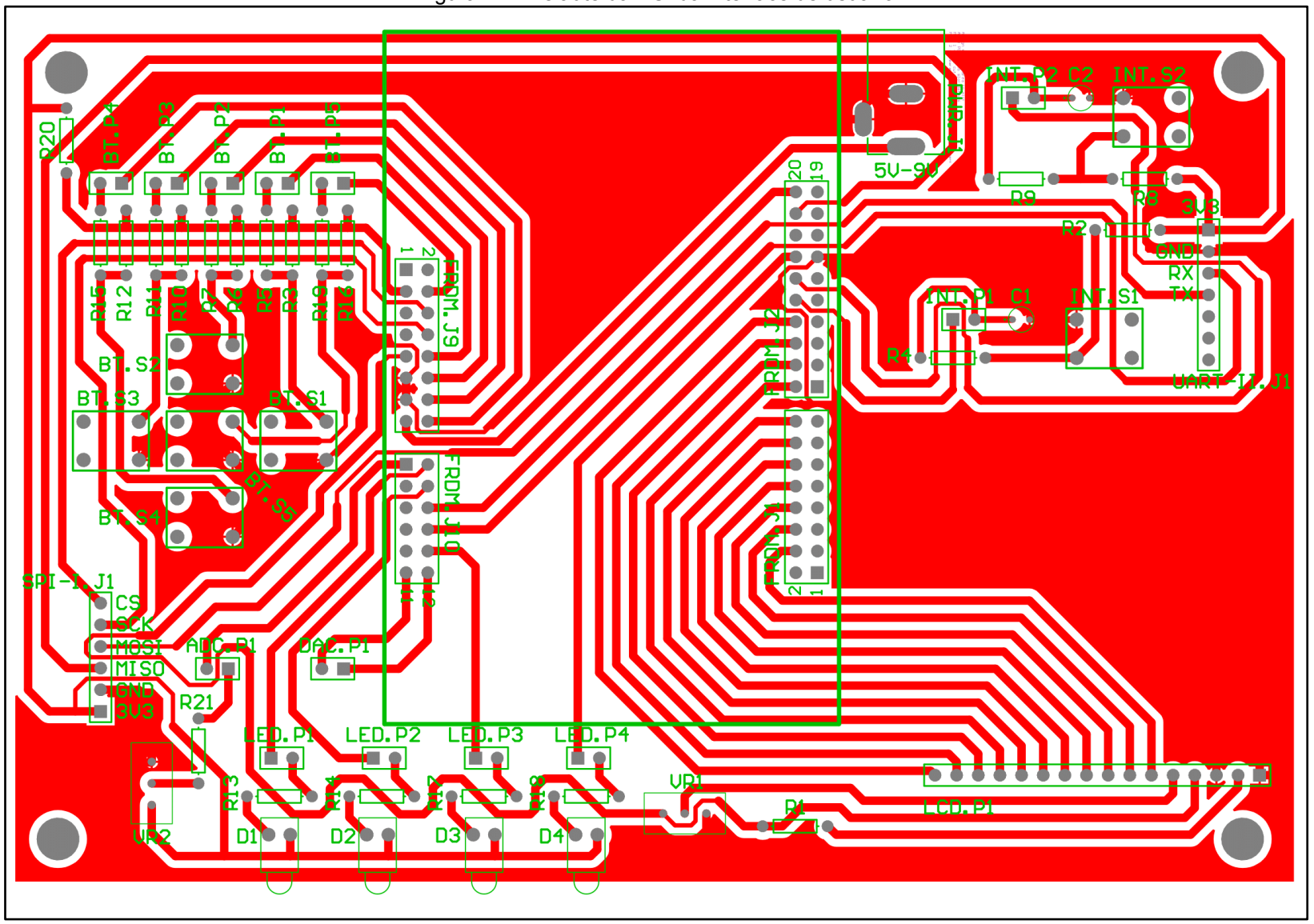
b) utilizar os pinos I/O que não desempenham nenhuma das funções de TPM, interrupção, UART, SPI, ADC e DAC exclusivamente para os circuitos de botões do *joystick* e LCD;
c) utilizar para os circuitos de LEDs canais distintos dos módulos TPM e não utilizar o módulo TPM2, uma vez que já está sendo usado pelo LED RGB;
d) circuitos dos botões de interrupção devem estar conectados a pinos que possibilitem interrupção externa (apenas portas A e D);
e) não utilizar o módulo UART0, uma vez que já está sendo usado pelo OpenSDA;
f) deixar pinos de comunicação SPI por último.

Um último comentário sobre o diagrama esquemático é que se faltou curto-circuitar dois terminais de terra no leiaute da placa. Entretanto, não há problema nisso, pois ao se conectar a FRDM-KL25Z na interface de usuário, esses pontos são conectados, uma vez que a plataforma realiza a conexão através de sua placa. Essa dificuldade no leiaute está relacionada à necessidade da placa de interface ter de ser em face simples. Os pinos de terra que são conectados através da plataforma FRDM-KL25Z são o 13 do cabeçalho “FRDM.J9” e o 13 do cabeçalho “FRDM.J2”. Verificou-se, na *homepage* do Arduino, que a plataforma Arduino Uno R3 e os Shields Arduino R3 também realizam essa conexão de terra em suas placas. Portanto, independentemente da plataforma que for conectada à interface de usuário todos os pontos de terra estarão na mesma referência.

4.3 LEIAUTE DA PCI

A Figura 47 apresenta o leiaute da placa:

Figura 47 – Leiaute da PCI da interface de usuário



Fonte: Elaborado pelo autor

A PCI projetada tem dimensões iguais a 15cm de comprimento por 10cm de largura, uma vez que este é um tamanho de placa de fácil localização no mercado. A maior parte das trilhas tem 1mm de largura, exceto onde a passagem da trilha ocorre entre terminais próximos e desconexos. Na pior das hipóteses foi utilizado 0,5mm de largura. Também, em todos os lugares possíveis, foi imposto um espaçamento mínimo entre trilhas de 0,9mm, para reduzir o efeito de *crosstalk*. Essas características de leiaute também foram pensadas de modo a facilitar a confecção da placa, principalmente em relação à parte de corrosão do cobre com percloreto de ferro, no qual pode ocorrer desmantelamento de trilhas pela excessiva corrosão.

Todas as medidas da FRDM-KL25Z foram tomadas com uma régua, de maneira que os componentes da placa não colidissem e a conexão entre placas ocorresse sem problemas. Além disso, os componentes foram posicionados de forma que se reduzisse ao mínimo possível o comprimento das trilhas. O conector para LCD foi posicionado no canto inferior direito, para que assim as mensagens aparecessem de frente para o usuário.

Quatro furos foram adicionados nas extremidades da placa. Dessa forma é possível adicionar pés de apoio na PCI para que os terminais dos componentes não encostem na mesa.

Os arquivos do projeto da PCI estão disponíveis para *download*, através do *link* “https://drive.google.com/open?id=0B5VDkyH_qYOubjZiZ3pkaS1rQTg”.

4.4 LISTA DE COMPONENTES E CUSTOS

O Quadro 4 lista todos os componentes para a produção da placa:

Quadro 4 – Lista de componentes da interface de usuário

Referência	Descrição	*Part Number*	Quantidade
ADC.P1, BT.P1, BT.P2, BT.P3, BT.P4, BT.P5, DAC.P1, INT.P1, INT.P2, LED.P1, LED.P2, LED.P3, LED.P4	Barra de pinos macho 180º 2x1	M20-9990246	13
BT.S1, BT.S2, BT.S3, BT.S4, BT.S5, INT.S1, INT.S2	Tecla Táctil 50mA 24VDC SPST-NO-TH	1825910-6	7
C1, C2	Capacitor Eletrolítico 1uF/50V	ESK105M050AC3AA	2
D1, D2, D3, D4	LED Alto Brilho Vermelho	C503B-RCN-CW0Z0AA1	4
FRDM.J1, FRDM.J9	Barra de pinos fêmea 180º 8x2	PPTC082LFBN-RC	2
FRDM.J2	Barra de pinos fêmea 180º 10x2	PPTC102LFBN-RC	1
FRDM.J10	Barra de pinos fêmea 180º 6x2	PPTC062LFBN-RC	1
LCD.P1	Barra de pinos macho 180º 16x1	PREC016SAAN-RC	1
PWR.J1	Jack DC-005 J4/P4 DC Fêmea 2,5mm	PJ-002B	1
R1	Resistor 10R	CF14JT10R0	1
R2, R3, R6, R8, R10, R12, R16	Resistor 10K	CF14JT10K0	7
R4, R5, R7, R9, R11, R13, R14, R15, R17, R18, R19, R21	Resistor 180R	CF14JT180R	12
R20	Resistor 0R	ZR0207C	1
SPI-I.J1	Barra de pinos fêmea 90º 6x1	PPPC061LGBN-RC	1
UART-II.J1	Barra de pinos fêmea 90º 7x1	PPTC071LGBN-RC	1
VR1, VR2	Trimpot 90º Código 3296, 10K	3296W-1-103LF	2
Display de LCD 16x2	Display de LCD 16x2	LMC-SSC2E16	1
Jumpers	Jumper	969102-0000-DA	13
FRDM-KL25Z: Regulador para J22 (opcional)	Regulador de Tensão 7805 no encapsulamento TO-220	L7805CV	1
FRDM-KL25Z: J1, J9	Barra de pinos fêmea 180º 8x2, pino longo	SSQ-108-03-T-D	2
FRDM-KL25Z: J2	Barra de pinos fêmea 180º 10 x2, pino longo	SSQ-110-03-T-D	1
FRDM-KL25Z: J10	Barra de pinos fêmea 180º 6x2, pino longo	SSQ-106-03-T-D	1

Fonte: Elaborado pelo autor

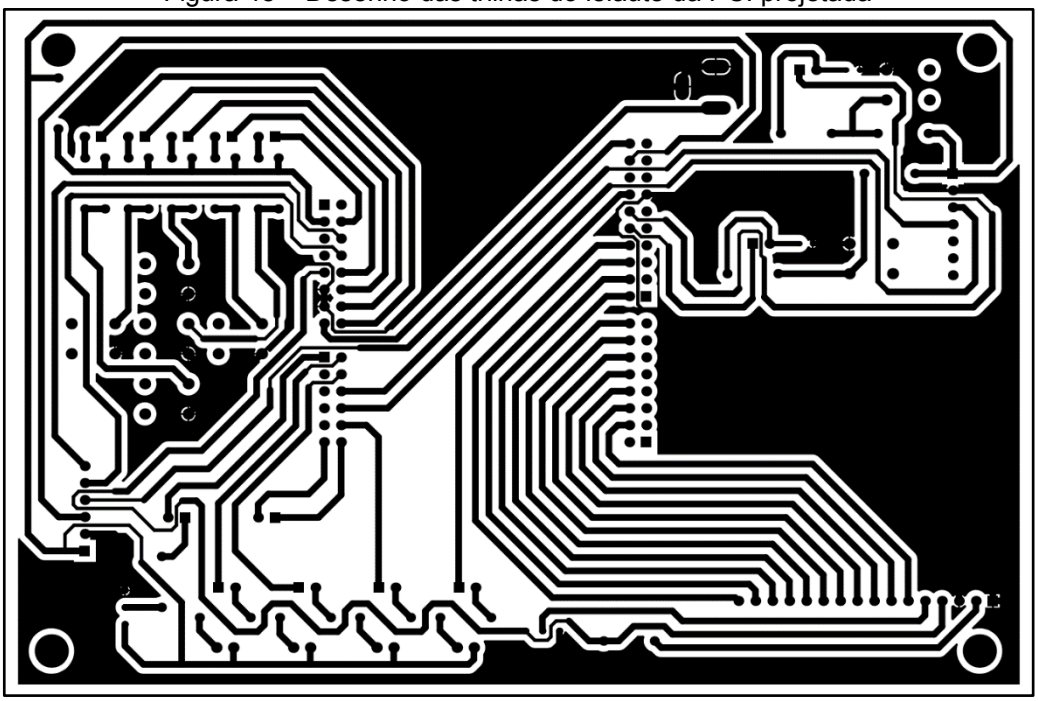
Os componentes foram orçados através de dois meios: lojas brasileiras virtuais (PROESI e ROBOCORE) e pesquisa de preço em lojas da Rua Alberto Bins, em Porto Alegre. O custo de produção de uma única placa através de compras pela *web* é de aproximadamente R$ 145,00. Esse valor unitário tende a baixar com o aumento do número produzido de placas, visto que o custo estimado do frete somado pelas duas lojas virtuais fica em torno de R$ 80,00 na remessa. Assim, o custo unitário via *web* na produção de oito placas fica em torno de R$ 75,00. Além disso, o custo para a produção de uma única placa através de compras em lojas da Rua Alberto Bins é de aproximadamente R$ 140,00.

O arquivo da lista de componentes completa, com informações de custos e locais de venda, está disponível para *download* através *link* “https://drive.google.com/open?id=0B5VDkyH_qYOueWRmUHFQeThLT2M”.

4.5 PRODUÇÃO DA PCI

Após o leiaute da PCI no computador, imprimiu-se, com uma impressora a laser e papel couchê brilhoso, apenas as trilhas do leiaute em tinta preta, como exemplificado na Figura 48:

Figura 48 – Desenho das trilhas do leiaute da PCI projetada

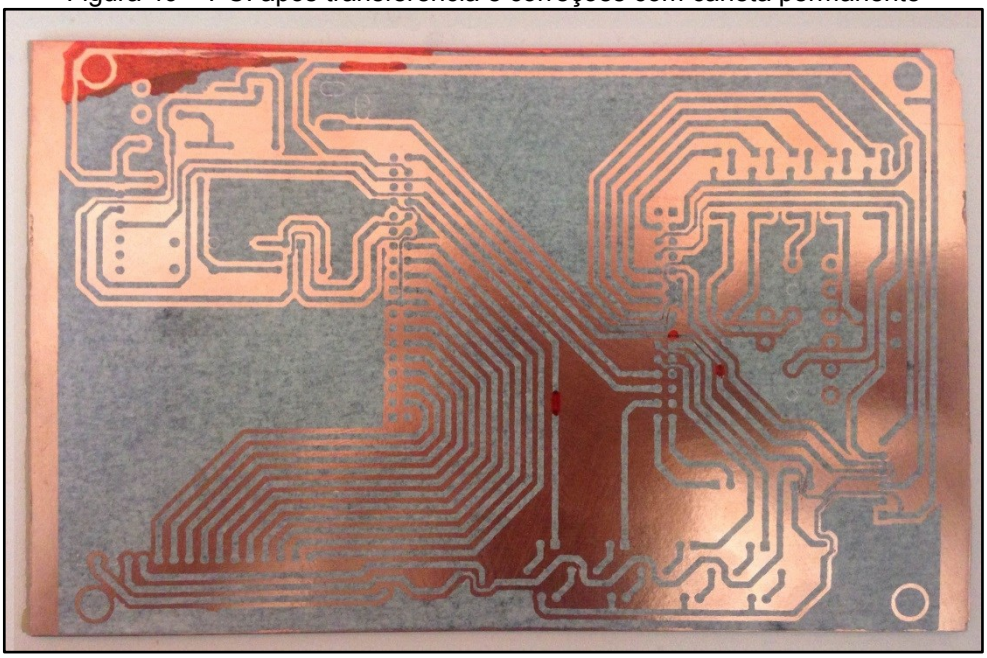


Fonte: Elaborado pelo autor

Em seguida, compraram-se os componentes e alguns utensílios para a confecção da placa (uma placa de fenolite face simples, um frasco de percloreto de ferro, um frasco de álcool isopropílico e uma lixa). Todos os itens foram adquiridos em lojas da Rua Alberto Bins, em Porto Alegre.

Após a limpeza da placa, com lã de aço e álcool, realizou-se a transferência das trilhas impressas para a placa através de um ferro de passar roupa. Para tal, encostou-se a impressão das trilhas na parte de cobre da placa e usou-se uma flanela por cima para passar o ferro. Na sequência, quando a placa já estava bastante aquecida e as trilhas pareciam ter grudado, aplicou-se água fria de forma a fazer com que o papel fixasse na placa. Alguns esfregões com a mão foram realizados, e a transferência foi terminada. Nos pontos da placa em que a transferência não fora bem-sucedida, corrigiram-se as inconsistências com uma caneta permanente. A Figura 49 mostra a aparência da placa naquele instante:

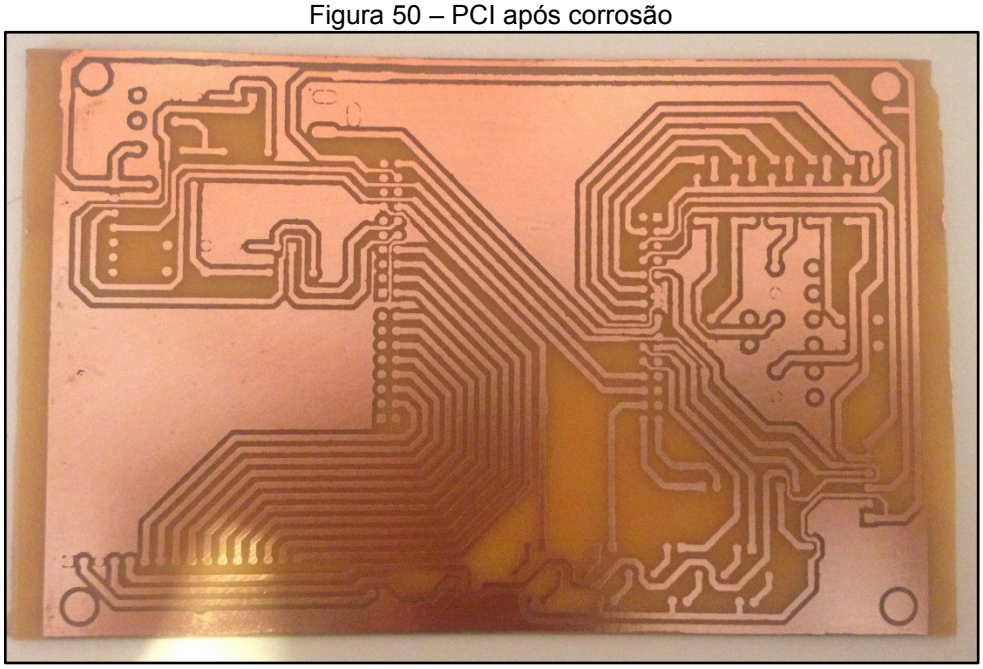
Figura 49 – PCI após transferência e correções com caneta permanente



Fonte: Elaborado pelo autor

Em seguida, usou-se o percloreto para realizar a corrosão do cobre apenas nas regiões onde não havia tinta. Depois de aguardar aproximadamente vinte minutos, retirou-se a placa do percloreto, limpou-se bem com água e aplicou-se o álcool para remover a tinta. A placa ficou como ilustrado na Figura 50:

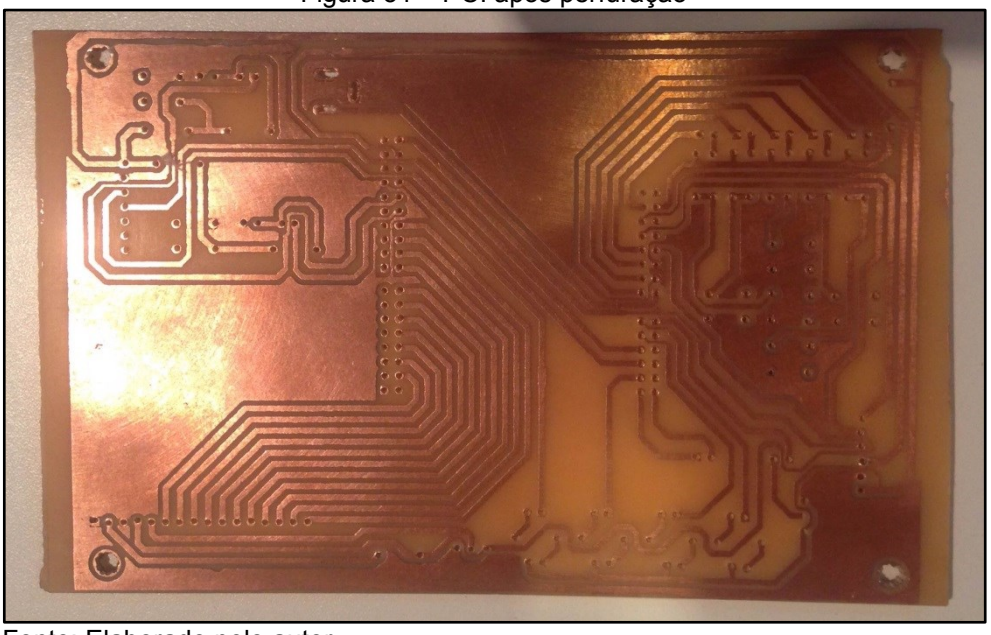
Figura 50 – PCI após corrosão



Fonte: Elaborado pelo autor

Após esse processo, perfurou-se a PCI, com um perfurador especial de placas, em todas as posições dos terminais. O resultado é verificado na Figura 51:

Figura 51 – PCI após perfuração

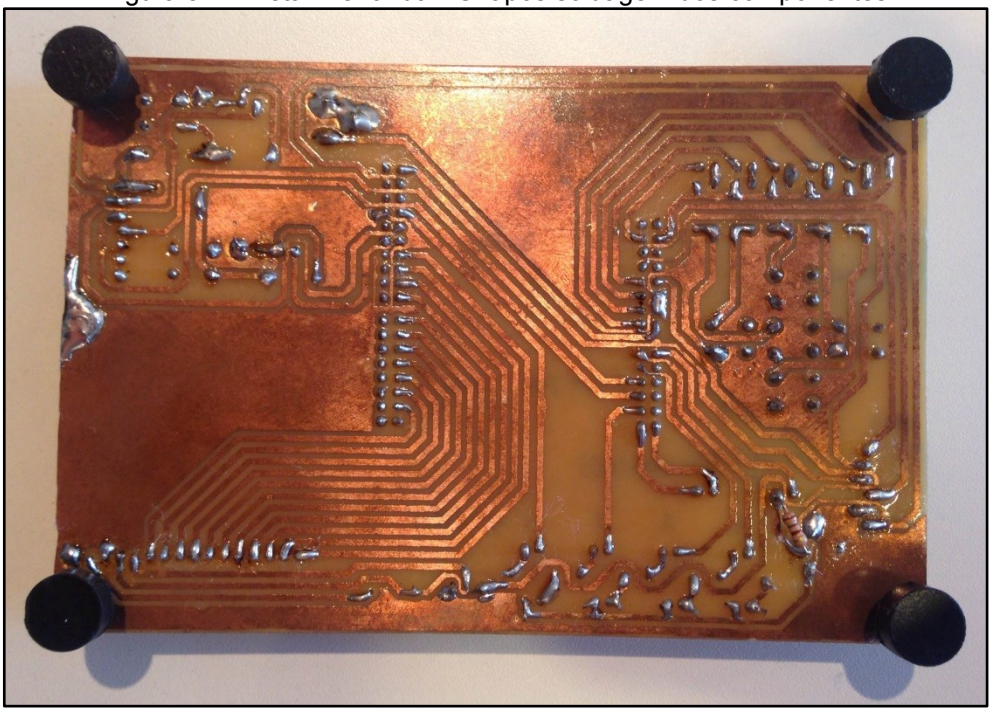


Fonte: Elaborado pelo autor

O próximo passo foi recortar as partes sobressalentes da PCI (parte além das trilhas) e lixar as bordas. Em seguida, soldaram-se os componentes, inseriram-se os pés de apoio e passou-se o verniz para impedir a corrosão das trilhas. A Figura 52

mostra a vista inferior da placa nessa etapa:

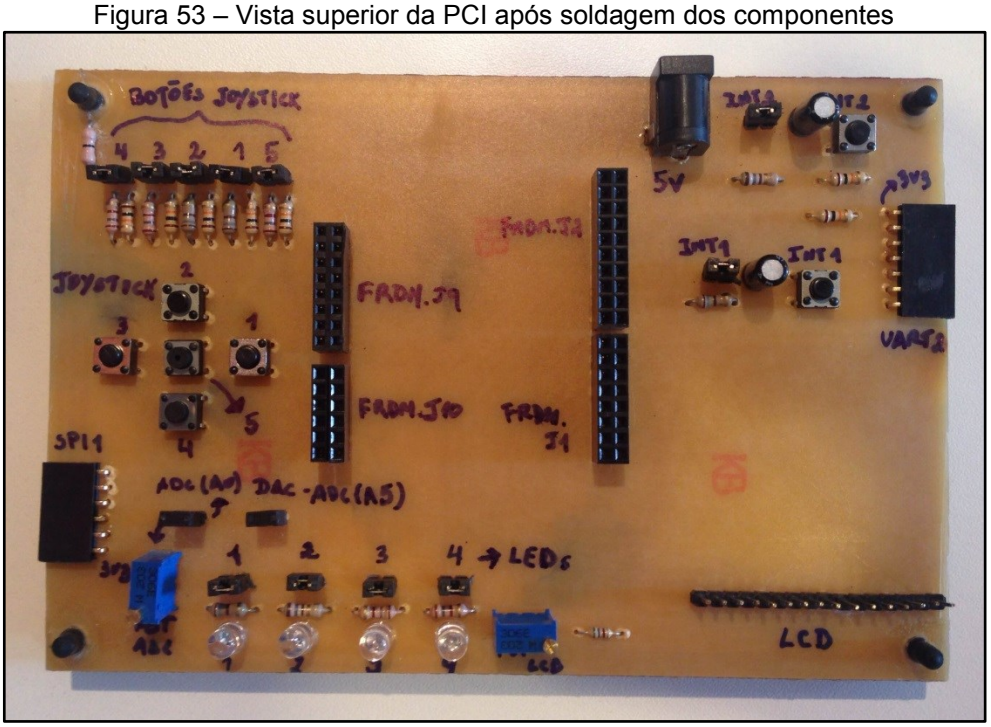
Figura 52 – Vista inferior da PCI após soldagem dos componentes



Fonte: Elaborado pelo autor

O resistor soldado na parte inferior, que pode ser observado no canto inferior direito da Figura 52, foi um erro cometido em um leiaute de versão anterior, em que se havia esquecido de projetar o resistor R21. Para não haver incoerências com o projeto correto (o correto é o descrito neste trabalho), raspou-se com uma faca a trilha que anteriormente correspondia ao resistor e se adicionou o resistor. A vista superior da PCI (lado dos componentes) está ilustrada na Figura 53:

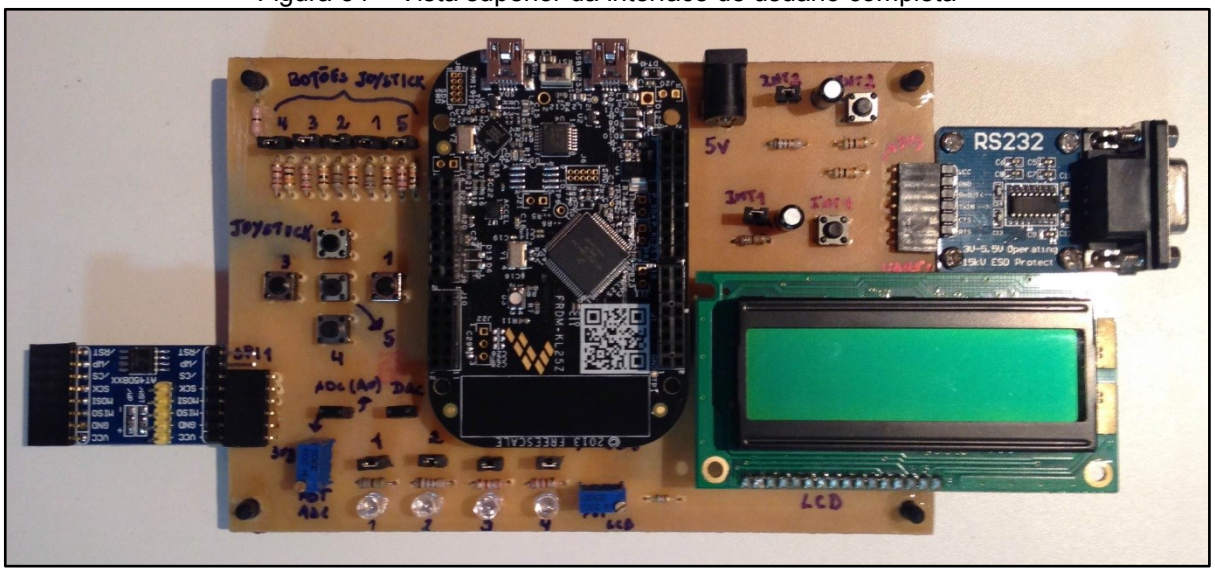
Figura 53 – Vista superior da PCI após soldagem dos componentes



Fonte: Elaborado pelo autor

Por final, conectou-se à interface de usuário a plataforma FRDM-KL25Z, o LCD, uma placa de comunicação UART e uma placa de comunicação SPI. A Figura 54 exibe a vista superior:

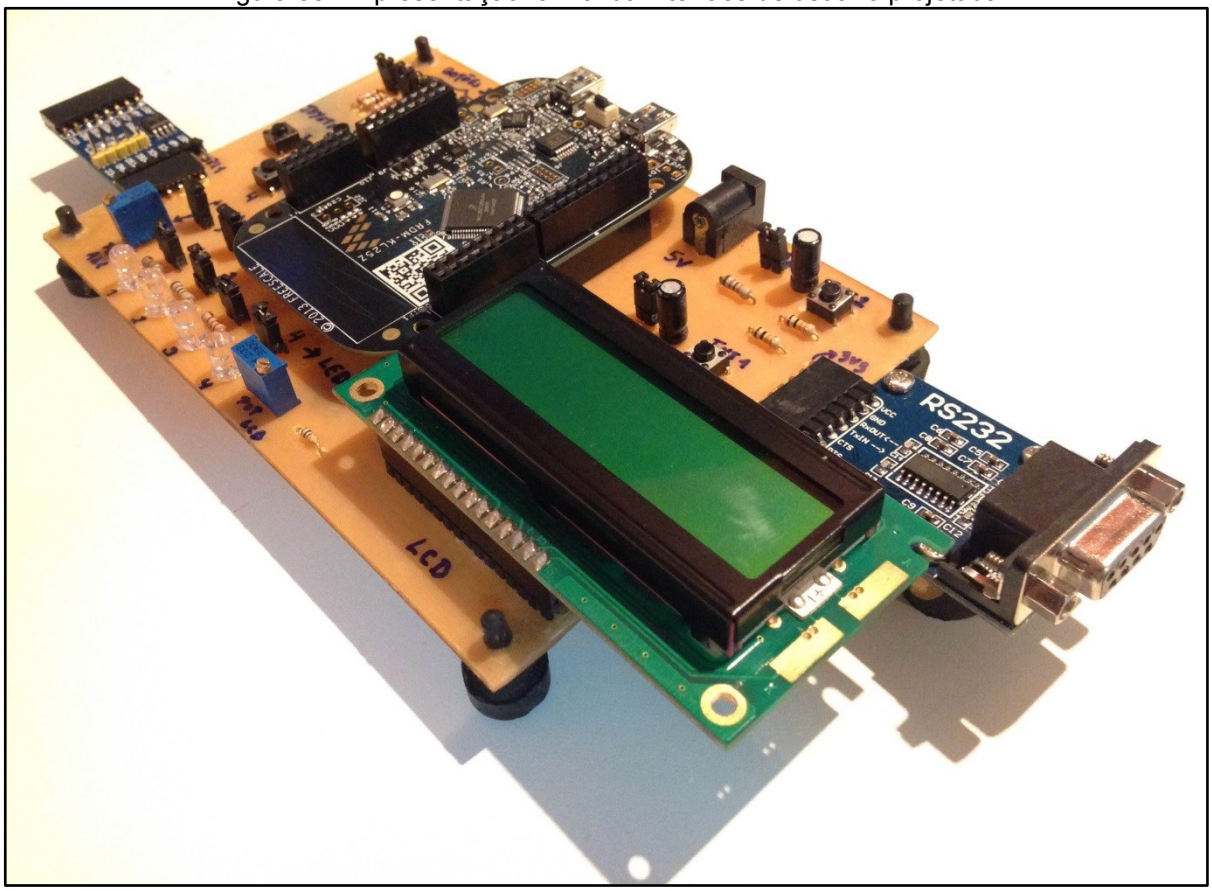
Figura 54 – Vista superior da interface de usuário completa



Fonte: Elaborado pelo autor

A Figura 55 ilustra a interface de usuário em sua forma de apresentação:

Figura 55 – Apresentação formal da interface de usuário projetada



Fonte: Elaborado pelo autor

5 DESENVOLVIMENTO DO SOFTWARE

Através do diagrama esquemático da IU (Figura 46 no capítulo 4) foi possível criar a Tabela 8:

Tabela 8 – Relação entre componentes da IU, nomenclaturas e funções

Componente da IU	Pino do MCU	Função utilizada	Nome do terminal na FRDM-KL25Z
LED 1	PTE20	TPM1_CH0	J10_1
LED 2	PTE21	TPM1_CH1	J10_3
LED 3	PTE29	TPM0_CH2	J10_9
LED 4	PTE31	TPM0_CH4	J2_13
Botão 1 Joystick	PTE2	PTE2	J9_9
Botão 2 Joystick	PTE3	PTE3	J9_11
Botão 3 Joystick	PTE4	PTE4	J9_13
Botão 4 Joystick	PTE5	PTE5	J9_15
Botão 5 Joystick	PTB9	PTB9	J9_3
Botão 1 Interrupção	PTA16	EXT_IRQ-PortA	J2_9
Botão 2 Interrupção	PTA17	EXT_IRQ-PortA	J2_11
LCD RS	PTC0	PTC0	J1_3
LCD R/W	PTC3	PTC3	J1_5
LCD E	PTC4	PTC4	J1_7
LCD DB0	PTC5	PTC5	J1_9
LCD DB1	PTC6	PTC6	J1_11
LCD DB2	PTC10	PTC10	J1_13
LCD DB3	PTC11	PTC11	J1_15
LCD DB4	PTC12	PTC12	J2_1
LCD DB5	PTC13	PTC13	J2_3
LCD DB6	PTC16	PTC16	J2_5
LCD DB7	PTC17	PTC17	J2_7
ADC Trimpot	PTB0	ADC0_SE8	J10_2
ADC DAC	PTC1	ADC0_SE15	J10_12
DAC	PTE30	DAC0_OUT	J10_11
UART RX	PTE23	UART2_RX	J10_7
UART TX	PTE22	UART2_TX	J10_5
SPI MISO	PTD6	SPI1_MISO	J2_17
SPI MOSI	PTD7	SPI1_MOSI	J2_19
SPI SCK	PTB11	SPI1_SCK	J9_7
SPI CS	PTB10	SPI1_CS1	J9_5

Nas explicações que seguem sobre como o *software* da IU foi desenvolvido, sempre será levado em consideração as relações da Tabela 8, uma vez que o projeto exige este conhecimento.

5.1 CRIAÇÃO DE UM NOVO PROJETO NO KDS

Para facilitar a configuração e inicialização dos periféricos do MCU, optou-se por utilizar o Processor Expert no desenvolvimento deste projeto, explanado no item 5 do Apêndice A. Com o KDS aberto e todos os projetos que estavam visíveis no Project Explorer fechados, criou-se um projeto do tipo Processor Expert, através da aba “File”, opção “New” e opção “Processor Expert Project”. O projeto foi nomeado como “Kit_v1” e se utilizou o local *workspace* padrão. O assistente de novo projeto do KDS pede para que seja informada qual a plataforma alvo do programa. Em “Boards” e “Kinetis” foi escolhida a plataforma FRDM-KL25Z. Após clicar em “Next” no assistente, dentro da nova janela, clicou-se na caixa “Processor Expert” para possibilitar o uso das ferramentas gráficas relativas aos periféricos do MCU. Mais uma vez clicou-se em “Next” e depois em “Finish”. O KDS e o Processor Expert nesse momento trabalham juntos. Eles criam um projeto compatível com o KSDK e que tenha as mínimas configurações e inicializações necessárias para que o MCU opere adequadamente.

Na *view* Project Explorer, dentro do projeto “Kit_v1”, é possível observar algumas pastas e arquivos já criados. A pasta “Generated_Code” contém arquivos que sempre serão atualizados pelo Processor Expert toda vez que um novo periférico for adicionado ao projeto. A pasta “Sources” contém os arquivos “main.c” (já explicado no capítulo 3) e o arquivo “Events.c”. Este último é muito importante também, pois é dentro dele que as funções de interrupção são declaradas. Se o usuário deseja criar seus próprios arquivos, recomenda-se que sejam colocados dentro da pasta “Sources”.

Como o projeto criado usa os recursos do Processor Expert, é natural que a *view* “Components”, logo abaixo da *view* Project Explorer, e as *views* “Component Inspector” e “Components Library” (*views* centrais) estejam abertas e mostrando alguma informação. Caso essas *views* não estejam visíveis, pode-se abri-las ao se clicar sobre a aba “Processor Expert” e selecionando a opção “Show Views”. Dentro da *view* Components, já existem quatro componentes pré-definidos pelo Processor Expert. São eles:

a) um componente “fsl_os_abstraction”, nomeado como “osa1”, que é responsável pela abstração e configuração de operações de baixo nível;

b) um componente “MKL25Z128VLK4”, nomeado como “Cpu”, que é responsável por identificar o microcontrolador alvo do programa;
c) um componente “fsl_clock_manager”, nomeado como “clockMan1”, que faz as configurações de *clock* do MCU;
d) um componente “PinSettings”, nomeado como “pin_init”, que já faz a inicialização de vários pinos do MCU com determinadas funções;
e) um componente “fsl_interrupt_manager”, nomeado como “intMan1”, que gerencia as interrupções do MCU e pode ser utilizado por outros componentes criados pelo usuário.

A princípio, estes componentes que foram criados automaticamente pelo Processor Expert, não serão manipulados pelo usuário, visto que não há necessidade disso. O componente “pin_init” é reconfigurado automaticamente toda vez que o usuário utilizar um pino I/O através da criação de um novo componente. É tudo muito simples dentro do Processor Expert. Os componentes que forem adicionados pelo usuário, normalmente representam algum periférico do MCU.

A *view* Component Inspector é simplesmente o lugar onde é possível observar e modificar as propriedades do componente selecionado na *view* Components. É interessante selecionar dentro da *view* Component Inspector a opção “Advanced” e desmarcar a opção “Tabs view” (clicar na setinha do canto superior direito), pois assim será possível ter uma visão global das opções do componente selecionado.

A *view* Components Library é o lugar para se procurar por um novo componente que se deseja criar. Por exemplo, se na barra *search* for pesquisado o nome “tpm”, aparecerão na tela todos os componentes disponíveis associados ao periférico TPM do MCU. Todos os casos de componentes utilizados no projeto “Kit_v1” iniciam com o prefixo “fsl” e não tem o sufixo “hal”, pois são os componentes de periféricos que já possuem implementados todos os possíveis *drivers* do KSDK. Para adicionar qualquer componente, basta dar duplo clique sobre o componente pesquisado na *view* Components Library.

5.2 CRIAÇÃO DOS COMPONENTES PARA OS PERIFÉRICOS

Antes mesmo de começar a se escrever qualquer linha de código, é

interessante que já se deixem configurados e inicializados todos os periféricos que devem ser utilizados pela IU. Isso é bastante simples de ser realizado pelo Processor Expert, uma vez que somente se necessita adicionar componentes específicos e editá-los na *view* Component Inspector. Dessa forma, após a adição dos componentes só restará escrever nos arquivos “main.c” e “Events.c” o comportamento que se deseja para os periféricos. Nesse processo, é importante recorrer à Tabela 8, pois ali estão especificados os pinos de cada funcionalidade dos periféricos. O módulo SPI é o único que não será utilizado.

O conteúdo de produção desta sessão está toda descrita no Apêndice B, por ser bastante extensa e conter muitas ilustrações.

5.3 ESCRITA DOS CÓDIGOS

Os códigos foram desenvolvidos, em sua quase totalidade, arquivos “main.c”, “Events.c” e “LCD.c”, respectivamente Apêndices C, D e E, que implementam as seguintes funcionalidades:

a) inicialização e manipulação do LCD;
b) funções que aumentam e diminuem a frequência dos módulos TPM (que controlam os LEDs), através de ações dadas pelos botões de interrupção;
c) *debounce* dos botões *joystick* e acionamento/desligamento (pelo canal TPM) do LED correspondente à mesma numeração do botão *joystick* pressionado (botão 5 corresponde a todos os LEDs);
d) geração de uma senoide pelo pino do módulo DAC;
e) leituras recorrentes de tensão pelos canais 8 (trimpot) e 15 (DAC) do módulo ADC e atualização no LCD com estes valores;
f) recebimento de dados via UART e exibida no LCD.

Para um bom entendimento dos códigos, foram realizados comentários dentro dos arquivos, de forma explicar as operações programadas. Todos os arquivos de criação exclusiva do autor, como o “LCD.c”, estão dentro da pasta “Sources”.

É importante ressaltar que também foi realizada uma alteração no arquivo "fsl_tpm_driver.c". Comentou-se a linha "TPM_HAL_SetClockMode(tpmBase, kTpmClockSourceNoneClk);" dentro da funcao "TPM_DRV_PwmStop" para que os

LEDs (canais TPM) pudessem ser desligados individualmente.

Além disso, a função “Delay”, que se utiliza diversas vezes no programa, foi criada e declarada dentro do arquivo “funções_gerais.c” (criado dentro da pasta “Sources”) da seguinte forma: “void Delay(__IO uint32_t delayCntr) {while(delayCntr--);}”.

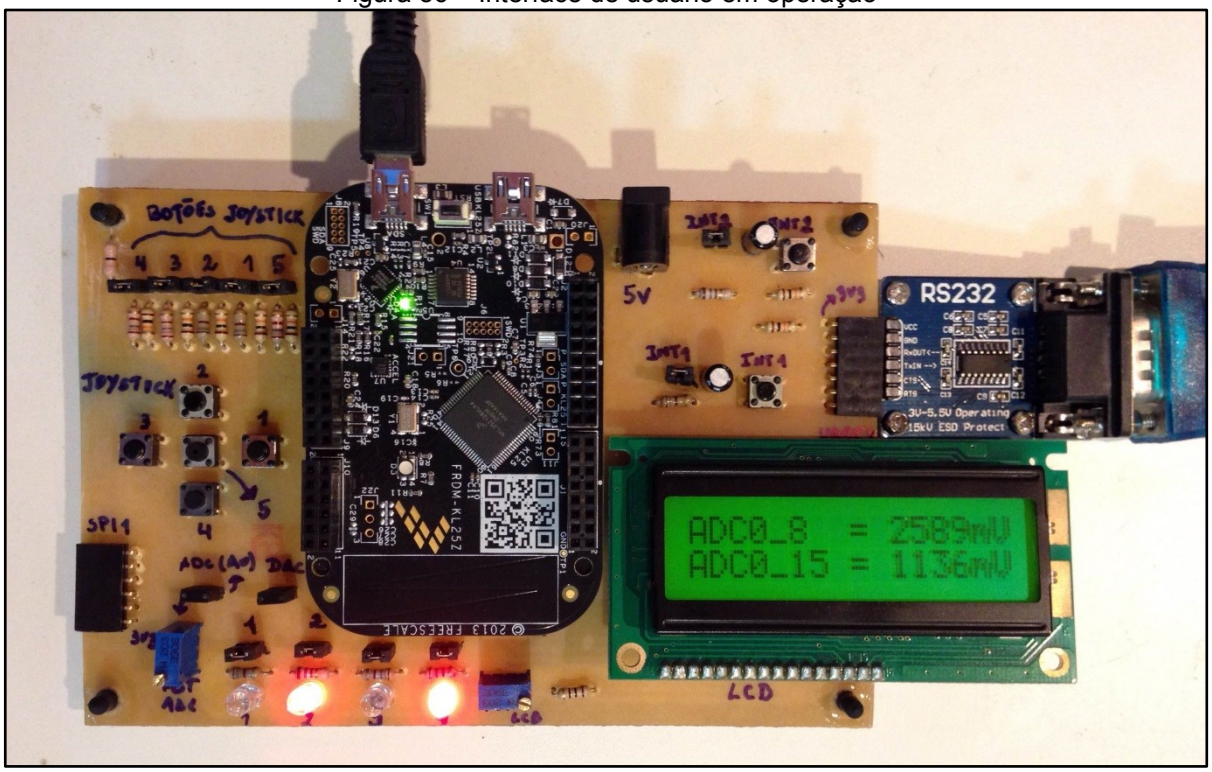
Para se conseguir reproduzir de forma exata os códigos deste trabalho, é importante lembrar que se devem implementar os protótipos das funções criadas através de seus correspondentes arquivos *header*.

Os arquivos de projeto, no KDS, do software desenvolvido estão disponíveis para *download* através do *link* “https://drive.google.com/open?id=0B5VDkyH_qYOuSS12R0ItN28tcVU”.

6 RESULTADOS E DISCUSSÕES

Após a gravação do programa na interface de usuário, observou-se que nenhuma informação era visível no LCD, pois o seu contraste estava desregulado. Para corrigir este problema, ajustou-se a resistência do trimpot “VR1”. A interface demonstrou funcionar exatamente como o esperado. A Figura 56 exibe a interface de usuário em funcionamento após algum tempo de uso:

Figura 56 – Interface de usuário em operação



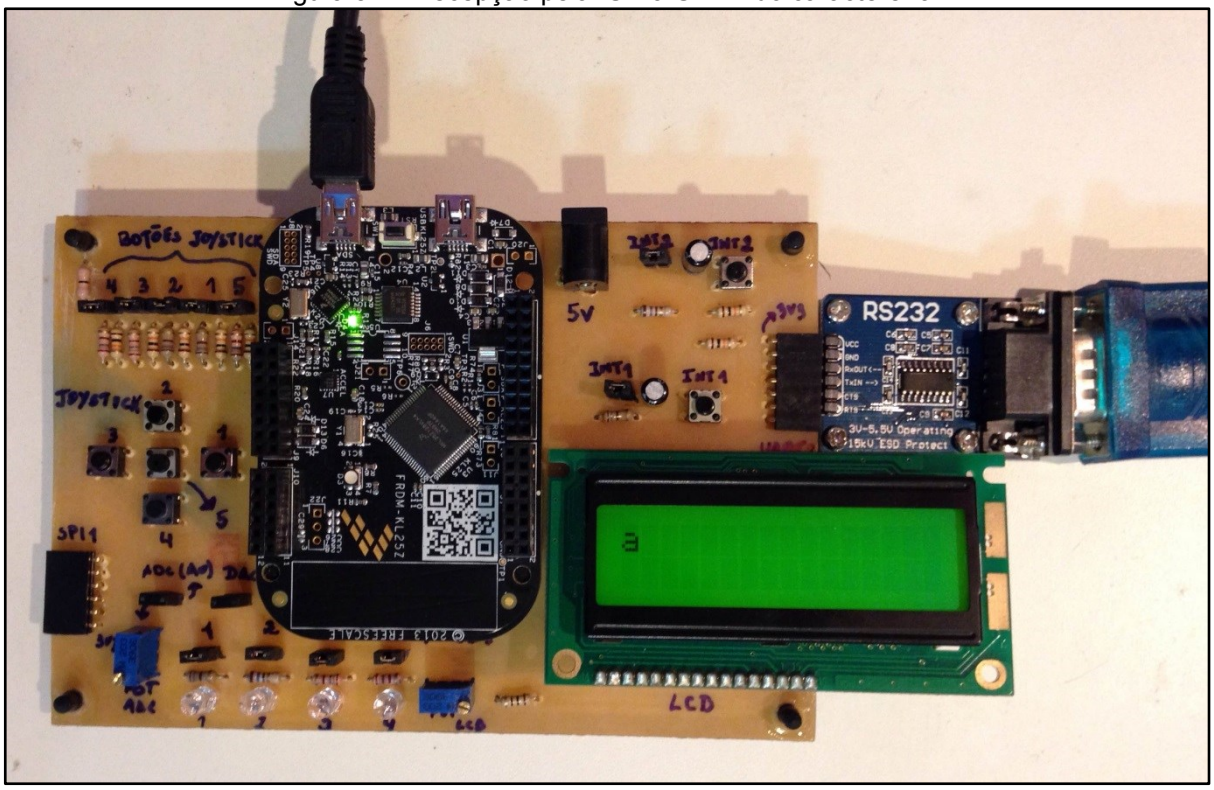
Fonte: Elaborado pelo autor

Nessa ilustração, a frequência dos LEDs havia sido aumentada com o botão de interrupção “INT.P2”, de tal forma que o piscar não era mais compreendido pelo olho. Além disso, os LEDs 1 e 3 haviam sido desligados com os botões *joystick* 1 e 3, respectivamente. A mensagem na primeira linha do LCD é respectiva à tensão lida, naquele instante, pelo canal 8 do módulo ADC0, o qual está conectado ao circuito do trimpot “VR2”. A mensagem da segunda linha é referente à tensão lida, naquele instante, pelo canal 15 do módulo ADC0, o qual está conectado no pino de saída do módulo DAC. Este último gera um sinal senoidal com valores entre 0V e 3,3V (nível DC de 1,65V e amplitude de 1,65V) e período de 24 segundos.

Para o teste da comunicação serial via módulo UART2, conectou-se a placa

“RS232 Board” à interface de usuário e, utilizando um cabo RS-232/USB, ligou-se a placa ao computador. Com o uso de um programa terminal de porta COM no computador, enviaram-se dados para a IU através de comandos pelo teclado. O terminal foi configurado com *baud rate* de 9600, 8bits de dados, sem paridade e 1 stop bit. A Figura 57 mostra a IU no momento em que se apertou o caractere “a” no teclado:

Figura 57 – Recepção pela IU via UART do caractere “a”



Fonte: Elaborado pelo autor

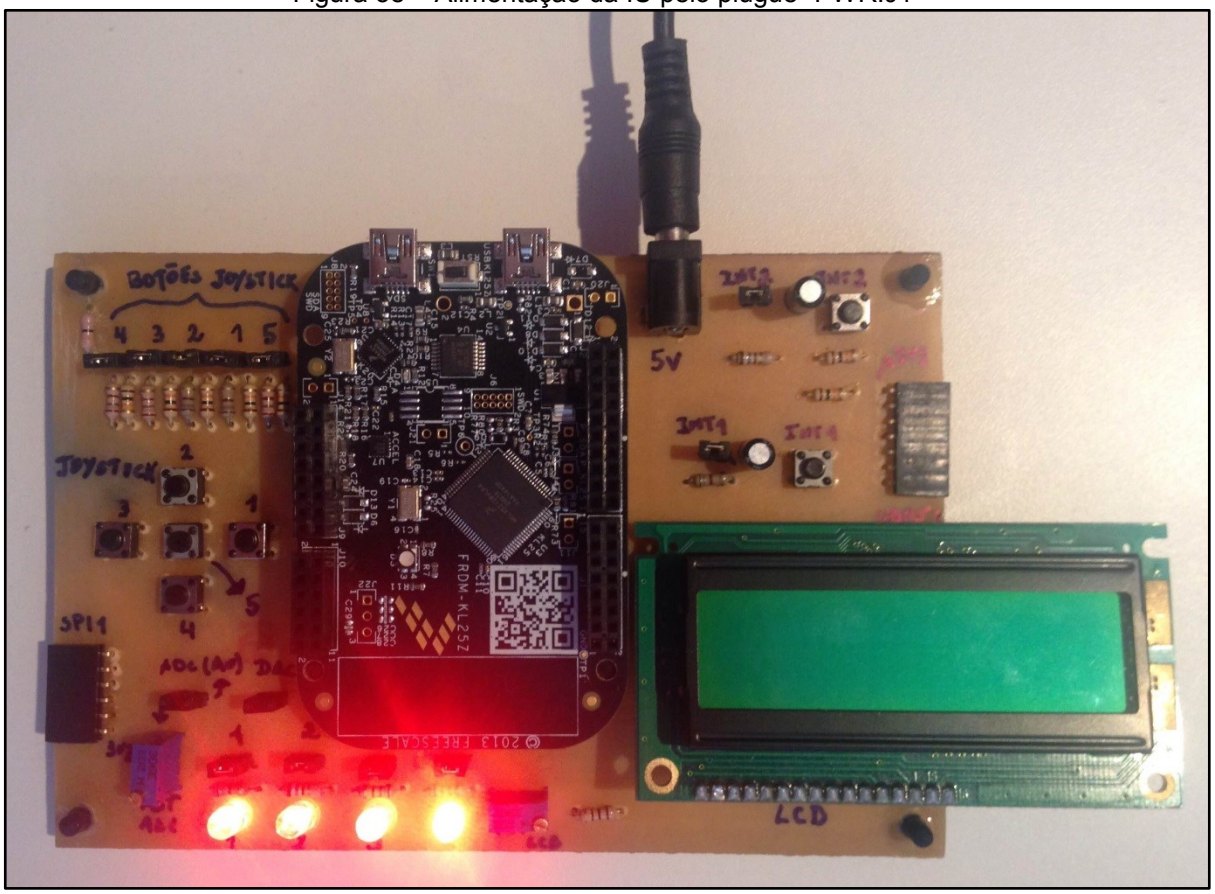
Como se pode observar, o LCD exibe o caractere que foi pressionado no teclado. Após um tempo de aproximadamente 1,2 segundos o LCD passa a exibir novamente as informações referentes aos canais 8 e 15 do ADC0, como na Figura 56. Na Figura 57 os LEDs haviam sido desligados pelo botão 5 do *joystick*.

O módulo SPI1 não foi utilizado através da IU. No entanto, testes de continuidade entre os terminais do conector da interface e os respectivos pinos na FRDM-KL25Z foram realizados com um multímetro. Constatou-se que tais conexões estavam bem estabelecidas.

Também fora testada a alimentação da interface através do plugue de “PWR.J1”, conectando-se ao plugue uma fonte de 5V. Por não se ter comprado e

soldado o regulador opcional, referente ao cabeçalho J22 da FRDM-KL25Z, o pino 9 do cabeçalho “FRDM.J9” não fornece 5V ao LCD. No entanto, todos os outros circuitos (que são alimentados por 3,3V) funcionaram normalmente. A operação da IU pode ser verificada na Figura 58:

Figura 58 – Alimentação da IU pelo plugue “PWR.J1”



Fonte: Elaborado pelo autor

Caso o usuário tenha necessidade de alimentar a interface exclusivamente pelo plugue “PWR.J1” e também utilizar o LCD, recomenda-se que seja adicionado o regulador de tensão 7805 à FRDM-KL25Z, como mostrado na Figura 17, no capítulo 2.

7 CONCLUSÃO

É possível dizer que se obteve sucesso na execução deste trabalho, visto que a interface de usuário funcionou exatamente conforme o seu projeto. No entanto, para que objetivo final do trabalho seja alcançado, é necessário que as ferramentas elaboradas sejam disponibilizadas aos estudantes da disciplina de Microprocessadores I. A PCI produzida será doada ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul e ficará aos encargos do professor Alexandre Ambrozi Junqueira. Dessa forma, os estudantes terão a oportunidade de aprender a utilizar a plataforma FRDM-KL25Z, e assim, aprimorar os seus conhecimentos sobre sistemas embarcados. Sugere-se também que a universidade financie os custos de produção de pelo menos mais oito placas, pois então todos os estudantes conseguirão ter contato com as ferramentas desenvolvidas. Todos os *links*, descritos ao longo do trabalho, para a reprodução exata do projeto, ficarão disponíveis para *download* por tempo indeterminado.

Caso seja desejado realizar algum aperfeiçoamento neste projeto, recomenda-se a alteração do trimpot “VR2” por um potenciômetro linear de 10kΩ, visto que a modificação da tensão lida pelo ADC é dificultada pela requisição do uso de uma chave de fenda. Ainda, outra melhoria que poderia futuramente ser realizada seria o projeto de um leiaute da placa em dupla face ao invés de face simples, para que assim se reduza ainda mais o efeito de *crosstalk* entre as trilhas, além de se poder obter um melhor posicionamento dos componentes. No entanto, com esta alteração para a placa em dupla face, prevê-se um aumento na complexidade de confecção da PCI e um custo de produção mais elevado.

Espera-se também que o texto deste trabalho possa ser utilizado como referência pelos estudantes para um melhor entendimento sobre o funcionamento da FRDM-KL25Z e a interface projetada.

REFERÊNCIAS

NXP. **FRDM-KL25Z**. Disponível em: <http://www.nxp.com/products/software-and-tools/hardware-development-tools/freedom-development-boards/freedom-development-platform-for-kinetis-kl14-kl15-kl24-kl25-mcus:FRDM-KL25Z>. Acesso em: 29 ago. 2016.

WIKIPEDIA. **Freescale Semiconductor**. Disponível em: <https://en.wikipedia.org/wiki/Freescale_Semiconductor>. Acesso em: 29 ago. 2016.

NXP. **FRDM-KL25Z User's Manual (REV 2.0)**. 2013. 20p. Disponível em: <http://www.nxp.com/products/software-and-tools/hardware-development-tools/freedom-development-boards/freedom-development-platform-for-kinetis-kl14-kl15-kl24-kl25-mcus:FRDM-KL25Z?fpsp=1&tab=Documentation_Tab>. Acesso em: 29 ago. 2016.

NXP. **Freedom Development Boards**. Disponível em: <http://www.nxp.com/products/software-and-tools/hardware-development-tools/freedom-development-boards:FREDEVPLA>. Acesso em: 29 ago. 2016.

WIKIPEDIA. **Arduino**. Disponível em: <https://en.wikipedia.org/wiki/Arduino>. Acesso em: 29 ago. 2016.

NXP. **Kinetis Cortex-M MCUs**. Disponível em: <http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/kinetis-cortex-m-mcus:KINETIS>. Acesso em: 31 ago. 2016.

NXP. **L Series Ultra-Low Power M0+**. Disponível em: <http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/kinetis-cortex-m-mcus/l-series-ultra-low-power-m0-plus:KINETIS_L_SERIES>. Acesso em: 31 ago. 2016.

ARROWAR. Rafael Charro. **Applications Development on the ARM® Cortex™-M0+**. [201-?]. 100 lâminas. Apresentação em PowerPoint.

WIKIPEDIA. **ARM Holdings**. Disponível em: <https://en.wikipedia.org/wiki/ARM_Holdings>. Acesso em: 15 set. 2016.

ARM. **Product Backgrounder**. Cambridge, 2005. 18p. Disponível em: <https://web.archive.org/web/20071203000700/http://www.arm.com/miscPDFs/3823.pdf >. Acesso em: 22 set. 2016.

WIKIPEDIA. **ARM architecture**. Disponível em: <https://en.wikipedia.org/wiki/ARM_architecture>. Acesso em: 20 set. 2016.

STANFORD. **What is RISC?**. Disponível em: <http://cs.stanford.edu/people/eroberts/courses/soco/projects/2000-01/risc/whatis/index.html>. Acesso em: 01 set. 2016.

STANFORD. **RISC vs. CISC**. Disponível em: <http://cs.stanford.edu/people/eroberts/courses/soco/projects/2000-01/risc/risccisc/index.html>. Acesso em: 01 set. 2016.

ARM. **ARM® Cortex®-M for Beginners (final v3)**. 2016. 25p. Disponível em: <https://community.arm.com/servlet/JiveServlet/previewBody/8587-102-4-24837/Cortex-M%20for%20Beginners%20-%202016%20(final%20v3).pdf>. Acesso em: 20 set. 2016.

INFRASTUCTURE DEVELOPER'S BLOG. **ARM/Thumb/Thumb-2**. Disponível em: <https://kmittal82.wordpress.com/2012/02/17/armthumbthumb-2/>. Acesso em: 24 nov. 2016

WIKIPEDIA. **ARM Cortex-M**. Disponível em: <https://en.wikipedia.org/wiki/ARM_Cortex-M>. Acesso em: 01 set. 2016.

ARM. **Cortex-M0+ Technical Reference Manual (Revision: r0p1)**. 2012. 51p. Disponível em: <http://infocenter.arm.com/help/topic/com.arm.doc.ddi0484c/DDI0484C_cortex_m0p_r0p1_trm.pdf>. Acesso em: 01 set. 2016.

NXP. **Kinetis KL25 Sub-Family (REV 5)**. 2014. 59p. Disponível em: <http://www.nxp.com/products/software-and-tools/hardware-development-tools/freedom-development-boards/freedom-development-platform-for-kinetis-kl14-kl15-kl24-kl25-mcus:FRDM-KL25Z?fpsp=1&tab=Documentation_Tab>. Acesso em: 31 ago. 2016.

NXP. **FRDM-KL25Z Schematics (REV E)**. 2013. 5p. Disponível em: <http://cache.nxp.com/files/soft_dev_tools/hardware_tools/schematics/FRDM-KL25Z_SCH_REV_E.pdf>. Acesso em: 22 set. 2016.

DIGI-KEY. **Electronic Components**. Disponível em: <http://www.digikey.com/>. Acesso em: 02 set. 2016.

WAVESHARE. **Store**. Disponível em: <http://www.waveshare.com/>. Acesso em: 02 set. 2016.

NXP. **Writing Touch Sensing Software Using TSI Module (REV 0)**. 2011. 11p. Disponível em: <http://www.nxp.com/files/sensors/doc/app_note/AN4330.pdf>. Acesso em: 23 set. 2016.

NXP. ***Datasheet* do circuito integrado MMA8451Q (Rev. 10.1)**. 2016. 59p. Disponível em: <http://cache.nxp.com/files/sensors/doc/data_sheet/MMA8451Q.pdf>. Acesso em: 25 set. 2016.

DOMÍNIO DA IMAGEM. **Entendendo os Espaços de Cores**. Disponível em: <http://www.escoladominiodaimagem.com/single-post/2015/02/20/Entendendo-os-Espa%C3%A7os-de-Cores->. Acesso em: 28 set. 2016.

NXP. **Kinetis SDK v.1.3 API Reference Manual (Rev. 1)**. 2016. 2895p. Disponível em: <https://cache.nxp.com/files/soft_dev_tools/doc/support_info/KSDK13APIRM.pdf>. Acesso em: 18 nov. 2016.

P&E MICROCOMPUTER SYSTEMS. **OpenSDA Support**. Disponível em: <http://www.pemicro.com/opensda/>. Acesso em: 18 nov. 2016.

ROBOCORE. **Loja Virtual**. Disponível em: <https://www.robocore.net/loja>. Acesso em: 20 nov. 2016.

PROESI. **Componentes Eletrônicos**. Disponível em: <http://proesi.com.br/>. Acesso em: 20 nov. 2016.

TE CONNECTIVITY. ***Datasheet* da tecla táctil 1825910-6**. Disponível em: <http://www.te.com/commerce/DocumentDelivery/DDEController?Action=srchrtrv&DocNm=1825910&DocType=Customer+Drawing&DocLang=English>. Acesso em: 20 nov. 2016.

CREE INC. ***Datasheet* do LED C503B-RCN-CW0Z0AA1**. Disponível em: <http://www.cree.com/~/media/Files/Cree/LED%20Components%20and%20Modules/HB/Data%20Sheets/C503B%20RAS%20RAN%20AAS%20AAN%20RBS%20RBN%20ABS%20ABN%20RCS%20RCN%20ACS%20ACN%201079.pdf>. Acesso em: 21 nov. 2016.

WAVESHARE. **RS232 Board**. Disponível em: <http://www.waveshare.com/wiki/RS232_Board>. Acesso em: 24 nov. 2016.

LIGHT-TECH. ***Datasheet* do LCD LMC-SSC2E16**. Disponível em: <http://www.ltc.com.tw/pdf/lcd2e161.pdf>. Acesso em: 21 nov. 2016.

GLOSSÁRIO

Buffer de dados – Região de uma memória digital usada para temporariamente armazenar algum dado enquanto este é transferido de um lugar a outro.

Circuito Integrado – Conjunto de circuitos eletrônicos integrados em uma pequena pastilha de material semicondutor.

Compilador – Programa de computador que transforma o código de uma linguagem de programação para outra linguagem, assim podendo ser executada em última instância.

Clock (processador) – Ciclo de máquina por segundo.

Crosstalk – Fenômeno no qual o sinal transmitido por um circuito cria um efeito indesejado em outro circuito.

Datasheet – Documento que resume o desempenho e outras características técnicas de um produto.

Debounce – Processo que interpreta o contato entre dois metais como um único sinal, ao invés da leitura de múltiplos sinais manifestados pelo ressalto do impacto.

Depuração – Processo de encontrar e resolver defeitos em *software* e/ou em *hardware*.

Diodo – Componente eletrônico de dois terminais que conduz a corrente elétrica em um sentido com muito mais facilidade do que no outro sentido.

Filtro anti-*aliasing* – Filtro utilizado previamente à amostragem de um sinal para restringir a sua largura de banda, no intuito de satisfazer o teorema da amostragem de Nyquist-Shannon.

Hardware – Parte concreta (dispositivos físicos) de um equipamento eletrônico.

Interrupção (programação de sistemas) – Sinal emitido a um processador, indicando que um evento necessita de atenção imediata.

Linker – Programa de computador que liga objetos gerados por um compilador, formando assim um arquivo executável final.

Microcontrolador – Pequeno computador em um único circuito integrado.

Multiplexação – Técnica que consiste na combinação de dois ou mais canais de informação em apenas um meio de transmissão usando um dispositivo chamado multiplexador.

Pipeline (*hardware*) – Conjunto de elementos de processamento de dados conectados em série, no qual a saída de um elemento é a entrada do próximo.

Processador (computação) – Circuito eletrônico que realiza operações com os dados de uma fonte externa.

Registrador (processador) – Local de acesso rápido (normalmente representado por um endereço) disponível em um processador, que consiste em uma pequena quantidade de armazenamento.

Sistema embarcado – Sistema digital com uma função dedicada (por exemplo, uma calculadora eletrônica).

Software – Informação codificada para o controle do funcionamento de um sistema digital.

Timer (processador) – Relógio que controla a sequência de um evento enquanto realiza contagens em intervalos de tempo.

Transdutor – Dispositivo que converte uma forma de energia em outra.

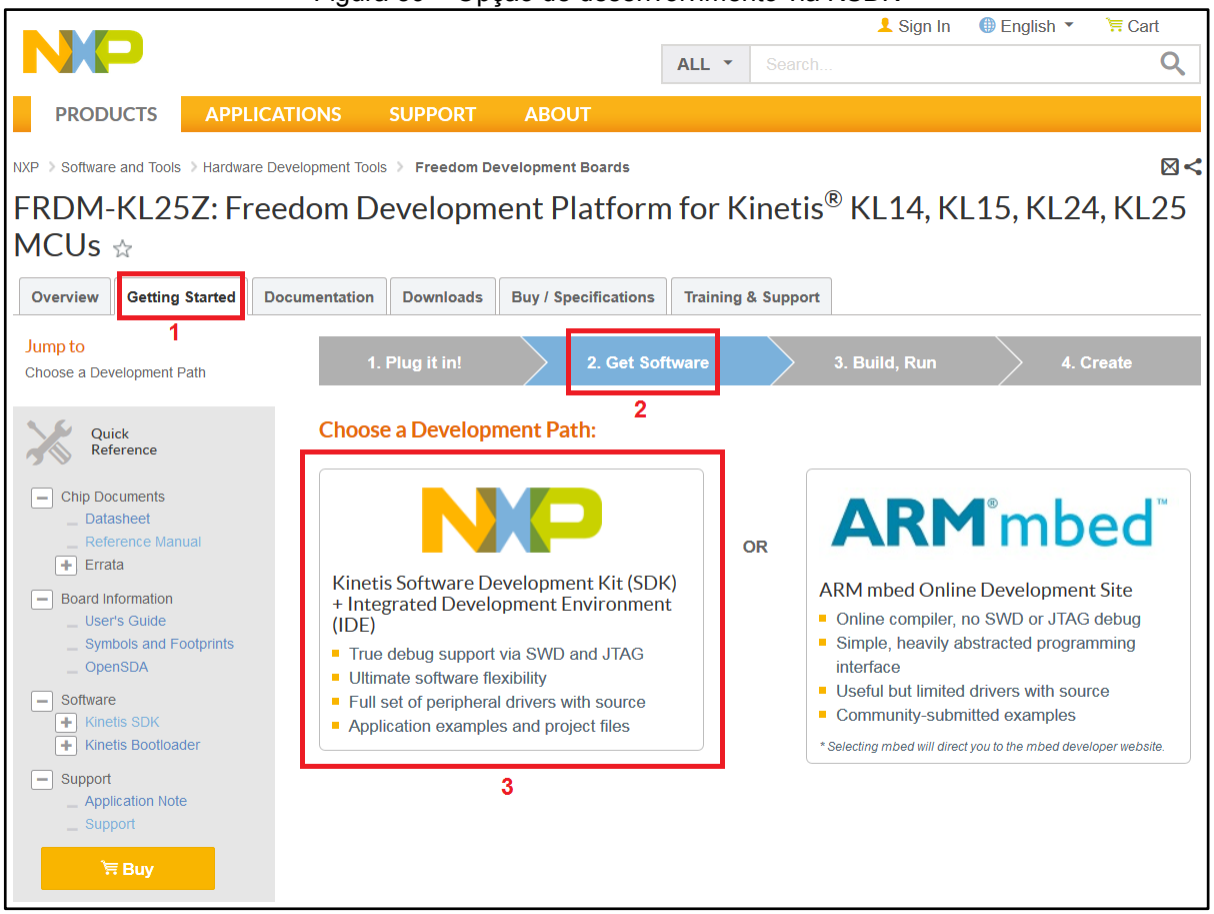
Transistor – Dispositivo semicondutor utilizado para amplificar ou chavear sinais elétricos.

APÊNDICE A – TUTORIAL DE DESENVOLVIMENTO

1. Instalação do pacote KSDK

Acesse a *homepage* da plataforma FRDM-KL25 através do endereço "http://www.nxp.com/products/software-and-tools/hardware-development-tools/freedom-development-boards/freedom-development-platform-for-kinetis-kl14-kl15-kl24-kl25-mcus:FRDM-KL25Z". Clique na aba "Getting Started", em seguida em "2. Get Software" e depois na opção "NXP", como indicado na figura Figura 59:

Figura 59 – Opção de desenvolvimento via KSDK

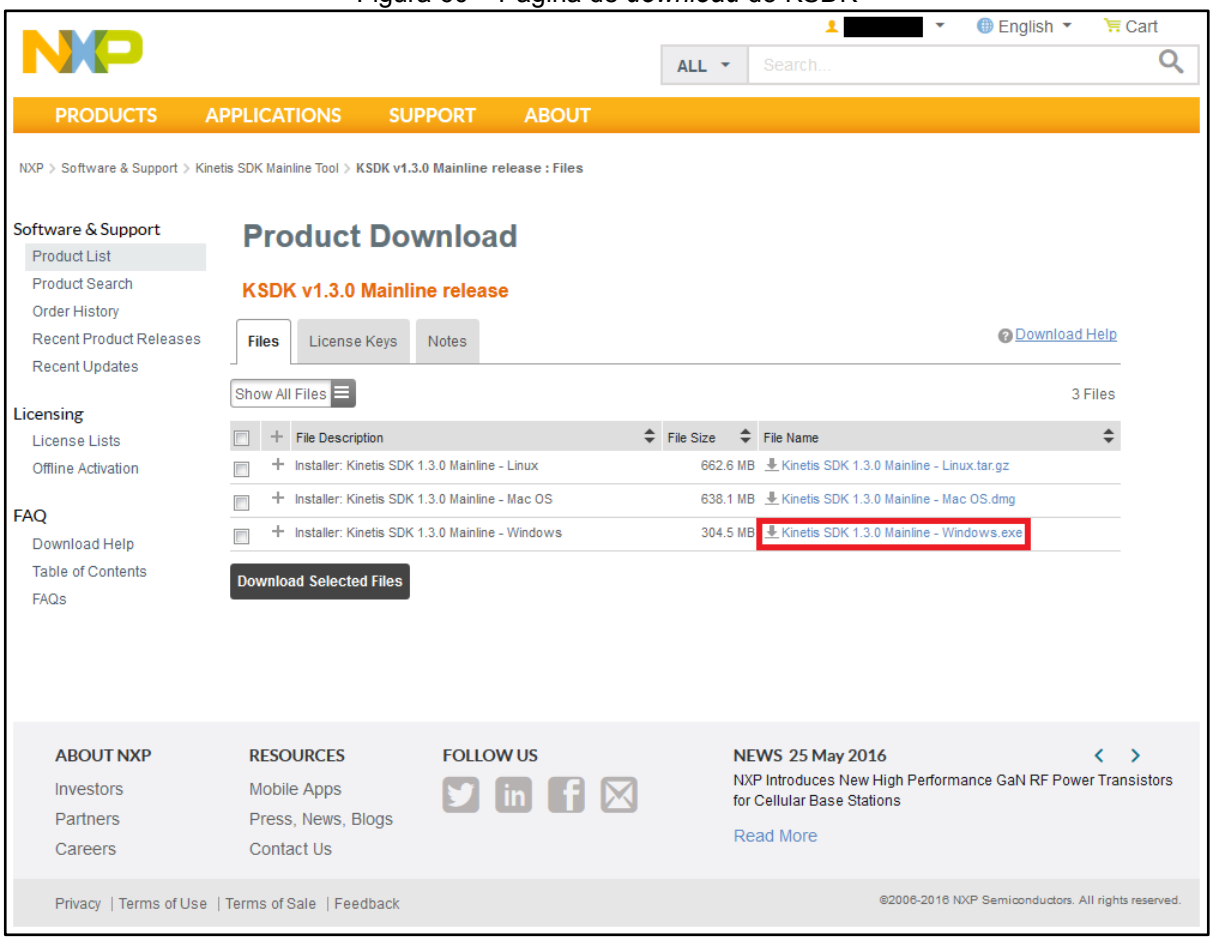


Fonte: NXP. FRDM-KL25Z. Disponível em: <http://www.nxp.com/products/software-and-tools/hardware-development-tools/freedom-development-boards/freedom-development-platform-for-kinetis-kl14-kl15-kl24-kl25-mcus:FRDM-KL25Z>. Acesso em: 29 ago. 2016.

Agora role a página para baixo até o item "2.1 Jump Start Your Design with the Kinetis SDK!" e clique no botão "Get SDK". Se você ainda não estiver conectado com um usuário e senha no *website* da NXP, uma nova aba será aberta no seu

navegador, redirecionando para uma página de *login*. Caso você ainda não esteja registrado no *website* da NXP, registre-se através do botão “Register” na própria página de *login*. Após o *login* ter sido realizado com sucesso o navegador redirecionará para a página de *download* do KSDK. Se o redirecionamento não tiver ocorrido após o *login*, clique novamente no botão “Get SDK” da aba anterior. Antes do *download* iniciar, a NXP pede que sejam aceitos os termos e condições para o uso do KSDK. Após ler, clique em “I Agree” caso você esteja de acordo. Após isso, o navegador redirecionará para a página de *download*. Existem três opções de arquivo, cada um referente ao um sistema operacional: Windows, MAC OS ou Linux. Clique no *hyperlink* referente ao arquivo para o sistema operacional do seu computador. A Figura 60 destaca essa etapa:

Figura 60 – Página de *download* do KSDK



Fonte: NXP. KSDK Download. Disponível em: <https://nxp.flexnetoperations.com/control/frse/download?agree=Accept&element=6373617>. Acesso em: 18 nov. 2016.

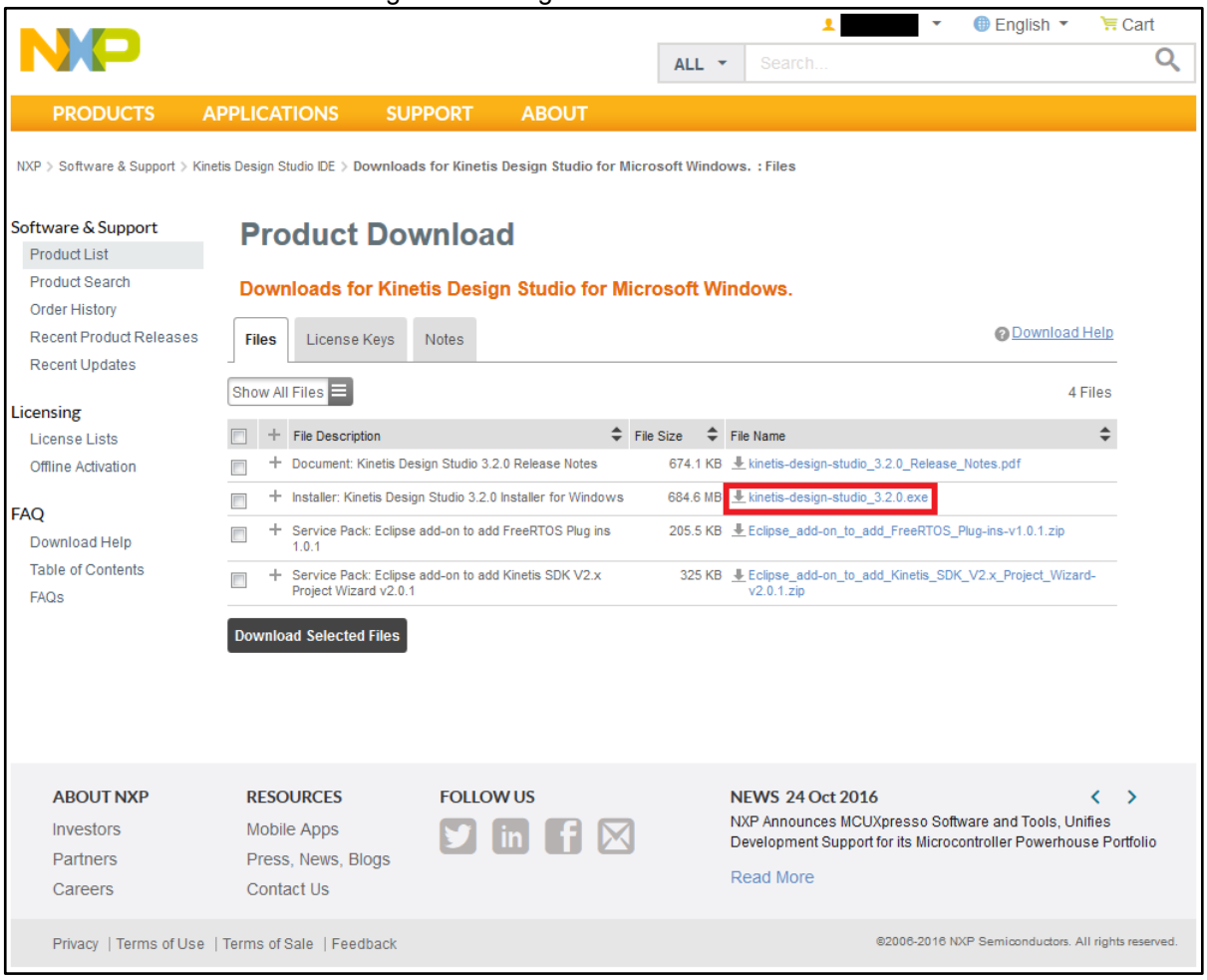
Agora, escolha o lugar de destino do arquivo em seu computador e espere o

download terminar. Assim que o *download* concluir, dê duplo clique no arquivo baixado e confirme a instalação no diretório padrão (normalmente em "C:\Freescale\KSDK_x.x.x") e espere a instalação terminar. Pronto, o KSDK está instalado no seu computador.

2. Instalação da IDE KDS

Ainda, na aba anterior do seu navegador (aquela na qual se clicou na opção NXP), role a página para baixo até o item "2.2 Install Your Toolchain" e clique no botão "Get Kinetis Design Studio". Uma nova aba será aberta nos seu navegador, redirecionando para a página de seleção do seu sistema operacional. Clique no *hyperlink* referente ao arquivo para o sistema operacional do seu computador. Após isso, a NXP pede que sejam aceitos os termos e condições para o uso do KDS. Clique em "I Agree" caso você esteja de acordo. O navegador redirecionará para a página de *download* do KDS. Clique no *hyperlink* referente ao instalador do KDS. A Figura 61 evidencia essa etapa:

Figura 61 – Página de *download* do KDS



Fonte: NXP. KDS Download. Disponível em: <https://nxp.flexnetoperations.com/control/frse/download?agree=Accept&element=7490587>. Acesso em: 18 nov. 2016.

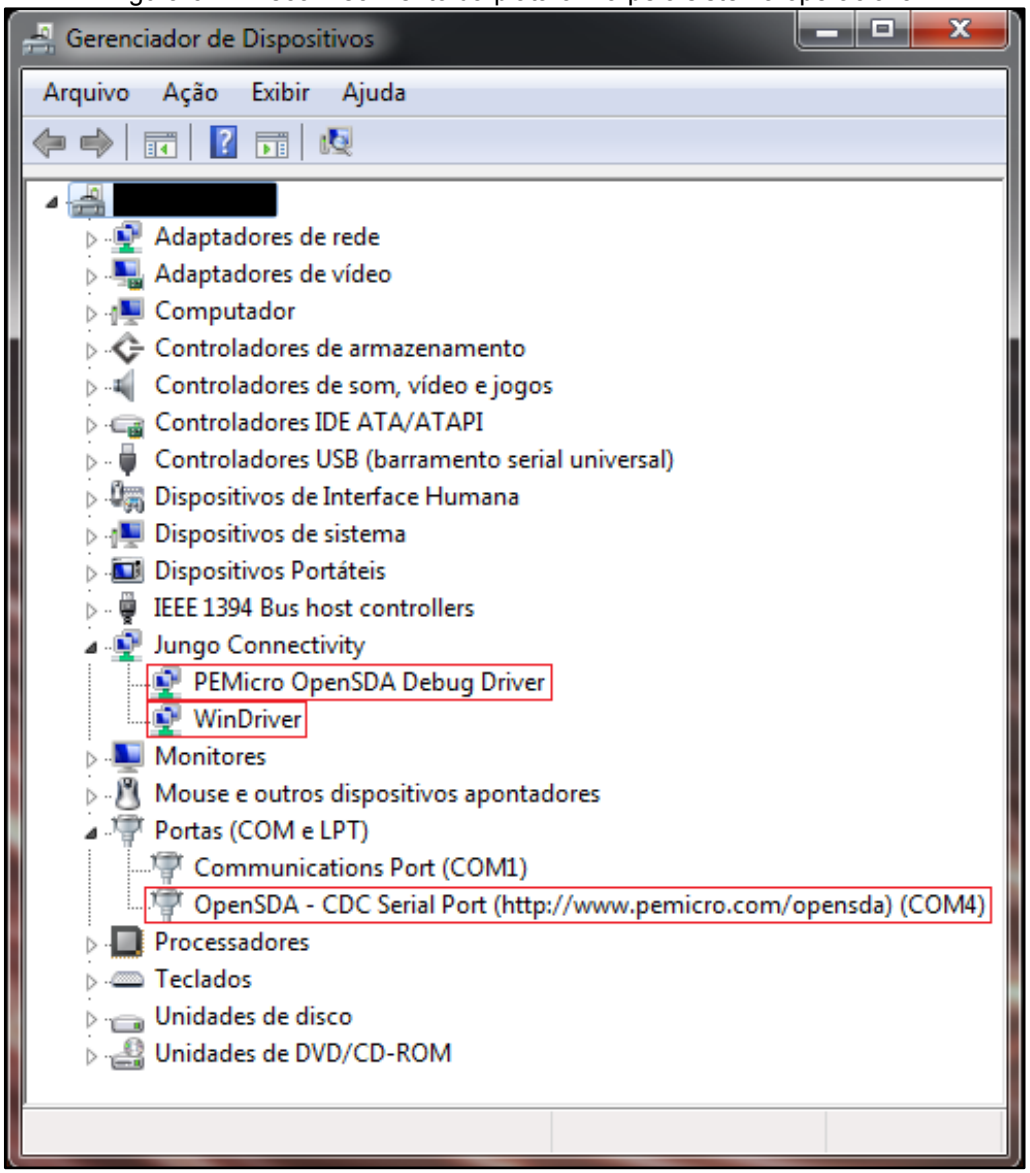
Escolha o lugar de destino do arquivo em seu computador e espere o *download* terminar. Assim que o *download* concluir, dê duplo clique no arquivo baixado e confirme a instalação no diretório padrão (normalmente em “C:\Freescale\KDS_vx”) e espere a instalação terminar. Pronto, o KDS está instalado no seu computador.

3. Primeira conexão da FRDM-KL25Z ao computador

Previamente ao início do uso da IDE KDS, é necessário configurar a comunicação da plataforma FRDM-KL25Z com o computador. Para isso, faça a conexão da plataforma pelo conector USB OpenSDA através de um cabo miniUSB/USB. Neste ponto o seu sistema operacional pode ou não reconhecer e

instalar o dispositivo automaticamente. É possível verificar se os *drivers* do dispositivo foram instalados com êxito através do gerenciador de dispositivos do sistema operacional. Caso a instalação tenha ocorrido, poderão se observar três dispositivos no gerenciador de dispositivos: dois do tipo “Jungo Connectivity” com os nomes “PEMicro OpenSDA Debug Driver” e “WinDriver” e outro do tipo “Porta COM” com o nome “OpenSDA – CDC Serial Port”. Isto está ilustrado na Figura 62:

Figura 62 – Reconhecimento da plataforma pelo sistema operacional



Fonte: Elaborado pelo autor

Caso o sistema não tenha instalado automaticamente os *drivers* do dispositivo, é possível baixá-los e instalá-los manualmente. Para fazer o *download* dos *drivers* acesse o endereço “http://www.pemicro.com/opensda/” e clique em no

hyperlink da opção “Windows USB Drivers” ou “Linux USB Drivers”, dependendo do seu sistema operacional. A Figura 63 exibe esse procedimento:

Figura 63 – *Download* dos *drivers* de comunicação da plataforma

Fonte: P&E MICROCOMPUTER SYSTEMS. OpenSDA Support. Disponível em: <http://www.pemicro.com/opensda/>. Acesso em: 18 nov. 2016.

O navegador redirecionará para a página de *download*. Para receber o *link* do arquivo é necessário fazer uma requisição. Forneça o seu e-mail no campo “E-Mail”, abaixo de onde está escrito “Or receive a direct download link via email right away”. Em poucos instantes será enviado o *link* do arquivo para o seu e-mail. Após baixá-lo dê duplo clique nele e instale normalmente. Verifique se o sistema reconheceu o dispositivo, como na ilustração da Figura 62. Caso o computador ainda não tenha reconhecido o dispositivo, o próximo passo resolverá isso.

4. Atualização e configuração da plataforma FRDM-KL25Z

Agora é necessário atualizar os arquivos de *firmware* OpenSDA. Entre no endereço “http://www.pemicro.com/opensda/” e clique no *hyperlink* da opção “OpenSDA Firmware (MSD & Debug)”, como evidenciado na Figura 64:

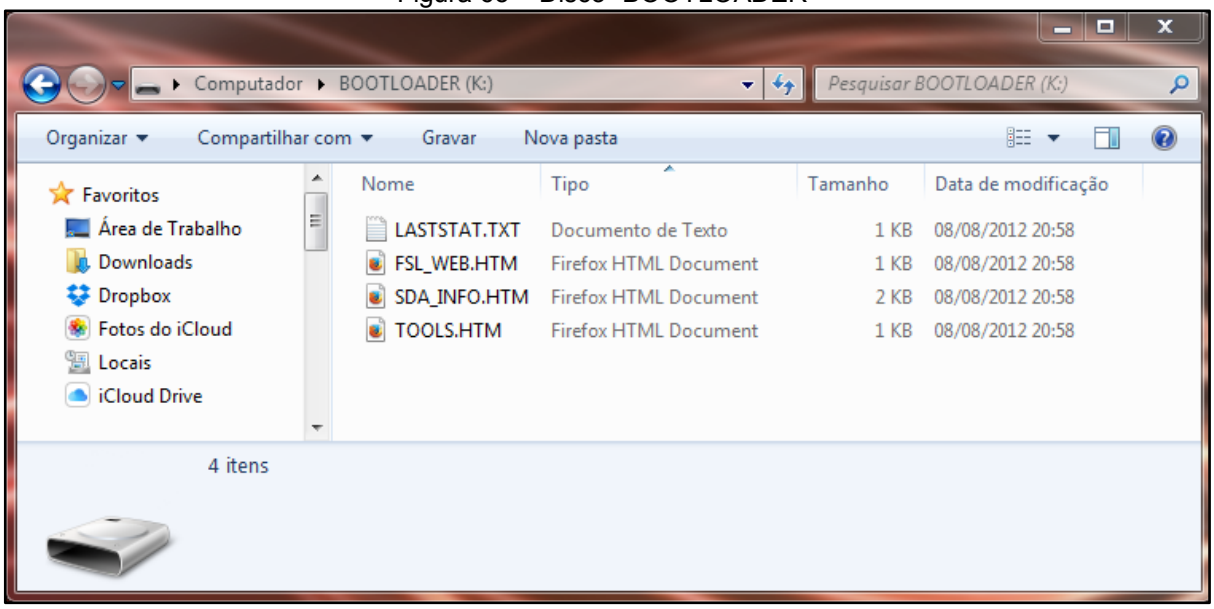
Figura 64 – *Download* do *firmware* OpenSDA

Fonte: P&E MICROCOMPUTER SYSTEMS. OpenSDA Support. Disponível em: <http://www.pemicro.com/opensda/>. Acesso em: 18 nov. 2016.

O navegador redirecionará para a página de *download*. Para receber o *link* do arquivo é necessário fazer uma requisição. Forneça o seu e-mail no campo “E-Mail”, abaixo de onde está escrito “Or receive a direct download link via email right away”. Em poucos instantes será enviado o *link* do arquivo para o seu e-mail. Após baixá-lo, descompacte o arquivo em alguma pasta vazia do seu computador. É possível notar

que entre os arquivos extraídos há um arquivo compactado com o seguinte nome: “OpenSDA_Bootloader_Update_App_v111_2013_12_11.zip” (a parte final do nome pode ser diferente por ser uma versão mais recente). Descompacte esse arquivo em uma outra pasta vazia. O arquivo extraído deve ter o nome “BOOTUPDATEAPP_Pemicro_v111.SDA” (ou versão mais recente). Agora desconecte a plataforma através do cabo USB do seu computador. Enquanto o botão de *reset* da plataforma (ao lado do conector USB OpenSDA) é pressionado, reconecte a plataforma ao computador pelo cabo USB. Após esse procedimento, um LED verde na plataforma deverá piscar com frequência de 1Hz. Seu computador também deverá reconhecer um disco chamado “BOOTLOADER”. Entre no disco como ilustrado na Figura 65:

Figura 65 – Disco “BOOTLOADER”



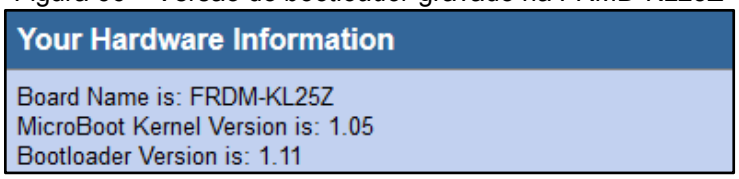
Fonte: Elaborado pelo autor

Copie o arquivo “BOOTUPDATEAPP_Pemicro_v111.SDA” para dentro do disco. A cópia dura em torno de 2 a 3 segundos, e o LED deve continuar piscando com frequência de 1Hz após a operação. Se o LED piscar mais rápido do que isso, a operação de cópia falhou e o todo procedimento do *bootloader* deve ser realizado novamente.

Agora desconecte o cabo USB da plataforma e, depois de passados cinco segundos, reconecte, desta vez sem pressionar o botão de *reset*. Novamente o computador deverá reconhecer o disco “BOOTLOADER” e o LED verde da

plataforma deverá piscar em 1Hz. É possível conferir se o *bootloader* foi corretamente programado abrindo o arquivo “BOOTLOADER\SDA_INFO.HTM” com um navegador *web*. A versão do *bootloader* deve ser maior ou igual a “1.11”, como destacado na Figura 66:

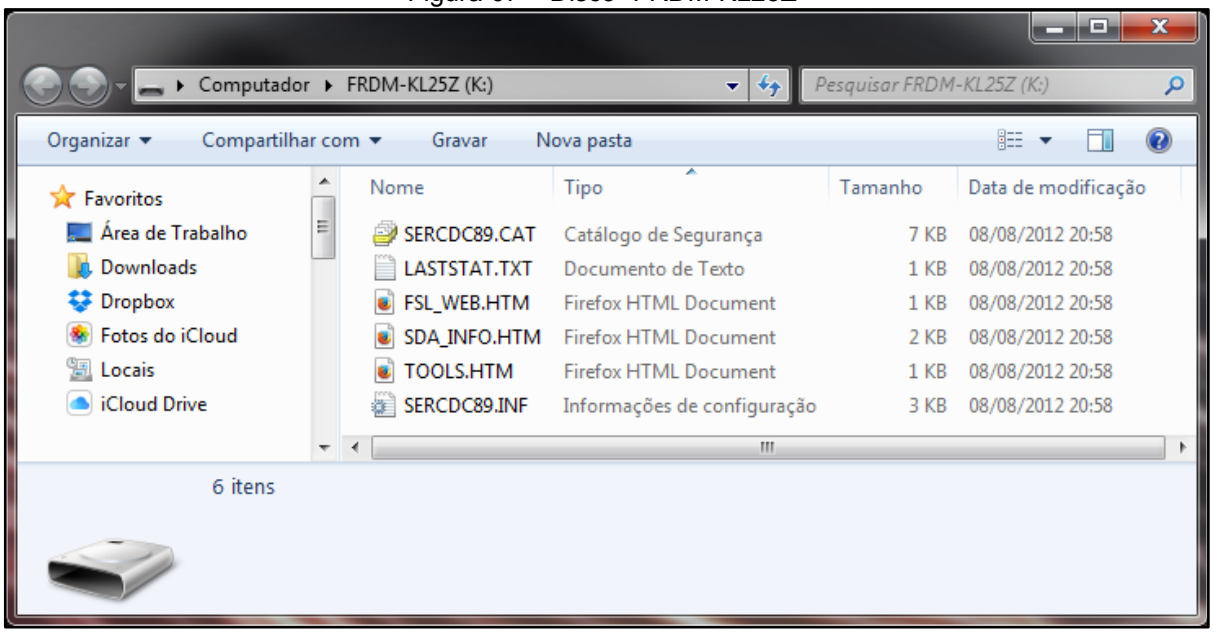
Figura 66 – Versão do *bootloader* gravado na FRMD-KL25Z



Fonte: Elaborado pelo autor

Agora que o *bootloader* já foi atualizado é preciso atualizar o *firmware* da plataforma. Novamente, desconecte a plataforma pelo cabo USB e reconecte com o botão de *reset* estando pressionado. Dentre os arquivos que foram extraídos na primeira descompactação, localize o arquivo “MSD-DEBUG-FRDM-KL25Z_Pemicro_vXXX.SDA” (XXX é o número da versão) e o copie para dentro do disco “BOOTLOADER”. Pela última vez, desconecte a plataforma pelo cabo USB e conecte novamente sem pressionar o botão de *reset*. O seu sistema operacional agora deve reconhecer o disco da plataforma com o nome “FRDM-KL25Z” ao invés de “BOOTLOADER”, como exibido na Figura 67:

Figura 67 – Disco “FRDM-KL25Z”



Fonte: Elaborado pelo autor

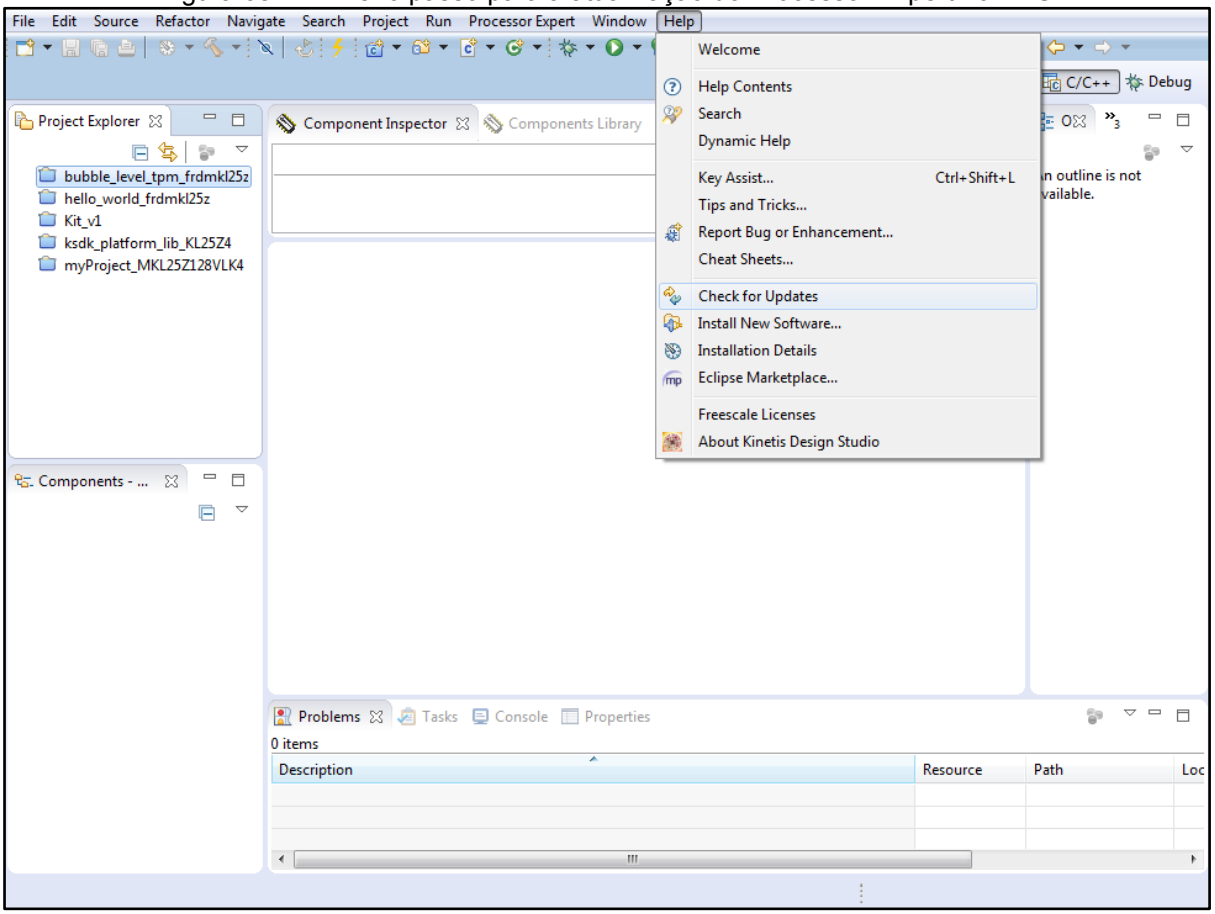
Nesta etapa o LED verde da plataforma deve estar permanentemente “aceso” e o sistema operacional deve ter reconhecido o dispositivo, como na ilustração da Figura 62. Pronto, agora a plataforma FRDM-KL25Z está configurada e atualizada para ser utilizada através de uma IDE.

5. Atualização e configuração da IDE KDS

Para se utilizar todas as vantagens da IDE KDS e sincronizá-la com o pacote KSDK, é importante que todos os passos nesta seção sejam seguidos. Primeiramente, abra a IDE KDS no seu computador. Por ser a primeira vez de execução da IDE, antes de tudo o programa irá perguntar o seu local desejado para uso como *workspace*. O *workspace* é a pasta em que todos os projetos de *software* desenvolvidos através da IDE são salvos. É possível escolher qualquer lugar do seu computador para isso. No caso deste tutorial, o local estabelecido foi o diretório “C:\Users\Fulano\Documents\Workspace-Eclipse\KDS”. Após informar o diretório, selecione a caixa “Use this as default and do not ask again” para que a IDE entenda que este será o seu *workspace* definitivo. Clique em “OK”. Em seguida o programa abrirá a interface na tela de boas vindas. Essa tela aparece somente na primeira vez de execução do KDS. Clique no botão “Workbench” para fechar a tela de boas vindas e abrir a tela dos projetos.

O próximo passo é atualizar o Processor Expert, que é uma ferramenta da NXP disponível para uso em IDEs e extremamente útil no desenvolvimento dos programas. Ele fornece uma maneira gráfica, simples e rápida para geração dos códigos de configuração e inicialização dos periféricos do microcontrolador através dos *drivers* do pacote KSDK. Para atualizá-lo clique na aba “Help” e depois em “Check for Updates”. Este passo está ilustrado na Figura 68:

Figura 68 – Primeiro passo para a atualização do Processor Expert no KDS



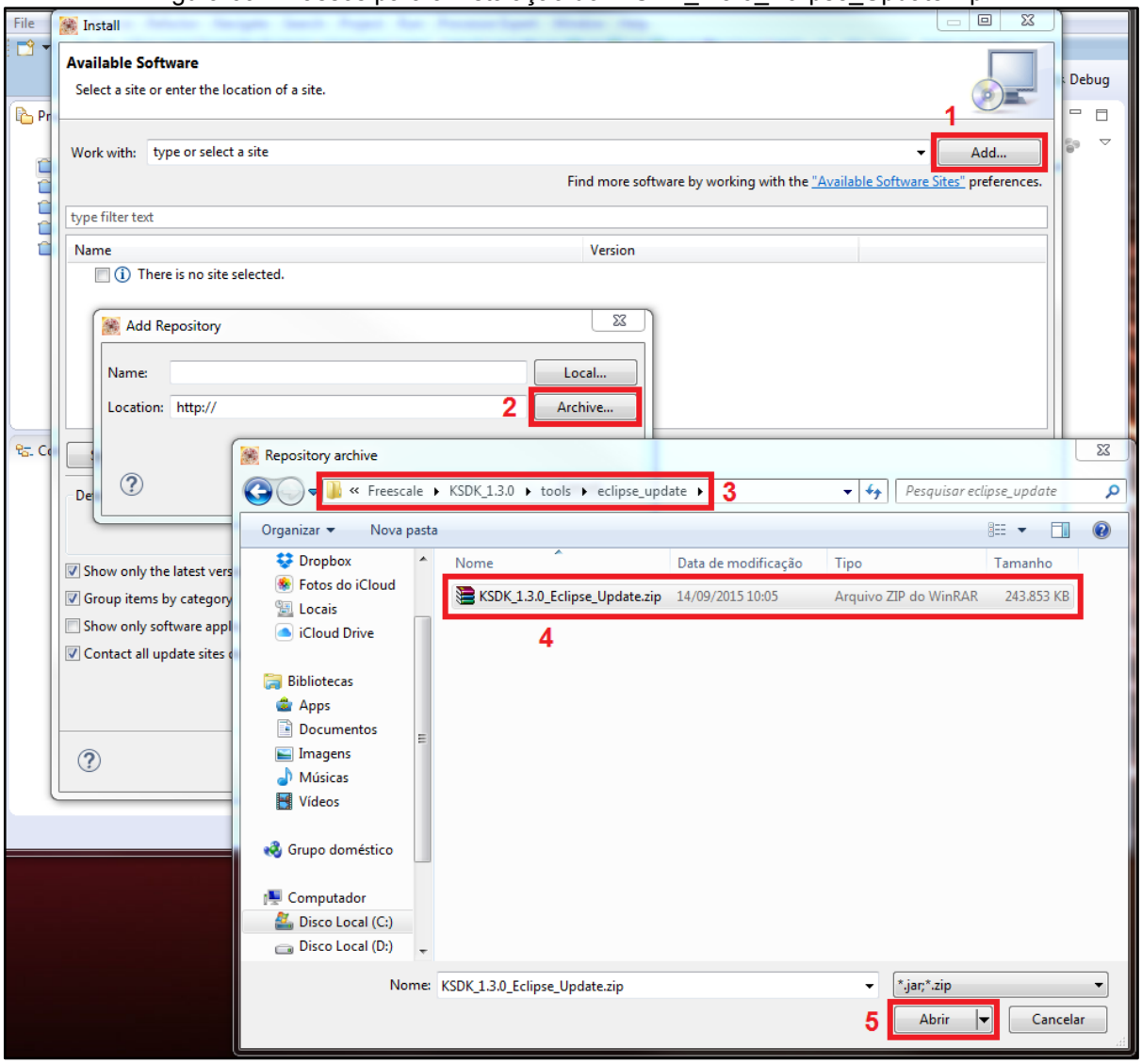
Fonte: Elaborado pelo autor

Em seguida será aberta uma janela que mostra todas as atualizações possíveis. Selecione apenas a caixa “Processor Expert for Kinetis”, clique em “Next” e em “Next” novamente. Antes da atualização iniciar, a NXP pede que sejam aceitos os termos e condições para o uso do Processor Expert. Após ler, clique na opção “I accept the terms of the license agreement”, caso você esteja de acordo, e após isso em “Finish”. O Processor Expert será atualizado e, ao finalizar, uma janela aparecerá informando que o KDS deve ser reiniciado para as alterações tomarem efeito. Clique em “Yes” para que a reinicialização do KDS seja feita automaticamente.

Agora é necessário instalar no KDS um arquivo chamado “KSDK_1.3.0_Eclipse_Update.zip”, que possibilita que a IDE gere projetos compatíveis com os *drivers* do pacote KSDK. Clique na aba “Help” e em seguida em “Install New Software...”. Uma janela será aberta para o usuário especificar qual é o arquivo que se deseja instalar. Clique em “Add” e depois em “Archive...”. Busque o arquivo dentro do diretório onde o KSDK foi instalado (o arquivo provavelmente estará em “C:\Freescale\KSDK_1.3.0\tools\eclipse_update”). Clique uma vez no

arquivo e depois em “Abrir”. A Figura 69 evidencia estes últimos passos:

Figura 69 – Passos para a instalação do “KSDK_1.3.0_Eclipse_Update.zip”



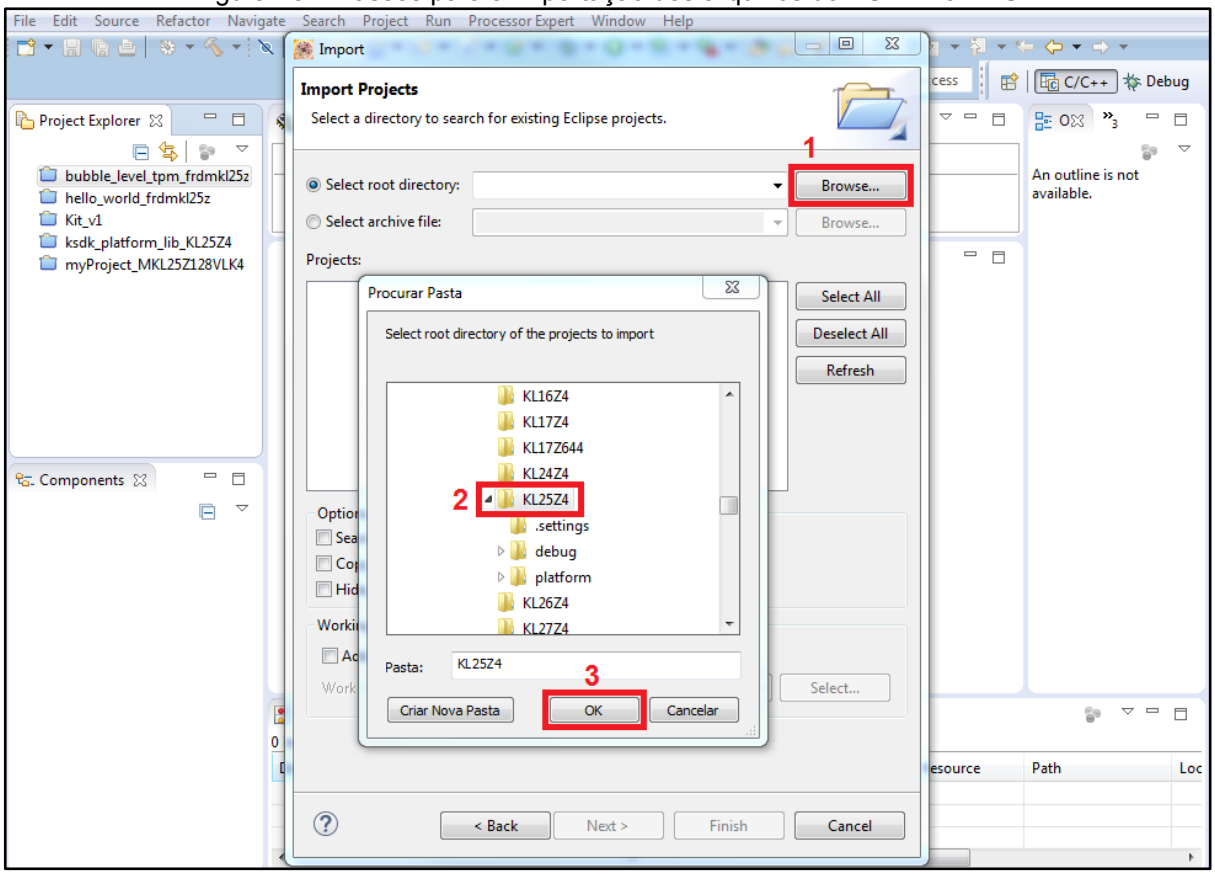
Fonte: Elaborado pelo autor

Clique em “OK”. Agora selecione a caixa referente ao arquivo “KSDK 1.3.0 Eclipse Update”, clique em “Next” e em “Next” novamente. Antes da instalação iniciar, a NXP pede que sejam aceitos os termos e condições. Após ler, clique na opção “I accept the terms of the license agreement”, caso você esteja de acordo, e após isso em “Finish”. O arquivo será instalado e, ao finalizar, uma janela aparecerá informando que o KDS deve ser reiniciado para as alterações tomarem efeito. Clique em “Yes” para que a reinicialização do KDS seja feita automaticamente.

Agora chegou a vez de importar para o KDS a biblioteca de arquivos que contém os *drivers* do pacote KSDK para a plataforma FRDM-KL25Z. Clique na aba

“File” e depois em “Import...”. Na janela que se abre selecione dentro da pasta “General” a opção “Existing Project into Workspace” e depois clique em “Next”. Agora clique em “Browse...”, procure pela pasta “KL25Z4” dentro do diretório de instalação do KSDK (a pasta provavelmente estará em “C:\Freescale\KSDK_1.3.0\lib\ksdk_platform_lib\kds”) e clique em “OK”. A Figura 70 ilustra estes últimos passos:

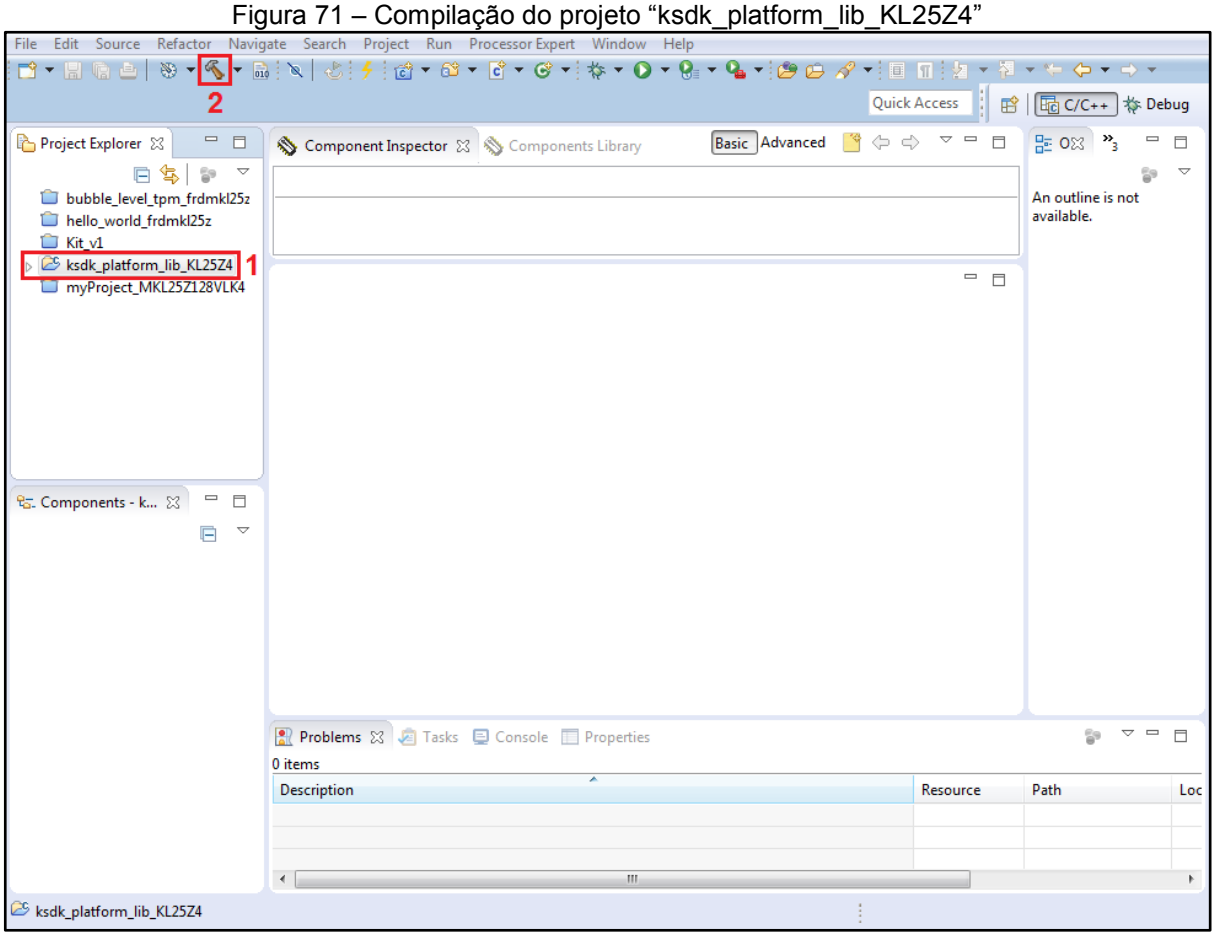
Figura 70 – Passos para a importação dos arquivos do KSDK no KDS



Fonte: Elaborado pelo autor

Por final clique em “Finish” para importar a biblioteca. É importante que a biblioteca seja compilada para que a IDE consiga identificar os *drivers*. Na *view* “Project Explorer” a biblioteca importada terá o nome “ksdk_platform_lib_KL25Z4”. Para compilar o projeto é necessário que este esteja aberto, o qual é indicado por um ícone de pasta aberta. Caso o ícone do projeto esteja indicado por uma pasta fechada, clique sobre ele com o botão direito do *mouse* e em seguida clique em “Open Project”. Agora selecione o projeto clicando sobre ele e depois compile clicando sobre o ícone “Build”, representado por um martelo. A Figura 71 destaca estes passos:

Figura 71 – Compilação do projeto “ksdk_platform_lib_KL25Z4”



Fonte: Elaborado pelo autor

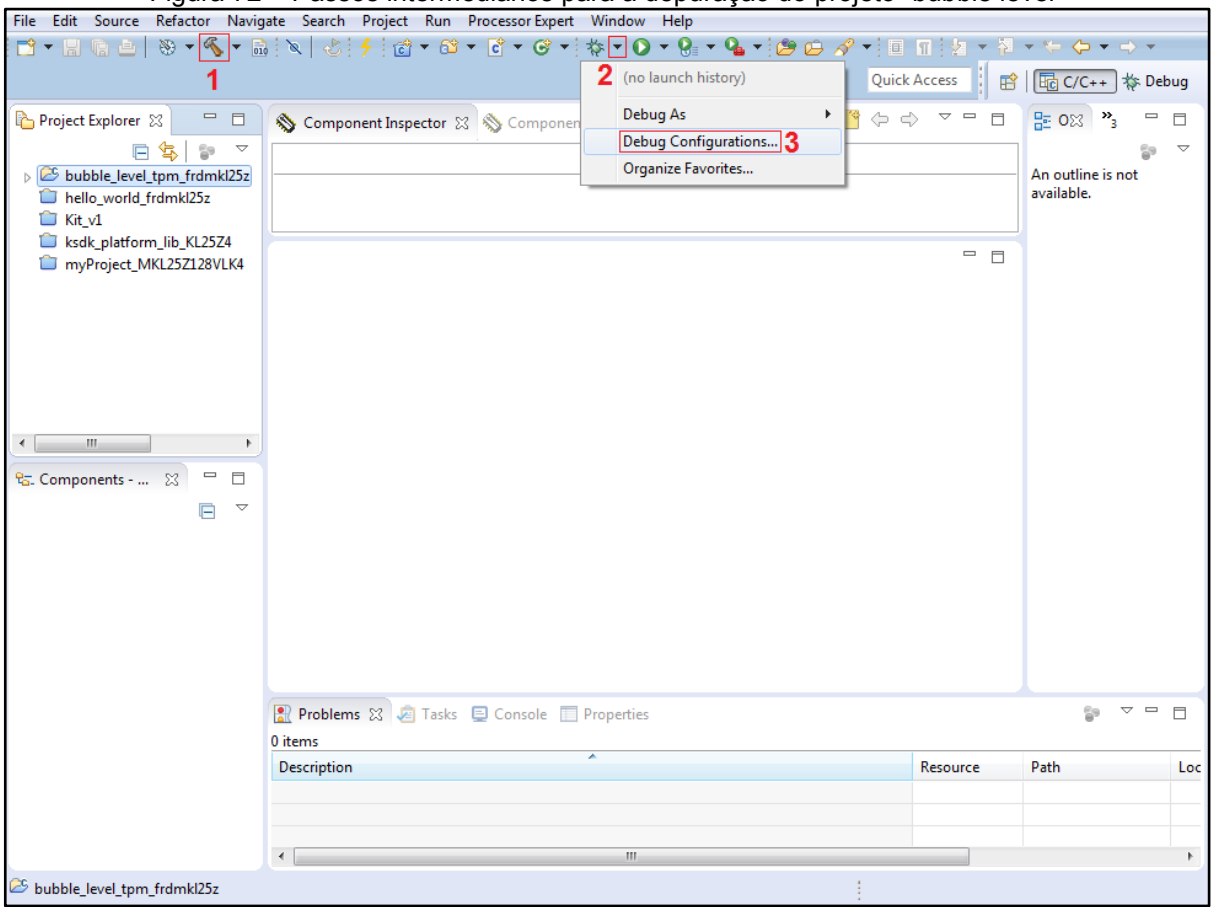
Uma janela que mostra o processo de compilação será aberta. Pronto, agora é possível desenvolver projetos na IDE KDS utilizando o KSDK.

6. Gravação, execução e depuração do programa “bubble level”

Para a concretização de todos os passos anteriores, esta seção explica como realizar a gravação, execução e depuração de um programa na FRDM-KL25Z através do KDS. O program “bubble level” é um projeto exemplo que vem junto com o pacote KSDK e é bastante útil na verificação do correto funcionamento de todas as partes. Importe o projeto “bubble level” da mesma maneira que se importou o projeto “ksdk_platform_lib_KL25Z4” na seção anterior. Nesta importação busque pela pasta “kds” no diretório “C:\Freescale\KSDK_1.3.0\examples\frdmkl25z\demo_apps\bubble_level_tpm”. Na *view* “Project Explorer” o projeto importado terá o nome “bubble_level_tpm_frdmkl25z”. Com o projeto aberto e selecionado, compile através

do ícone representado por um martelo. Após isso, clique na flecha ao lado do ícone representado por inseto e em seguida em “Debug Configurations...”. Estes últimos passos estão evidenciados na Figura 72:

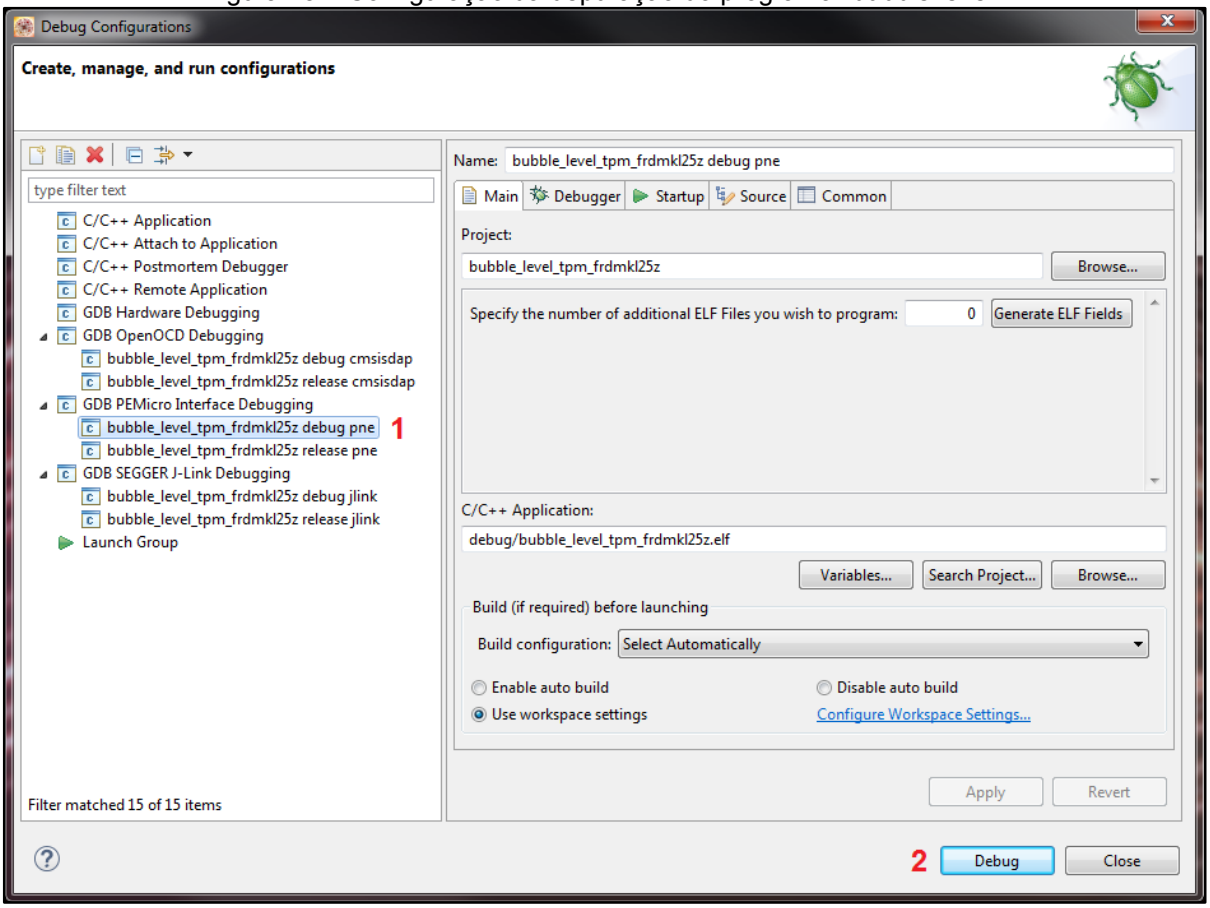
Figura 72 – Passos intermediários para a depuração do projeto “bubble level”



Fonte: Elaborado pelo autor

A janela de configuração de depuração do programa será aberta. Clique na opção “bubble_level_tpm_frdmkl25z debug pne” dentro da opção “GDB PEMicro Interface Debugging” (lembrando que anteriormente foram instalados os *drivers* da PEMicro para a depuração dos programas na plataforma). Finalmente clique em “Debug”. Estes passos estão ilustrados na Figura 73:

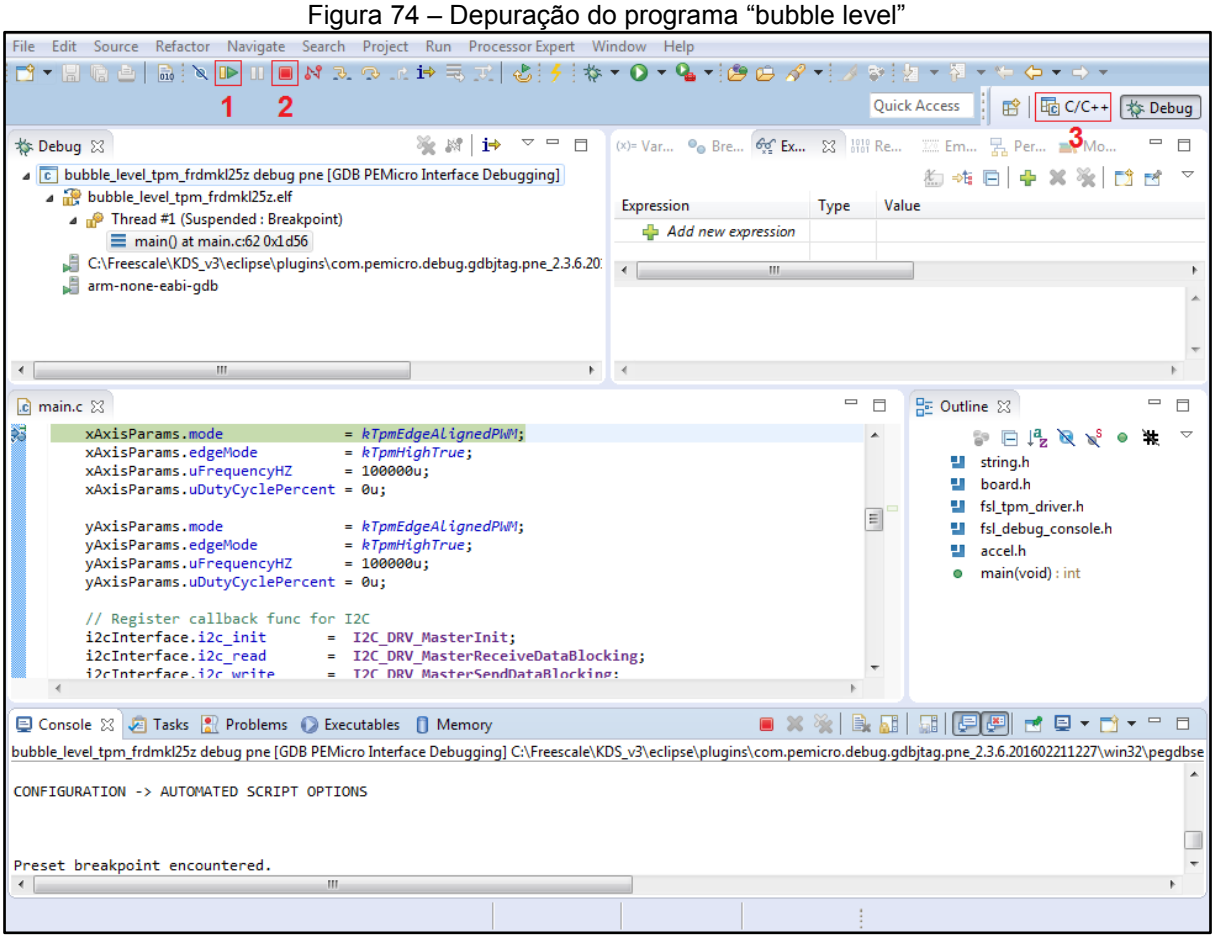
Figura 73 – Configuração da depuração do programa “bubble level”



Fonte: Elaborado pelo autor

A IDE perguntará se é desejado trocar a perspectiva de escrita dos códigos para a perspectiva de depuração. Clique em “Yes”. Agora clique no ícone *resume* (indicado por uma barra amarela e uma flecha verde) para iniciar a execução do programa na plataforma FRDM-KL25Z. Enquanto o programa está sendo executado é possível clicar no botão *pause* (indicado por duas barras amarelas) para pausar a execução do programa e verificar qual foi a última linha de código executada (esta opção de depuração é possível graças ao circuito MTB do Cortex-M0+, já explicado na seção 2.2.4). Para terminar com a execução do programa basta clicar no botão *terminate* (indicado por um quadrado vermelho) e para voltar para a perspectiva de escrita na IDE clique no botão indicado por “C/C++”. Estes passos estão ilustrados na Figura 74:

Figura 74 – Depuração do programa “bubble level”

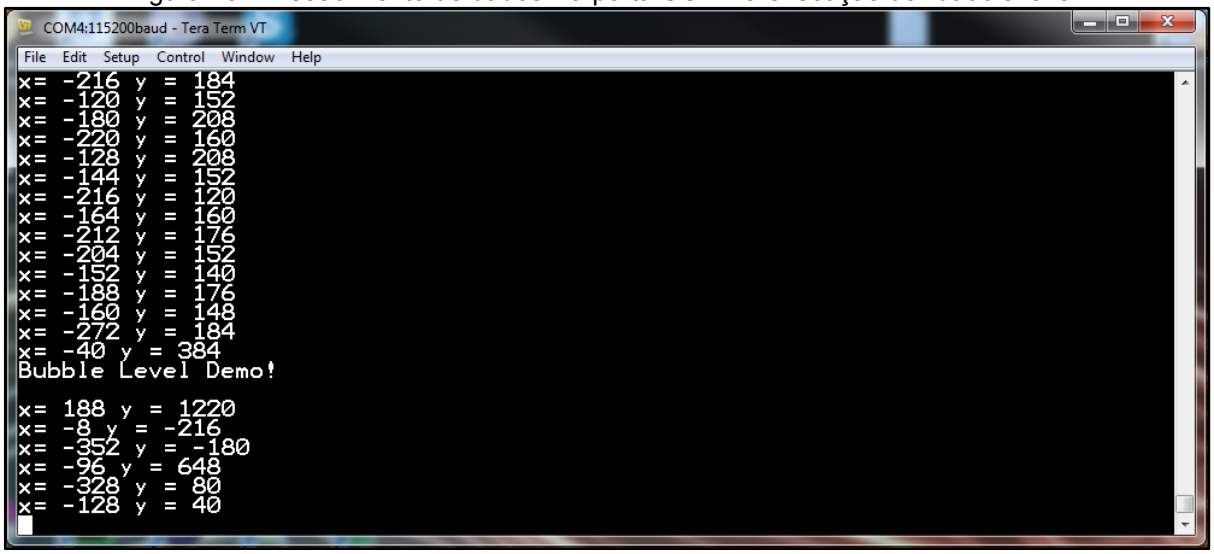


Fonte: Elaborado pelo autor

Após o término da depuração, o MCU para definitivamente de executar o programa. Para fazê-lo executar novamente sem depurar aperte no botão *reset* da plataforma.

Pode-se verificar que este programa faz com que a rotação no eixo X ou Y do acelerômetro resulte em mudança de cor no LED RGB. Também, há transmissão de dados das informações da rotação no eixo X e Y através de interface UART. É possível receber esses dados por um programa terminal de porta COM (serial) instalado no computador (como o Tera Term). Para o coerente recebimento dos dados deve-se especificar no terminal a porta correta (verificar no gerenciador de dispositivos qual é a porta COM respectiva ao OpenSDA), *baud rate* de 115200, 8bits de dados, sem paridade e 1 *stop* bit. A Figura 75 mostra o recebimento dos dados pelo Tera Term logo após se ter apertado o botão *reset* da plataforma:

Figura 75 – Recebimento de dados via porta COM na execução do “bubble level”



Fonte: Elaborado pelo autor

APÊNDICE B – CRIAÇÃO DOS COMPONENTES DA IU NO KDS

1. Módulos TPM (LEDs)

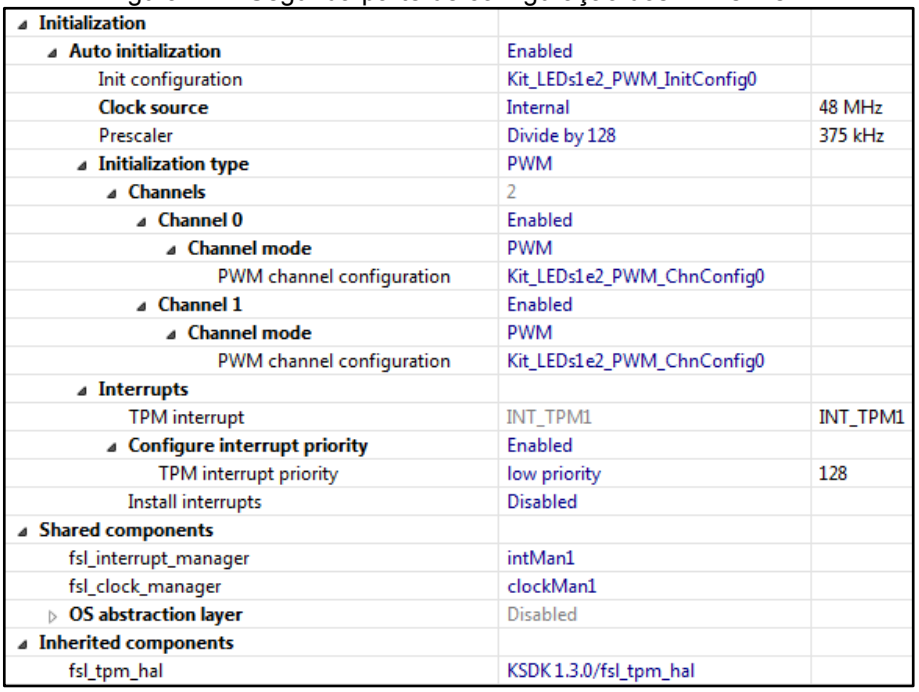
O módulo TMP1 controla os LEDs 1 e 2 e o módulo TPM0 controla os os LEDs 3 e 4. Logo, devem ser criados dois componentes de TPM (procurar por “tpm” na Components Library e criar dois componentes “fsl_tpm”). Selecione o primeiro componente criado na *view* Components e o configure da forma ilustrada pelas Figuras 76 e 77:

Figura 76 – Primeira parte de configuração dos LEDs 1 e 2

Name	Value	Details
Component name	Kit_LEDs1e2_PWM	
Device	TPM1	TPM1
Component version	1.3.0	
Configurations	Enabled	
Basic configurations	Enabled	
Configurations list	1	
Configuration 0	Enabled	
Name	Kit_LEDs1e2_PWM_InitConfig0	
Type	tpm_general_config_t	
Read only	Disabled	
Debug mode	Disabled	
Global time base	Disabled	
Trigger mode	Disabled	
Stop count on overflow	Disabled	
Reload counter on trigger	Disabled	
Trigger source	TPM trigger source 0	
PWM channels configurations	Enabled	
Configurations list	1	
Configuration 0	Enabled	
Name	Kit_LEDs1e2_PWM_ChnConfig0	
Type	tpm_pwm_param_t	
Read only	Disabled	
Mode	Edge aligned	
Edge mode	High true	
Frequency	6 D	6 Hz
Duty cycle	50 D	50 %
Pins		
Channel 0	Enabled	
Pin	J10_1	ADC0_DP0/ADC0_SE0/PTE20/TPM1_CH0/UART0_TX
Channel 1	Enabled	
Pin	J10_3	ADC0_DM0/ADC0_SE4a/PTE21/TPM1_CH1/UART0_RX
Channel 2	Disabled	
Channel 3	Disabled	
Channel 4	Disabled	
Channel 5	Disabled	
Channel 6	Disabled	
Channel 7	Disabled	
External clock input	Disabled	
Quadrature decoder phase 0 input	Disabled	
Quadrature decoder phase 1 input	Disabled	

Fonte: Elaborado pelo autor

Figura 77 – Segunda parte de configuração dos LEDs 1 e 2



Initialization		
Auto initialization	Enabled	
Init configuration	Kit_LEDs1e2_PWM_InitConfig0	
Clock source	Internal	48 MHz
Prescaler	Divide by 128	375 kHz
Initialization type	PWM	
Channels	2	
Channel 0	Enabled	
Channel mode	PWM	
PWM channel configuration	Kit_LEDs1e2_PWM_ChnConfig0	
Channel 1	Enabled	
Channel mode	PWM	
PWM channel configuration	Kit_LEDs1e2_PWM_ChnConfig0	
Interrupts		
TPM interrupt	INT_TPM1	INT_TPM1
Configure interrupt priority	Enabled	
TPM interrupt priority	low priority	128
Install interrupts	Disabled	
Shared components		
fsl_interrupt_manager	intMan1	
fsl_clock_manager	clockMan1	
OS abstraction layer	Disabled	
Inherited components		
fsl_tpm_hal	KSDK 1.3.0/fsl_tpm_hal	

Fonte: Elaborado pelo autor

Agora, da mesma forma que no primeiro componente, use as Figuras 78 e 79 para configurar os LEDs 3 e 4:

Figura 78 – Primeira parte de configuração dos LEDs 3 e 4

Name	Value	Details
Component name	Kit_LEDs3e4_PWM	
Device	TPM0	TPM0
Component version	1.3.0	
Configurations	Enabled	
Basic configurations	Enabled	
Configurations list	1	
Configuration 0	Enabled	
Name	Kit_LEDs3e4_PWM_InitConfig0	
Type	tpm_general_config_t	
Read only	Disabled	
Debug mode	Disabled	
Global time base	Disabled	
Trigger mode	Disabled	
Stop count on overflow	Disabled	
Reload counter on trigger	Disabled	
Trigger source	TPM trigger source 0	
PWM channels configurations	Enabled	
Configurations list	1	
Configuration 0	Enabled	
Name	Kit_LEDs3e4_PWM_ChnConfig0	
Type	tpm_pwm_param_t	
Read only	Disabled	
Mode	Edge aligned	
Edge mode	High true	
Frequency	6 D	6 Hz
Duty cycle	50 D	50 %
Pins		
Channel 0	Disabled	
Channel 1	Disabled	
Channel 2	Enabled	
Pin	J10_9	CMP0_IN5/ADC0_SE4b/PTE29/TPM0_CH2/TPM_CLKIN0
Channel 3	Disabled	
Channel 4	Enabled	
Pin	J2_13	PTE31/TPM0_CH4
Channel 5	Disabled	
Channel 6	Disabled	
Channel 7	Disabled	
External clock input	Disabled	
Quadrature decoder phase 0 input	Disabled	
Quadrature decoder phase 1 input	Disabled	

Fonte: Elaborado pelo autor

Figura 79 – Segunda parte de configuração dos LEDs 3 e 4

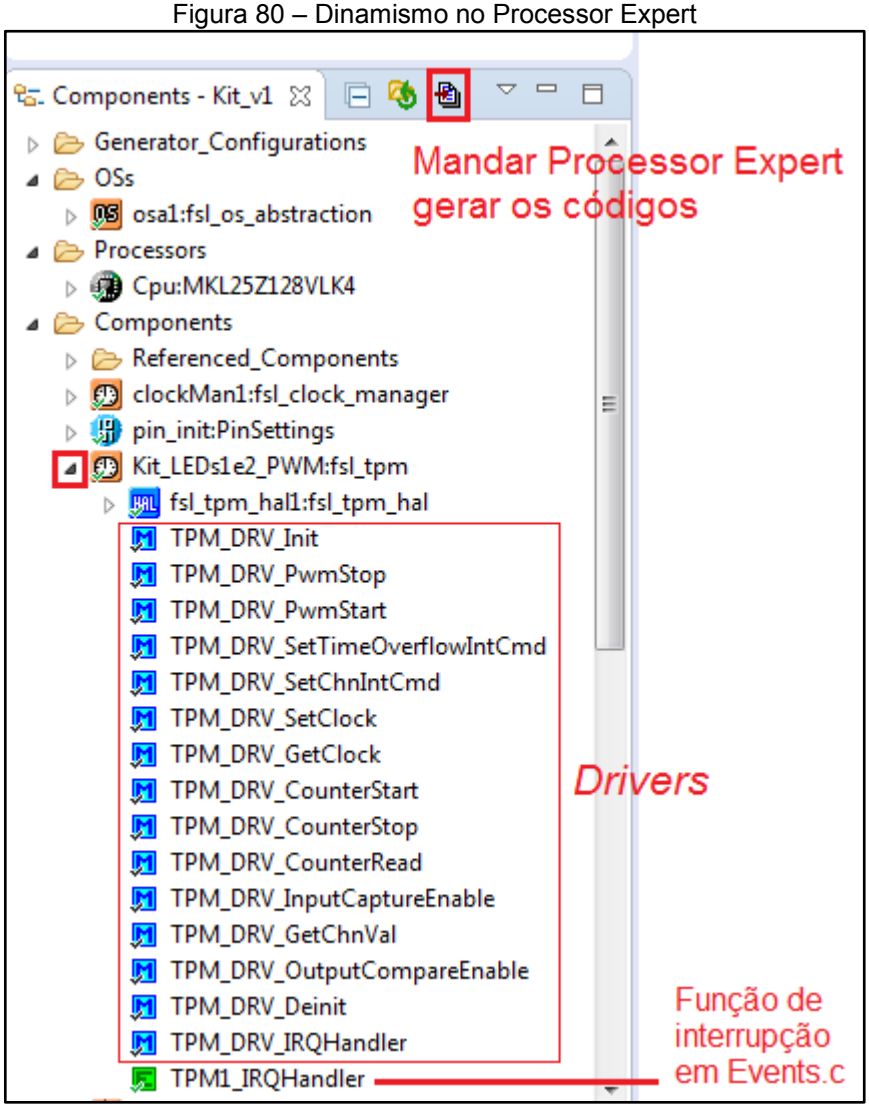
◢ **Initialization**		
◢ **Auto initialization**	Enabled	
Init configuration	Kit_LEDs3e4_PWM_InitConfig0	
Clock source	Internal	48 MHz
Prescaler	Divide by 128	375 kHz
◢ **Initialization type**	PWM	
◢ **Channels**	6	
▷ **Channel 0**	Disabled	
▷ **Channel 1**	Disabled	
◢ **Channel 2**	Enabled	
▷ **Channel mode**	PWM	
▷ **Channel 3**	Disabled	
◢ **Channel 4**	Enabled	
◢ **Channel mode**	PWM	
PWM channel configuration	Kit_LEDs3e4_PWM_ChnConfig0	
▷ **Channel 5**	Disabled	
◢ **Interrupts**		
TPM interrupt	INT_TPM0	INT_TPM0
◢ **Configure interrupt priority**	Enabled	
TPM interrupt priority	minimal priority	192
Install interrupts	Disabled	
◢ **Shared components**		
fsl_interrupt_manager	intMan1	
fsl_clock_manager	clockMan1	
▷ **OS abstraction layer**	Disabled	
◢ **Inherited components**		
fsl_tpm_hal	KSDK 1.3.0/fsl_tpm_hal	

Fonte: Elaborado pelo autor

A frequência que se estabeleceu para ambos TPM1 e TPM0 foi de 6Hz, pois essa é a menor frequência atingível pelo *clock* interno do MCU (48MHz).

Ao se clicar na seta ao lado do ícone do componente é possível observar as funções que estão disponíveis para uso pelo usuário na escrita de seus códigos. Esses são os *drivers* do KSDK. Ao dar um duplo clique em uma dessas funções o usuário é redirecionado ao arquivo que a implementa, podendo entender como usá-la ao ler os comentários deixados pelos funcionários da NXP. Algumas dessas funções já serão usadas automaticamente pelo Processor Expert quando se apertar no ícone de geração dos códigos após adicionar o componente. A Figura 80 elucida estas discussões:

Figura 80 – Dinamismo no Processor Expert



Fonte: Elaborado pelo autor

Se o projeto for compilado e gravado na FRDM-KL25Z após o Processor Expert ter gerado os códigos dos LEDs, será possível observá-los piscando em 6Hz na IU.

2. Pinos de entrada (botões joystick)

Adicione cinco componentes “fsl_gpio” e configure todos da maneira ilustrada pela Figura 81, alterando apenas o pino para cada botão (recorra à Tabela 8).

Figura 81 – Configuração modelo para botões *joystick*

Name	Value	Details
Component name	Joystick1	
Component version	1.3.0	
Input pins	Enabled	
Input configurations	1	
Input configuration 0	Enabled	
Configuration name	Joystick1_InpConfig0	
Input pins number	1	
Pin 0	Enabled	
Pin	J9_9	PTE2/SPI1_SCK
Pin name	J9_9	
Interrupt/DMA	Interrupt/DMA request is disabled.	
Electrical features		
Pull enable	Disabled	
Passive filter	Disabled	
Output pins	Disabled	
Initialization		
Auto initialization	Enabled	
Input pins	Enabled	
Init configuration	Joystick1_InpConfig0	
Output pins	Disabled	
Interrupts		
Shared components		
Inherited components		

Fonte: Elaborado pelo autor

Mande o Processor Expert gerar os códigos e compile o projeto.

3. Pinos de interrupção externa (botões de interrupção)

Crie mais dois componentes “fsl_gpio”. O primeiro configure como na Figura 82:

Figura 82 – Configuração das propriedades do primeiro pino de interrupção

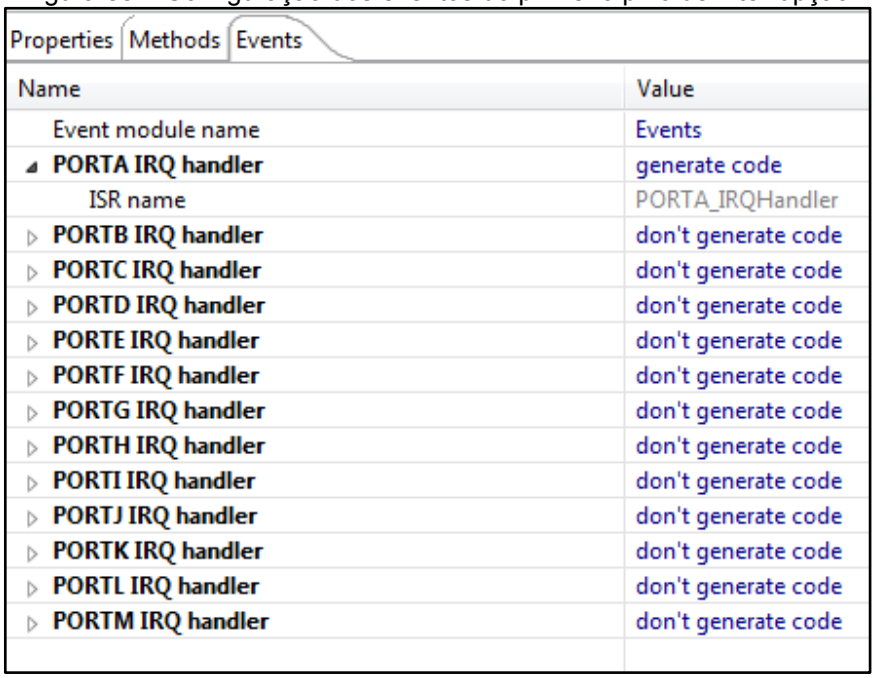
Properties | Methods | Events

Name	Value	Details
Component name	Int1	
Component version	1.3.0	
Input pins	Enabled	
Input configurations	1	
Input configuration 0	Enabled	
Configuration name	Int1_InpConfig0	
Input pins number	1	
Pin 0	Enabled	
Pin	J2_9	PTA16/SPI0_MOSI/SPI0_MISO
Pin name	J2_9	
Interrupt/DMA	Interrupt on falling edge.	
Electrical features		
Pull enable	Disabled	
Passive filter	Disabled	
Output pins	Disabled	
Initialization		
Auto initialization	Enabled	
Input pins	Enabled	
Init configuration	Int1_InpConfig0	
Output pins	Disabled	
Interrupts		
PORTA Interrupt	Enabled	
Device	PORTA	PORTA
Interrupt	INT_PORTA	INT_PORTA
Interrupt priority	Enabled	
Priority value	high priority	64
Install interrupts	Disabled	
PORTB Interrupt	Disabled	
PORTC Interrupt	Disabled	
PORTD Interrupt	Disabled	
PORTE Interrupt	Disabled	
PORTF Interrupt	Disabled	
PORTG Interrupt	Disabled	
PORTH Interrupt	Disabled	
PORTI Interrupt	Disabled	
PORTJ Interrupt	Disabled	
PORTK Interrupt	Disabled	
PORTL Interrupt	Disabled	
PORTM Interrupt	Disabled	
Shared components		
Inherited components		

Fonte: Elaborado pelo autor

Troque para a aba “Events” dentro do componente e peça para o Processor Expert gerar o código de interrupção da Porta A, como na Figura 83:

Figura 83 – Configuração dos eventos do primeiro pino de interrupção



Fonte: Elaborado pelo autor

O próximo componente, o segundo pino de interrupção pela porta A, não precisa declarar necessidade de gerar código de interrupção, pois o primeiro pino já fez essa requisição por todos os pinos da porta A. A Figura 84 ilustra as configurações do segundo pino de interrupção:

Figura 84 – Configuração do segundo pino de interrupção

Name	Value	Details
Component name	Int2	
Component version	1.3.0	
Input pins	Enabled	
Input configurations	1	
Input configuration 0	Enabled	
Configuration name	Int2_InpConfig0	
Input pins number	1	
Pin 0	Enabled	
Pin	J2_11	PTA17/SPI0_MISO/SPI0_MOSI
Pin name	J2_11	
Interrupt/DMA	Interrupt on falling edge.	
Electrical features		
Pull enable	Disabled	
Passive filter	Disabled	
Output pins	Disabled	
Initialization		
Auto initialization	Enabled	
Input pins	Enabled	
Init configuration	Int2_InpConfig0	
Output pins	Disabled	
Interrupts		
PORTA Interrupt	Disabled	
PORTB Interrupt	Disabled	
PORTC Interrupt	Disabled	
PORTD Interrupt	Disabled	
PORTE Interrupt	Disabled	
PORTF Interrupt	Disabled	
PORTG Interrupt	Disabled	
PORTH Interrupt	Disabled	
PORTI Interrupt	Disabled	
PORTJ Interrupt	Disabled	
PORTK Interrupt	Disabled	
PORTL Interrupt	Disabled	
PORTM Interrupt	Disabled	
Shared components		
fsl_interrupt_manager	intMan1	
fsl_clock_manager	clockMan1	
OS abstraction layer	Disabled	
Inherited components		
fsl_gpio_hal	KSDK 1.3.0/fsl_gpio_hal	
fsl_port_hal	KSDK 1.3.0/fsl_port_hal	

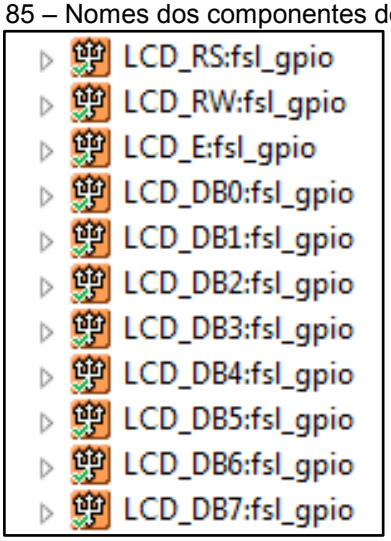
Fonte: Elaborado pelo autor

Mande o Processor Expert gerar os códigos e compile o projeto.

4. Pinos de saída (LCD)

Crie onze componentes do tipo “fsl_gpio” e os nomeie como na Figura 85:

Figura 85 – Nomes dos componentes do LCD



Fonte: Elaborado pelo autor

Agora configure todos esses componentes conforme o modelo da Figura 86 (recorra à Tabela 8 para saber os pinos):

Figura 86 – Configuração modelo para pinos de saída

Name	Value	Details
Component name	LCD_RS	
Component version	1.3.0	
Input pins	Disabled	
Output pins	Enabled	
Output configurations	1	
Output configuration 0	Enabled	
Configuration name	LCD_RS_OutConfig0	
Output pins number	1	
Pin 0	Enabled	
Pin	J1_3	ADC0_SE14/TSI0_CH13/PTC0/EXTRG_IN/CMP0_OUT
Pin name	J1_3	
Output logic	0 D	
Electrical features		
Slew rate	Slow	
Drive strength	Low	
Initialization		
Auto initialization	Enabled	
Input pins	Disabled	
Output pins	Enabled	
Init configuration	LCD_RS_OutConfig0	
Interrupts		
Shared components		
Inherited components		

Fonte: Elaborado pelo autor

Agora altere a configuração “Output logic” para 1 apenas nos componentes DB3, DB4 e DB5.

Mande o Processor Expert gerar os códigos e compile o projeto.

5. Canais ADCs

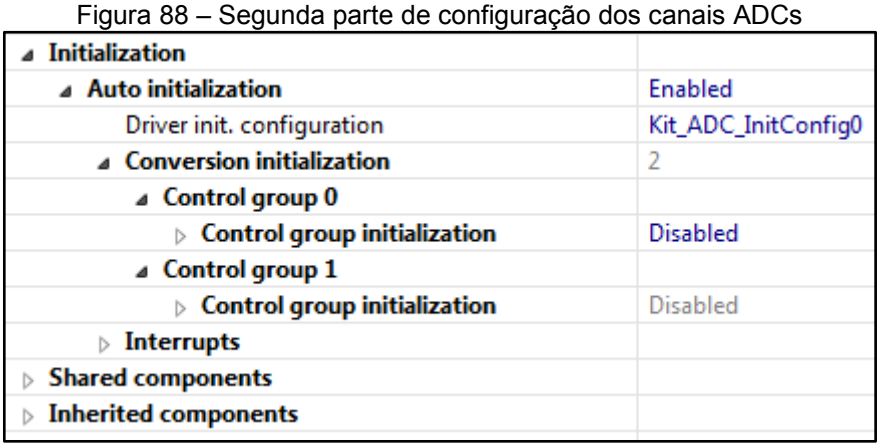
Crie um componente do tipo “fsl_adc16” e configure-o conforme as Figuras 87 e 88:

Figura 87 – Primeira parte de configuração dos canais ADCs

Name	Value	Details
Component name	Kit_ADC	
Device	ADC0	ADC0
Component version	1.3.0	
Configurations	Enabled	
Channel configurations	Enabled	
Configurations list	2	
Configuration 0	Enabled	
Name	Kit_ADC_ChnConfig0	
Type	adc16_chn_config_t	
Read only configuration	Enabled	
Interrupt	Disabled	
Differential mode	Disabled	
A/D channel (pin)	J10_2	ADC0_SE8/TSI0_CH0/PTB0/LLWU_P5/I2C0_SCL/TPM1_CH0
Configuration 1	Enabled	
Name	Kit_ADC_ChnConfig1	
Type	adc16_chn_config_t	
Read only configuration	Enabled	
Interrupt	Disabled	
Differential mode	Disabled	
A/D channel (pin)	J10_12	ADC0_SE15/TSI0_CH14/PTC1/LLWU_P6/RTC_CLKIN/I2C1_SCL/TPM0_CH0
ADC configurations	Enabled	
Configurations list	1	
Configuration 0	Enabled	
Name	Kit_ADC_InitConfig0	
Type	adc16_converter_config_t	
Read only	Enabled	
Low power mode	Disabled	
Clock divider	8	
Long sample time	Enabled	
Resolution	16-bit	
Clock source	Bus clock	
Internal async. clock	Disabled	
High speed mode	Disabled	
Long sample mode	20 extra ADCK cycles, 24 ADCK cyc...	
Hardware trigger	Disabled	
Voltage reference	Vref pair	
Continuous mode	Disabled	
DMA mode	Disabled	
HW compare configurations	Disabled	
ADC PGA configurations	Disabled	
Pins		
Trigger A	Disabled	
Trigger B	Disabled	

Fonte: Elaborado pelo autor

Figura 88 – Segunda parte de configuração dos canais ADCs



Fonte: Elaborado pelo autor

Mande o Processor Expert gerar os códigos e compile o projeto.

6. Componentes para a DAC

Para que se possa externar a tensão da DAC a cada certo período de tempo é importante que se crie um componente do tipo “fsl_pit”. O acrônimo PIT em português significa “temporizador de interrupção periódica”. Configure-o conforme a Figura 89:

Figura 89 – Configuração do módulo PIT

Name	Value	Details
Component name	Timer_DAC	
Device	PIT	PIT
Counter	PIT_CVAL0	PIT_CVAL0
Counter type	Down counter	
Component version	1.3.0	
Configurations	Enabled	
PIT configurations	Enabled	
Configurations list	1	
Configuration 0	Enabled	
Name	Timer_DAC_InitConfig0	
Type	pit_user_config_t	
Read only	Enabled	
Interrupt	Enabled	
Period	200 ms	Clock cfg. 4: 200 ms
Initialization		
Auto initialization	Enabled	
Driver init. configuration	Timer_DAC_InitConfig0	
Run in debug	Enabled	
Start PIT timer	yes	
Interrupts		
Interrupt	INT_PIT	INT_PIT
Interrupt priority	Enabled	
Priority value	medium priority	64
Install interrupts	Enabled	
Shared components		
Inherited components		

Fonte: Elaborado pelo autor

Agora crie um componente do tipo “fsl_dac”, para o módulo DAC. Configure-o conforme a Figura 90:

Figura 90 – Configuração do módulo DAC

Name	Value	Details
Component name	Kit_DAC	
Device	DAC0	DAC0
Component version	1.3.0	
Configurations	Enabled	
Buffer configurations	Enabled	
Configurations list	1	
Configuration 0	Enabled	
Name	Kit_DAC_BufferInitConfig0	
Type	dac_buffer_config_t	
Read only	Enabled	
Buffer	Disabled	
Trigger mode	HW	
Buffer bottom interrupt	Disabled	
Buffer top interrupt	Disabled	
DMA	Disabled	
Buffer mode	Normal	
Upper limit	0	
Basic configurations	Enabled	
Configurations list	1	
Configuration 0	Enabled	
Name	Kit_DAC_InitConfig0	
Type	dac_converter_config_t	
Read only	Enabled	
Voltage reference	Reference 2	
Low power mode	Disabled	
Pins		
Output pin	Enabled	
Output pin	J10_11	DAC0_OUT/ADC0_SE23/CMP0_IN4/PTE30/TPM0_CH3/TPM_CLKIN1
Trigger pin	Disabled	
Trigger pin		Property is disabled
Initialization		
Auto initialization	Enabled	
Driver init. configuration	Kit_DAC_InitConfig0	
Buffer function	Enabled	
Buffer init. configuration	Kit_DAC_BufferInitConfig0	
Interrupts		
Shared components		
Inherited components		

Fonte: Elaborado pelo autor

Mande o Processor Expert gerar os códigos e compile o projeto.

7. Módulo UART

Crie um componente do tipo “fsl_uart” e configure-o conforme a Figura 91:

Figura 91 – Configuração do módulo UART

Name	Value	Details
Component name	Kit_UART2	
Device	UART2	UART2
Component version	1.3.0	
Component mode	Interrupt mode	
Configurations	Enabled	
UART configurations	Enabled	
Configurations list	1	
Configuration 0	Enabled	
Name	Kit_UART2_InitConfig0	
Type	uart_user_config_t	
Read only	Enabled	
Baud rate	9600 baud	Clock cfg. 4: 9615.385 baud
Parity mode	Disabled	
Stop bits	1	
Bits per char	8	
Pins/Signals		
Receiver	Enabled	
RxD	J10_7	ADC0_DM3/ADC0_SE7a/PTE23/TPM2_CH1/UART2_RX
Transmitter	Enabled	
TxD	J10_5	ADC0_DP3/ADC0_SE3/PTE22/TPM2_CH0/UART2_TX
Initialization		
Auto initialization	Enabled	
Init configuration	Kit_UART2_InitConfig0	
State structure name	Kit_UART2_State	
Rx callback	Disabled	
Tx callback	Disabled	
Interrupts		
Common Rx/Tx interrupt		
Interrupt	INT_UART2	INT_UART2
Interrupt priority	Disabled	
Install interrupt	Disabled	
ISR name	UART2_IRQHandler	
Shared components		
fsl_interrupt_manager	intMan1	
fsl_clock_manager	clockMan1	
OS abstraction layer	Enabled	
fsl_os_abstraction	osa1	
Inherited components		
fsl_uart_hal	KSDK 1.3.0/fsl_uart_hal	

Fonte: Elaborado pelo autor

Mande o Processor Expert gerar os códigos e compile o projeto.

APÊNDICE C – ARQUIVO “MAIN.C” DO SOFTWARE DESENVOLVIDO

```
/* ###################################################################
**     Filename    : main.c
**     Project     : Kit_v1
**     Processor   : MKL25Z128VLK4
**     Version     : Driver 01.01
**     Compiler    : GNU C Compiler
**     Date/Time   : 2016-11-12, 10:30, # CodeGen: 0
**     Abstract    :
**         Main module.
**         This module contains user's application code.
**     Settings    :
**     Contents    :
**         No public methods
**
** ###################################################################*/
/*!
** @file main.c
** @version 01.01
** @brief
**         Main module.
**         This module contains user's application code.
*/
/*!
**  @addtogroup main_module main module documentation
**  @{
*/
/* MODULE main */


/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "clockMan1.h"
#include "pin_init.h"
#include "osa1.h"
#include "Kit_LEDs1e2_PWM.h"
#include "Joystick1.h"
```

```
#include "Kit_LEDs3e4_PWM.h"
#include "Joystick2.h"
#include "Joystick3.h"
#include "Joystick4.h"
#include "Joystick5.h"
#include "Int1.h"
#include "Int2.h"
#include "LCD_RS.h"
#include "LCD_RW.h"
#include "LCD_E.h"
#include "LCD_DB0.h"
#include "LCD_DB1.h"
#include "LCD_DB2.h"
#include "LCD_DB3.h"
#include "LCD_DB4.h"
#include "LCD_DB5.h"
#include "LCD_DB6.h"
#include "LCD_DB7.h"
#include "Kit_ADC.h"
#include "Kit_DAC.h"
#include "Timer_DAC.h"
#include "Kit_UART2.h"
#include "dmaController1.h"
#if CPU_INIT_CONFIG
  #include "Init_Config.h"
#endif


/* User includes (#include below this line is not maintained by Processor Expert) */
// Para funcoes requisitadas para uso do LCD
#include "LCD.h"
// Para funcoes de uso geral, como a funcao de Delay
#include "funcoes_gerais.h"

/*lint -save  -e970 Disable MISRA rule (6.3) checking. */
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
```

```
/* Write your local variable definition here */
// Definicao de variaveis ---------------------------------------------------------------------------

// vez_int1 e vez_int2 sao usados dentro do arquivo "Events.c", ao ocorrer uma interrupcao da Porta A
// Indicam a ocorrencia de interrupcao ao se apertar o botao INT1 ou INT2
extern uint8_t vez_int1;
extern uint8_t vez_int2;

// Variaveis que indicam se os botoes do joystick foram apertados um numero impar ou par de vezes
__IO uint8_t vez_joystick1;
__IO uint8_t vez_joystick2;
__IO uint8_t vez_joystick3;
__IO uint8_t vez_joystick4;
__IO uint8_t vez_joystick5;

// Variaveis para leitura e redefinicao dos clocks dos LEDs (LEDs 1 e 2 = TPM1 ; LEDs 2 e 3 = TPM0)
uint32_t clock_TPM0;
uint32_t clock_TPM1;

// Contador usado dentro do loop principal
uint32_t cntdr_loopMain;

// Variaveis usadas na leitura dos ADCs
uint32_t valorADC0_8;
char ADC0_8Digito1, ADC0_8Digito2, ADC0_8Digito3, ADC0_8Digito4;
uint32_t valorADC0_15;
char ADC0_15Digito1, ADC0_15Digito2, ADC0_15Digito3, ADC0_15Digito4;

// Variavel para leitura de algum comando recebido via UART2
uint8_t dadoRxUART2 = 0;

// Fim da definicao de variaveis --------------------------------------------------------------------


// Inicializacao de componentes e variaveis antes de entrar no loop principal ----------------------------------

// Delay inicial para estabilizacao dos componentes
Delay(4800000);
```

```
// Inicializacao de todos os drivers (componentes adicionados e configurados no Process Expert)
// A partir deste ponto, todos os timers/PWM, interrupcoes, etc. passam a funcionar
// Assim, os LEDs passam piscar, LCD liga e botoes comecam a ser interpraveis
/*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
PE_low_level_init();
/*** End of Processor Expert internal initialization.                    ***/

/* Write your code here */
/* For example: for(;;) { } */

// Delay apos inicializacao dos componentes
Delay(700000);

// Inicializacao de algumas variaveis
vez_joystick1 = 0;
vez_joystick2 = 0;
vez_joystick3 = 0;
vez_joystick4 = 0;
vez_joystick5 = 0;
cntdr_loopMain = 0;

// Inicializacao do LCD antes de entrar no loop principal
LCD_inicializar();

LCD_escrever('K');
LCD_escrever('i');
LCD_escrever('t');
LCD_escrever(' ');
LCD_escrever('F');
LCD_escrever('R');
LCD_escrever('D');
LCD_escrever('M');
LCD_escrever('-');
LCD_escrever('K');
LCD_escrever('L');
LCD_escrever('2');
LCD_escrever('5');
LCD_escrever('Z');

LCD_irInicio2aLinha();
```

```
LCD_escrever('B');
LCD_escrever('e');
LCD_escrever('m');
LCD_escrever(' ');
LCD_escrever('V');
LCD_escrever('i');
LCD_escrever('n');
LCD_escrever('d');
LCD_escrever('o');
LCD_escrever('!');

// Delay para dar tempo de visualizar no LCD as boas vindas
Delay(12000000);

// Limpa a tela do LCD para mostrar logo em seguida os valores dos ADCs
LCD_limparDisplay();

// Fim da inicializacao ---------------------------------------------------------------------------------------


// Loop principal do main (programa roda dentro deste "for" para sempre) ---------------------------------------

for(;;) {
    // Botoes de Interrupcao ----------------------------------------------------------------------------------
    // Toda vez que uma interrupcao por botao ocorre, o programa interrompe o que quer
    // que esteja sendo feito dentro do loop principal e "pula" para o arquivo "Events.c",
    // executa a funcao de interrupcao (neste caso PORTA_IRQHandler) e volta para o que estava fazendo previamente.
    // As variaveis vez_int1 e vez_int2 sao atualizadas dentro da funcao PORTA_IRQHandler.

    // Botao 1 de Interrupcao apertado?
    if(vez_int1 == 1) {
        vez_int1 = 0;
        clock_TPM1 = Kit_LEDs1e2_PWM_ChnConfig0.uFrequencyHZ;
        clock_TPM0 = Kit_LEDs3e4_PWM_ChnConfig0.uFrequencyHZ;

        if(clock_TPM1 >= 10) {
            clock_TPM1 = clock_TPM1 - 4U;
        }

        if(clock_TPM0 >= 10) {
            clock_TPM0 = clock_TPM0 - 4U;
        }
```

```
    TPM_DRV_PwmStop(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 0U);
    TPM_DRV_PwmStop(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 1U);
    TPM_DRV_PwmStop(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 2U);
    TPM_DRV_PwmStop(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 4U);

    Kit_LEDs1e2_PWM_ChnConfig0.uFrequencyHZ = clock_TPM1;
    Kit_LEDs3e4_PWM_ChnConfig0.uFrequencyHZ = clock_TPM0;

    TPM_DRV_PwmStart(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 0U);
    TPM_DRV_PwmStart(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 1U);
    TPM_DRV_PwmStart(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 2U);
    TPM_DRV_PwmStart(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 4U);
}

// Botao 2 de Interrupcao apertado?
if(vez_int2 == 1) {
    vez_int2 = 0;

    clock_TPM1 = Kit_LEDs1e2_PWM_ChnConfig0.uFrequencyHZ;
    clock_TPM0 = Kit_LEDs3e4_PWM_ChnConfig0.uFrequencyHZ;

    if(clock_TPM1 <= 200U) {
        clock_TPM1 = clock_TPM1 + 4U;
    }

    if(clock_TPM0 <= 200U) {
        clock_TPM0 = clock_TPM0 + 4U;
    }

    TPM_DRV_PwmStop(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 0U);
    TPM_DRV_PwmStop(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 1U);
    TPM_DRV_PwmStop(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 2U);
    TPM_DRV_PwmStop(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 4U);

    Kit_LEDs1e2_PWM_ChnConfig0.uFrequencyHZ = clock_TPM1;
    Kit_LEDs3e4_PWM_ChnConfig0.uFrequencyHZ = clock_TPM0;

    TPM_DRV_PwmStart(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 0U);
    TPM_DRV_PwmStart(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 1U);
    TPM_DRV_PwmStart(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 2U);
    TPM_DRV_PwmStart(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 4U);
}
// Fim dos Botoes de Interrupcao ----------------------------------------------------------------------
```

```
// Botoes do Joystick e LEDs --------------------------------------------------------------------
// Faz a leitura do nivel de tensao nos pinos dos botoes e, caso algum botao apertado,
// executa comandos relativos aos PWMs dos LEDs

// Botao 1 do Joystick apertado?
if(GPIO_DRV_ReadPinInput(J9_9) == 0) {
    // Faz Debounce de "entrada"
    Delay(480000);

    // Apertado vez impar o par? (impar -> vez_joystick1=1, par -> vez_joystick1=0)
    vez_joystick1 = 1 - vez_joystick1;

    if(vez_joystick1 == 1) {
        // Para a instrucao de stop ser apenas apenas no canal abaixo especificado, eh necessario comentar a linha
        // "TPM_HAL_SetClockMode(tpmBase, kTpmClockSourceNoneClk);" dentro da funcao "TPM_DRV_PwmStop" no arquivo "fsl_tpm_driver.c"
        // Kit_LEDs1e2_PWM_IDX = TPM1 ; 0U = Canal 0
        TPM_DRV_PwmStop(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 0U);
    }
    else {
        TPM_DRV_PwmStart(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 0U);
    }

    // Espera soltar botao (caso ainda esteja sendo pressionado pelo usuario) para sair da condicao do botao ter sido apertado
    if(GPIO_DRV_ReadPinInput(J9_9) == 0) {
        while(GPIO_DRV_ReadPinInput(J9_9) == 0);

        // Faz Debounce de "saida"
        Delay(480000);
    }
}

// Botao 2 do Joystick apertado?
if(GPIO_DRV_ReadPinInput(J9_11) == 0) {
    Delay(480000);

    vez_joystick2 = 1 - vez_joystick2;
```

```
    if(vez_joystick2 == 1) {
        // Kit_LEDs1e2_PWM_IDX = TPM1 ; 1U = Canal 1
        TPM_DRV_PwmStop(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 1U);
    }
    else {
        TPM_DRV_PwmStart(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 1U);
    }

    if(GPIO_DRV_ReadPinInput(J9_11) == 0) {
        while(GPIO_DRV_ReadPinInput(J9_11) == 0);

        Delay(480000);
    }
}

// Botao 3 do Joystick apertado?
if(GPIO_DRV_ReadPinInput(J9_13) == 0) {
    Delay(480000);

    vez_joystick3 = 1 - vez_joystick3;

    if(vez_joystick3 == 1) {
        // Kit_LEDs3e4_PWM_IDX = TPM0 ; 2U = Canal 2
        TPM_DRV_PwmStop(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 2U);
    }
    else {
        TPM_DRV_PwmStart(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 2U);
    }

    if(GPIO_DRV_ReadPinInput(J9_13) == 0) {
        while(GPIO_DRV_ReadPinInput(J9_13) == 0);

        Delay(480000);
    }
}

// Botao 4 do Joystick apertado?
if(GPIO_DRV_ReadPinInput(J9_15) == 0) {
    Delay(480000);

    vez_joystick4 = 1 - vez_joystick4;
```

```
    if(vez_joystick4 == 1) {
        // Kit_LEDs3e4_PWM_IDX = TPM0 ; 4U = Canal 4
        TPM_DRV_PwmStop(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 4U);
    }
    else {
        TPM_DRV_PwmStart(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 4U);
    }

    if(GPIO_DRV_ReadPinInput(J9_15) == 0) {
        while(GPIO_DRV_ReadPinInput(J9_15) == 0);

        Delay(480000);
    }
}

// Botao 5 do Joystick apertado?
if(GPIO_DRV_ReadPinInput(J9_3) == 0) {
    Delay(480000);

    // Por enquanto nao se usa vez_joystick5 para nada
    vez_joystick5 = 1 - vez_joystick5;

    if((vez_joystick1 == 0) || (vez_joystick2 == 0) || (vez_joystick3 == 0) || (vez_joystick4 == 0)) {
        vez_joystick1 = 1;
        vez_joystick2 = 1;
        vez_joystick3 = 1;
        vez_joystick4 = 1;
        TPM_DRV_PwmStop(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 0U);
        TPM_DRV_PwmStop(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 1U);
        TPM_DRV_PwmStop(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 2U);
        TPM_DRV_PwmStop(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 4U);
    }
    else {
        vez_joystick1 = 0;
        vez_joystick2 = 0;
        vez_joystick3 = 0;
        vez_joystick4 = 0;
        TPM_DRV_PwmStart(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 0U);
        TPM_DRV_PwmStart(Kit_LEDs1e2_PWM_IDX, &Kit_LEDs1e2_PWM_ChnConfig0, 1U);
        TPM_DRV_PwmStart(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 2U);
        TPM_DRV_PwmStart(Kit_LEDs3e4_PWM_IDX, &Kit_LEDs3e4_PWM_ChnConfig0, 4U);
    }
```

```
    if(GPIO_DRV_ReadPinInput(J9_3) == 0) {
        while(GPIO_DRV_ReadPinInput(J9_3) == 0);

        Delay(480000);
    }
}

// Fim dos Botoes do Joystick e LEDs ------------------------------------------------------------------


// Leitura dos ADCs e atualizacao no LCD --------------------------------------------------------------

// Apos 48000 voltas no loop, atualiza valores dos ADCs no LCD
cntdr_loopMain++;
if(cntdr_loopMain >= 48000) {
    cntdr_loopMain = 0;

    // Inicializa a conversao AD do canal 8
    ADC16_DRV_ConfigConvChn(Kit_ADC_IDX, 0U, &Kit_ADC_ChnConfig0);
    // Espera conversao AD terminar
    ADC16_DRV_WaitConvDone(Kit_ADC_IDX, 0U);
    // Busca o valor AD convertido
    valorADC0_8 = ADC16_DRV_GetConvValueRAW(Kit_ADC_IDX, 0U);
    // Pausa a conversao
    ADC16_DRV_PauseConv(Kit_ADC_IDX, 0U);

    // Inicializa a conversao AD do canal 15
    ADC16_DRV_ConfigConvChn(Kit_ADC_IDX, 0U, &Kit_ADC_ChnConfig1);
    // Espera conversao AD terminar
    ADC16_DRV_WaitConvDone(Kit_ADC_IDX, 0U);
    // Busca o valor AD convertido
    valorADC0_15 = ADC16_DRV_GetConvValueRAW(Kit_ADC_IDX, 0U);
    // Pausa a conversao
    ADC16_DRV_PauseConv(Kit_ADC_IDX, 0U);

    // Converte o valor discreto lido pelos ADCs (16bits) em mV;
    valorADC0_8 = (uint32_t)(valorADC0_8 * 0.050354);
    valorADC0_15 = (uint32_t)(valorADC0_15 * 0.050354);
```

```
// Separa em unidade, dezena, centena e milhar
ADC0_8Digito1 = valorADC0_8/1000;
ADC0_8Digito2 = (valorADC0_8-(ADC0_8Digito1*1000))/100 ;
ADC0_8Digito3 = (valorADC0_8-((ADC0_8Digito1*1000)+(ADC0_8Digito2*100)))/10;
ADC0_8Digito4 = valorADC0_8 -((ADC0_8Digito1*1000)+(ADC0_8Digito2*100)+ (ADC0_8Digito3*10));
ADC0_15Digito1 = valorADC0_15/1000;
ADC0_15Digito2 = (valorADC0_15-(ADC0_15Digito1*1000))/100 ;
ADC0_15Digito3 = (valorADC0_15-((ADC0_15Digito1*1000)+(ADC0_15Digito2*100)))/10;
ADC0_15Digito4 = valorADC0_15 -((ADC0_15Digito1*1000)+(ADC0_15Digito2*100)+ (ADC0_15Digito3*10));

// Atualiza LCD
// Obs: O valor de cada Digito eh somado de 0x30 (hexa),
// pois em ASCII 0x30 equivale ao caractere 0 e 0x39 equivale a 9
LCD_irInicio1aLinha();

LCD_escrever('A');
LCD_escrever('D');
LCD_escrever('C');
LCD_escrever('0');
LCD_escrever('_');
LCD_escrever('8');
LCD_escrever(' ');
LCD_escrever(' ');
LCD_escrever('=');
LCD_escrever(' ');
LCD_escrever((char)(ADC0_8Digito1+0x30));
LCD_escrever((char)(ADC0_8Digito2+0x30));
LCD_escrever((char)(ADC0_8Digito3+0x30));
LCD_escrever((char)(ADC0_8Digito4+0x30));
LCD_escrever('m');
LCD_escrever('V');

LCD_irInicio2aLinha();

LCD_escrever('A');
LCD_escrever('D');
LCD_escrever('C');
LCD_escrever('0');
LCD_escrever('_');
LCD_escrever('1');
LCD_escrever('5');
LCD_escrever(' ');
LCD_escrever('=');
```

```
        LCD_escrever(' ');
        LCD_escrever((char)(ADC0_15Digito1+0x30));
        LCD_escrever((char)(ADC0_15Digito2+0x30));
        LCD_escrever((char)(ADC0_15Digito3+0x30));
        LCD_escrever((char)(ADC0_15Digito4+0x30));
        LCD_escrever('m');
        LCD_escrever('V');
    }

    // Fim da leitura dos ADCs e atualizacao no LCD ----------------------------------------------------


    // UART (Serial)  ----------------------------------------------------------------------------------

    // Recebe dado (mesmo que nada tenha sido transmitido)
    UART_DRV_ReceiveData(Kit_UART2_IDX, &dadoRxUART2, 1u);

    // Checa se o dado recebido foi valido, e mostra por alguns instantes* o dado no LCD
    // * A parte dos ADCs se encarrega de mostrar novamente os valores no LCS, apos certas voltas no loop do main
    if(dadoRxUART2 != 0) {
        Delay(480000);

        LCD_limparDisplay();
        LCD_escrever(dadoRxUART2);

        Delay(4800000);
        dadoRxUART2 = 0;
    }

    // Fim da UART -------------------------------------------------------------------------------------

    // Fim do loop do main -----------------------------------------------------------------------------
}
```

```
    /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY THIS CODE!!! ***/
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is defined by the RTOS component. */
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

/* END main */
/*!
** @}
*/
/*
** ###################################################################
**
**     This file was created by Processor Expert 10.5 [05.21]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/
```

APÊNDICE D – ARQUIVO “EVENTS.C” DO SOFTWARE DESENVOLVIDO

```
/* ###################################################################
**     Filename    : Events.c
**     Project     : Kit_v1
**     Processor   : MKL25Z128VLK4
**     Component   : Events
**     Version     : Driver 01.00
**     Compiler    : GNU C Compiler
**     Date/Time   : 2016-11-12, 10:30, # CodeGen: 0
**     Abstract    :
**         This is user's event module.
**         Put your event handler code here.
**     Settings    :
**     Contents    :
**         No public methods
**
** ###################################################################*/
/*!
** @file Events.c
** @version 01.00
** @brief
**         This is user's event module.
**         Put your event handler code here.
*/
/*!
**  @addtogroup Events_module Events module documentation
**  @{
*/
/* MODULE Events */

#include "Cpu.h"
#include "Events.h"

#ifdef __cplusplus
extern "C" {
#endif
```

```
/* User includes (#include below this line is not maintained by Processor Expert) */
#include "LCD.h"
uint8_t vez_int1 = 0;
uint8_t vez_int2 = 0;

#ifdef Kit_LEDs1e2_PWM_IDX
/*
** ===================================================================
**     Interrupt handler : TPM1_IRQHandler
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void TPM1_IRQHandler(void)
{
  TPM_DRV_IRQHandler(Kit_LEDs1e2_PWM_IDX);
  /* Write your code here ... */
}
#else
  /* This IRQ handler is not used by Kit_LEDs1e2_PWM component. The purpose may be
   * that the component has been removed or disabled. It is recommended to
   * remove this handler because Processor Expert cannot modify it according to
   * possible new request (e.g. in case that another component uses this
   * interrupt vector). */
  #warning This IRQ handler is not used by Kit_LEDs1e2_PWM component.\
           It is recommended to remove this because Processor Expert cannot\
           modify it according to possible new request.
#endif

#ifdef Kit_LEDs3e4_PWM_IDX
/*
** ===================================================================
**     Interrupt handler : TPM0_IRQHandler
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
```

```
void TPM0_IRQHandler(void)
{
  TPM_DRV_IRQHandler(Kit_LEDs3e4_PWM_IDX);
  /* Write your code here ... */
}
#else
  /* This IRQ handler is not used by Kit_LEDs3e4_PWM component. The purpose may be
   * that the component has been removed or disabled. It is recommended to
   * remove this handler because Processor Expert cannot modify it according to
   * possible new request (e.g. in case that another component uses this
   * interrupt vector). */
  #warning This IRQ handler is not used by Kit_LEDs3e4_PWM component.\
           It is recommended to remove this because Processor Expert cannot\
           modify it according to possible new request.
#endif

/*
** ===================================================================
**     Interrupt handler : PORTA_IRQHandler
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void PORTA_IRQHandler(void)
{
    // Limpa interrupcoes pendentes da Porta A
    NVIC_ClearPendingIRQ(PORTA_IRQn);

    // Checa se foi Int1 o pino que gerou a interrupcao (Int1 = PA16).
    // Pino 16 correponde ao 17o bit da flag (comeca em 0), que em em hexa eh 0x10000.
    if((PORT_HAL_GetPortIntFlag(PORTA_BASE_PTR)) & 0x10000u) {
        vez_int1 = 1;
    }

    // Checa se foi Int2 o pino que gerou a interrupcao (Int2 = PA17).
    // Pino 17 correponde ao 18o bit da flag (comeca em 0), que em em hexa eh 0x20000.
    if((PORT_HAL_GetPortIntFlag(PORTA_BASE_PTR)) & 0x20000u) {
        vez_int2 = 1;
    }
```

```
    // Limpa todas as flags de interrupcao da Porta A
    PORT_HAL_ClearPortIntFlag(PORTA_BASE_PTR);
}

/*
** ===================================================================
**     Interrupt handler : DAC0_IRQHandler
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void DAC0_IRQHandler(void)
{
  /* Write your code here ... */
}

/*
** ===================================================================
**     Interrupt handler : ADC0_IRQHandler
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void ADC0_IRQHandler(void)
{
  /* Write your code here ... */
}

#ifdef Timer_DAC_IDX
```

```
// Valores para formar uma senoide pela DAC (120 valores)
#define DAC_BUFF_SIZE (120U)
uint16_t dac_buffer[DAC_BUFF_SIZE] =
{
        0x7ff, 0x86a, 0x8d5, 0x93f, 0x9a9, 0xa11, 0xa78, 0xadd, 0xb40, 0xba1,
        0xbff, 0xc5a, 0xcb2, 0xd08, 0xd59, 0xda7, 0xdf1, 0xe36, 0xe77, 0xeb4,
        0xeec, 0xf1f, 0xf4d, 0xf77, 0xf9a, 0xfb9, 0xfd2, 0xfe5, 0xff3, 0xffc,
        0xfff, 0xffc, 0xff3, 0xfe5, 0xfd2, 0xfb9, 0xf9a, 0xf77, 0xf4d, 0xf1f,
        0xeec, 0xeb4, 0xe77, 0xe36, 0xdf1, 0xda7, 0xd59, 0xd08, 0xcb2, 0xc5a,
        0xbff, 0xba1, 0xb40, 0xadd, 0xa78, 0xa11, 0x9a9, 0x93f, 0x8d5, 0x86a,
        0x7ff, 0x794, 0x729, 0x6bf, 0x655, 0x5ed, 0x586, 0x521, 0x4be, 0x45d,
        0x3ff, 0x3a4, 0x34c, 0x2f6, 0x2a5, 0x257, 0x20d, 0x1c8, 0x187, 0x14a,
        0x112, 0xdf,  0xb1,  0x87,  0x64,  0x45,  0x2c,  0x19,  0xb,   0x2,
        0x0,   0x2,   0xb,   0x19,  0x2c,  0x45,  0x64,  0x87,  0xb1,  0xdf,
        0x112, 0x14a, 0x187, 0x1c8, 0x20d, 0x257, 0x2a5, 0x2f6, 0x34c, 0x3a4,
        0x3ff, 0x45d, 0x4be, 0x521, 0x586, 0x5ed, 0x655, 0x6bf, 0x729, 0x794
};
uint16_t index = 0;

/*
** ===================================================================
**     Interrupt handler : Timer_DAC_IRQHandler
**
**     Description :
**         User interrupt service routine.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void Timer_DAC_IRQHandler(void)
{
    /* Clear interrupt flag.*/
    PIT_HAL_ClearIntFlag(g_pitBase[Timer_DAC_IDX], Timer_DAC_CHANNEL);
    /* Write your code here ... */

    // Escreve na DAC valor dado pelo index dentro de dac_buffer (forma uma senoide com o tempo)
    DAC_DRV_Output(Kit_DAC_IDX,dac_buffer[index++]);
    // Se index chega ao maximo, comecar do inicio novamente
    if(index == DAC_BUFF_SIZE) {
        index = 0;
    }
}
```

```
#else
  /* This IRQ handler is not used by Timer_DAC component. The purpose may be
   * that the component has been removed or disabled. It is recommended to
   * remove this handler because Processor Expert cannot modify it according to
   * possible new request (e.g. in case that another component uses this
   * interrupt vector). */
  #warning This IRQ handler is not used by Timer_DAC component.\
           It is recommended to remove this because Processor Expert cannot\
           modify it according to possible new request.
#endif

/*
** ===================================================================
**     Callback    : Kit_UART2_RxCallback
**     Description : This callback occurs when data are received.
**     Parameters  :
**       instance - The UART instance number.
**       uartState - A pointer to the UART driver state structure
**       memory.
**     Returns : Nothing
** ===================================================================
*/
void Kit_UART2_RxCallback(uint32_t instance, void * uartState)
{
    /* Write your code here ... */
}

/* END Events */

#ifdef __cplusplus
}  /* extern "C" */
#endif

/*!
** @}
*/
/*
** ###################################################################
**
**     This file was created by Processor Expert 10.5 [05.21]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################*/
```

APÊNDICE E – ARQUIVO “LCD.C” DO SOFTWARE DESENVOLVIDO

```
/*
 * LCD.c
 *
 *  Created on: 15/11/2016
 *      Author: Guilherme
 */

#include "Cpu.h"
#include "funcoes_gerais.h"
#include "LCD.h"

// Inicializacao do LCD
void LCD_inicializar(void) {
    // Configuracao inicial de todos pinos do LCD
    // J1_3 = RS (Register Select)
    GPIO_DRV_WritePinOutput(J1_3, 0);
    // J1_5 = R/W (Write (0) ou Read (1))
    GPIO_DRV_WritePinOutput(J1_5, 0);
    // J1_7 = E (Enable: 1 para habilitar registro dos comandos, 0 para desabilitar)
    GPIO_DRV_WritePinOutput(J1_7, 0);
    // J1_9 = DB0
    GPIO_DRV_WritePinOutput(J1_9, 0);
    // J1_11 = DB1
    GPIO_DRV_WritePinOutput(J1_11, 0);
    // J1_13 = DB2
    GPIO_DRV_WritePinOutput(J1_13, 0);
    // J1_15 = DB3
    GPIO_DRV_WritePinOutput(J1_15, 1);
    // J2_1 = DB4
    GPIO_DRV_WritePinOutput(J2_1, 1);
    // J2_3 = DB5
    GPIO_DRV_WritePinOutput(J2_3, 1);
    // J2_5 = DB6
    GPIO_DRV_WritePinOutput(J2_5, 0);
    // J2_7 = DB7
    GPIO_DRV_WritePinOutput(J2_7, 0);
```

```
    Delay(480000);

    // Setando LCD para operacao em 8-bits, duas linhas e 5x7pontos
    GPIO_DRV_WritePinOutput(J1_15, 1);
    GPIO_DRV_WritePinOutput(J2_1, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    GPIO_DRV_WritePinOutput(J1_7, 1);
    Delay(480000);
    GPIO_DRV_WritePinOutput(J1_7, 0);
    Delay(480000);

    // Display OFF
    GPIO_DRV_WritePinOutput(J2_1, 0);
    GPIO_DRV_WritePinOutput(J2_3, 0);
    GPIO_DRV_WritePinOutput(J1_7, 1);
    Delay(480000);
    GPIO_DRV_WritePinOutput(J1_7, 0);
    Delay(480000);

    // Limpar Display
    GPIO_DRV_WritePinOutput(J1_15, 0);
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_7, 1);
    Delay(480000);
    GPIO_DRV_WritePinOutput(J1_7, 0);
    Delay(480000);

    // Display ON
    GPIO_DRV_WritePinOutput(J1_9, 0);
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J1_15, 1);
    GPIO_DRV_WritePinOutput(J1_7, 1);
    Delay(480000);
    GPIO_DRV_WritePinOutput(J1_7, 0);
    Delay(480000);

    // Entry Mode Set
    GPIO_DRV_WritePinOutput(J1_15, 0);
    GPIO_DRV_WritePinOutput(J1_11, 1);
    GPIO_DRV_WritePinOutput(J1_7, 1);
    Delay(480000);
    GPIO_DRV_WritePinOutput(J1_7, 0);
}
```

```
// Escrita de um caractere no LCD
// Estudantes de Microprocessador I: Completar funcao com os caracteres que faltam
void LCD_escrever(char caractere) {
    // Configurar todos pinos em zero inicialmente
    // J1_3 = RS (Register Select)
    GPIO_DRV_WritePinOutput(J1_3, 0);
    // J1_5 = R/W (Write (0) ou Read (1))
    GPIO_DRV_WritePinOutput(J1_5, 0);
    // J1_7 = E (Enable: 1 para habilitar registro dos comandos, 0 para desabilitar)
    GPIO_DRV_WritePinOutput(J1_7, 0);
    // J1_9 = DB0
    GPIO_DRV_WritePinOutput(J1_9, 0);
    // J1_11 = DB1
    GPIO_DRV_WritePinOutput(J1_11, 0);
    // J1_13 = DB2
    GPIO_DRV_WritePinOutput(J1_13, 0);
    // J1_15 = DB3
    GPIO_DRV_WritePinOutput(J1_15, 0);
    // J2_1 = DB4
    GPIO_DRV_WritePinOutput(J2_1, 0);
    // J2_3 = DB5
    GPIO_DRV_WritePinOutput(J2_3, 0);
    // J2_5 = DB6
    GPIO_DRV_WritePinOutput(J2_5, 0);
    // J2_7 = DB7
    GPIO_DRV_WritePinOutput(J2_7, 0);

    // Ligar pinos correspondentes ao caractere a ser escrito
    switch(caractere) {
    case ' ':
        GPIO_DRV_WritePinOutput(J2_3, 1);
        break;
    case '!':
        GPIO_DRV_WritePinOutput(J1_9, 1);
        GPIO_DRV_WritePinOutput(J2_3, 1);
        break;
    case ',':
        GPIO_DRV_WritePinOutput(J1_13, 1);
        GPIO_DRV_WritePinOutput(J1_15, 1);
        GPIO_DRV_WritePinOutput(J2_3, 1);
        break;
```

```
case '-':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J1_15, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    break;
case '_':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_11, 1);
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J1_15, 1);
    GPIO_DRV_WritePinOutput(J2_1, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
case '0':
    GPIO_DRV_WritePinOutput(J2_1, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    break;
case '1':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J2_1, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    break;
case '2':
    GPIO_DRV_WritePinOutput(J1_11, 1);
    GPIO_DRV_WritePinOutput(J2_1, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    break;
case '3':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_11, 1);
    GPIO_DRV_WritePinOutput(J2_1, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    break;
case '4':
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J2_1, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    break;
```

```
case '5':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J2_1, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    break;
case '6':
    GPIO_DRV_WritePinOutput(J1_11, 1);
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J2_1, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    break;
case '7':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_11, 1);
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J2_1, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    break;
case '8':
    GPIO_DRV_WritePinOutput(J1_15, 1);
    GPIO_DRV_WritePinOutput(J2_1, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    break;
case '9':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_15, 1);
    GPIO_DRV_WritePinOutput(J2_1, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    break;
case '=':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J1_15, 1);
    GPIO_DRV_WritePinOutput(J2_1, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    break;
case 'A':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
```

```
case 'a':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
case 'B':
    GPIO_DRV_WritePinOutput(J1_11, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
case 'C':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_11, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
case 'D':
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
case 'd':
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
case 'e':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
case 'F':
    GPIO_DRV_WritePinOutput(J1_11, 1);
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
case 'i':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_15, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
```

```
case 'K':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_11, 1);
    GPIO_DRV_WritePinOutput(J1_15, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
case 'L':
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J1_15, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
case 'M':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J1_15, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
case 'm':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J1_15, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
case 'n':
    GPIO_DRV_WritePinOutput(J1_11, 1);
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J1_15, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
case 'o':
    GPIO_DRV_WritePinOutput(J1_9, 1);
    GPIO_DRV_WritePinOutput(J1_11, 1);
    GPIO_DRV_WritePinOutput(J1_13, 1);
    GPIO_DRV_WritePinOutput(J1_15, 1);
    GPIO_DRV_WritePinOutput(J2_3, 1);
    GPIO_DRV_WritePinOutput(J2_5, 1);
    break;
```

```
    case 'R':
        GPIO_DRV_WritePinOutput(J1_11, 1);
        GPIO_DRV_WritePinOutput(J2_1, 1);
        GPIO_DRV_WritePinOutput(J2_5, 1);
        break;
    case 'S':
        GPIO_DRV_WritePinOutput(J1_9, 1);
        GPIO_DRV_WritePinOutput(J1_11, 1);
        GPIO_DRV_WritePinOutput(J2_1, 1);
        GPIO_DRV_WritePinOutput(J2_5, 1);
        break;
    case 't':
        GPIO_DRV_WritePinOutput(J1_13, 1);
        GPIO_DRV_WritePinOutput(J2_1, 1);
        GPIO_DRV_WritePinOutput(J2_3, 1);
        GPIO_DRV_WritePinOutput(J2_5, 1);
        break;
    case 'V':
        GPIO_DRV_WritePinOutput(J1_11, 1);
        GPIO_DRV_WritePinOutput(J1_13, 1);
        GPIO_DRV_WritePinOutput(J2_1, 1);
        GPIO_DRV_WritePinOutput(J2_5, 1);
        break;
    case 'Z':
        GPIO_DRV_WritePinOutput(J1_11, 1);
        GPIO_DRV_WritePinOutput(J1_15, 1);
        GPIO_DRV_WritePinOutput(J2_1, 1);
        GPIO_DRV_WritePinOutput(J2_5, 1);
        break;
    }

    // Ligar pino de escrita no registrador do LCD
    GPIO_DRV_WritePinOutput(J1_3, 1);

    // Habilitar escrita de dados, esperar um pouco, e desabilitar
    GPIO_DRV_WritePinOutput(J1_7, 1);
    Delay(4800);
    GPIO_DRV_WritePinOutput(J1_7, 0);
}
```

```
void LCD_irInicio1aLinha(void) {
    // Configurar todos pinos em zero inicialmente
    GPIO_DRV_WritePinOutput(J1_3, 0);
    GPIO_DRV_WritePinOutput(J1_5, 0);
    GPIO_DRV_WritePinOutput(J1_7, 0);
    GPIO_DRV_WritePinOutput(J1_9, 0);
    GPIO_DRV_WritePinOutput(J1_11, 0);
    GPIO_DRV_WritePinOutput(J1_13, 0);
    GPIO_DRV_WritePinOutput(J1_15, 0);
    GPIO_DRV_WritePinOutput(J2_1, 0);
    GPIO_DRV_WritePinOutput(J2_3, 0);
    GPIO_DRV_WritePinOutput(J2_5, 0);
    GPIO_DRV_WritePinOutput(J2_7, 0);

    GPIO_DRV_WritePinOutput(J1_11, 1);

    // Habilitar escrita de dados, esperar um pouco, e desabilitar
    GPIO_DRV_WritePinOutput(J1_7, 1);
    Delay(80000);
    GPIO_DRV_WritePinOutput(J1_7, 0);
}

void LCD_irInicio2aLinha(void) {
    // Configurar todos pinos em zero inicialmente
    GPIO_DRV_WritePinOutput(J1_3, 0);
    GPIO_DRV_WritePinOutput(J1_5, 0);
    GPIO_DRV_WritePinOutput(J1_7, 0);
    GPIO_DRV_WritePinOutput(J1_9, 0);
    GPIO_DRV_WritePinOutput(J1_11, 0);
    GPIO_DRV_WritePinOutput(J1_13, 0);
    GPIO_DRV_WritePinOutput(J1_15, 0);
    GPIO_DRV_WritePinOutput(J2_1, 0);
    GPIO_DRV_WritePinOutput(J2_3, 0);
    GPIO_DRV_WritePinOutput(J2_5, 0);
    GPIO_DRV_WritePinOutput(J2_7, 0);

    GPIO_DRV_WritePinOutput(J2_5, 1);
    GPIO_DRV_WritePinOutput(J2_7, 1);
```

```
    // Habilitar escrita de dados, esperar um pouco, e desabilitar
    GPIO_DRV_WritePinOutput(J1_7, 1);
    Delay(4800);
    GPIO_DRV_WritePinOutput(J1_7, 0);
}

void LCD_recuarCursor(void) {
    // Configurar todos pinos em zero inicialmente
    GPIO_DRV_WritePinOutput(J1_3, 0);
    GPIO_DRV_WritePinOutput(J1_5, 0);
    GPIO_DRV_WritePinOutput(J1_7, 0);
    GPIO_DRV_WritePinOutput(J1_9, 0);
    GPIO_DRV_WritePinOutput(J1_11, 0);
    GPIO_DRV_WritePinOutput(J1_13, 0);
    GPIO_DRV_WritePinOutput(J1_15, 0);
    GPIO_DRV_WritePinOutput(J2_1, 0);
    GPIO_DRV_WritePinOutput(J2_3, 0);
    GPIO_DRV_WritePinOutput(J2_5, 0);
    GPIO_DRV_WritePinOutput(J2_7, 0);

    GPIO_DRV_WritePinOutput(J2_1, 1);

    // Habilitar escrita de dados, esperar um pouco, e desabilitar
    GPIO_DRV_WritePinOutput(J1_7, 1);
    Delay(4800);
    GPIO_DRV_WritePinOutput(J1_7, 0);
}

void LCD_limparDisplay(void) {
    // Configurar todos pinos em zero inicialmente
    GPIO_DRV_WritePinOutput(J1_3, 0);
    GPIO_DRV_WritePinOutput(J1_5, 0);
    GPIO_DRV_WritePinOutput(J1_7, 0);
    GPIO_DRV_WritePinOutput(J1_9, 0);
    GPIO_DRV_WritePinOutput(J1_11, 0);
    GPIO_DRV_WritePinOutput(J1_13, 0);
    GPIO_DRV_WritePinOutput(J1_15, 0);
    GPIO_DRV_WritePinOutput(J2_1, 0);
    GPIO_DRV_WritePinOutput(J2_3, 0);
    GPIO_DRV_WritePinOutput(J2_5, 0);
    GPIO_DRV_WritePinOutput(J2_7, 0);

    GPIO_DRV_WritePinOutput(J1_9, 1);
```

```
    // Habilitar escrita de dados, esperar um pouco, e desabilitar
    GPIO_DRV_WritePinOutput(J1_7, 1);
    Delay(480000);
    GPIO_DRV_WritePinOutput(J1_7, 0);
    Delay(480000);
}
```