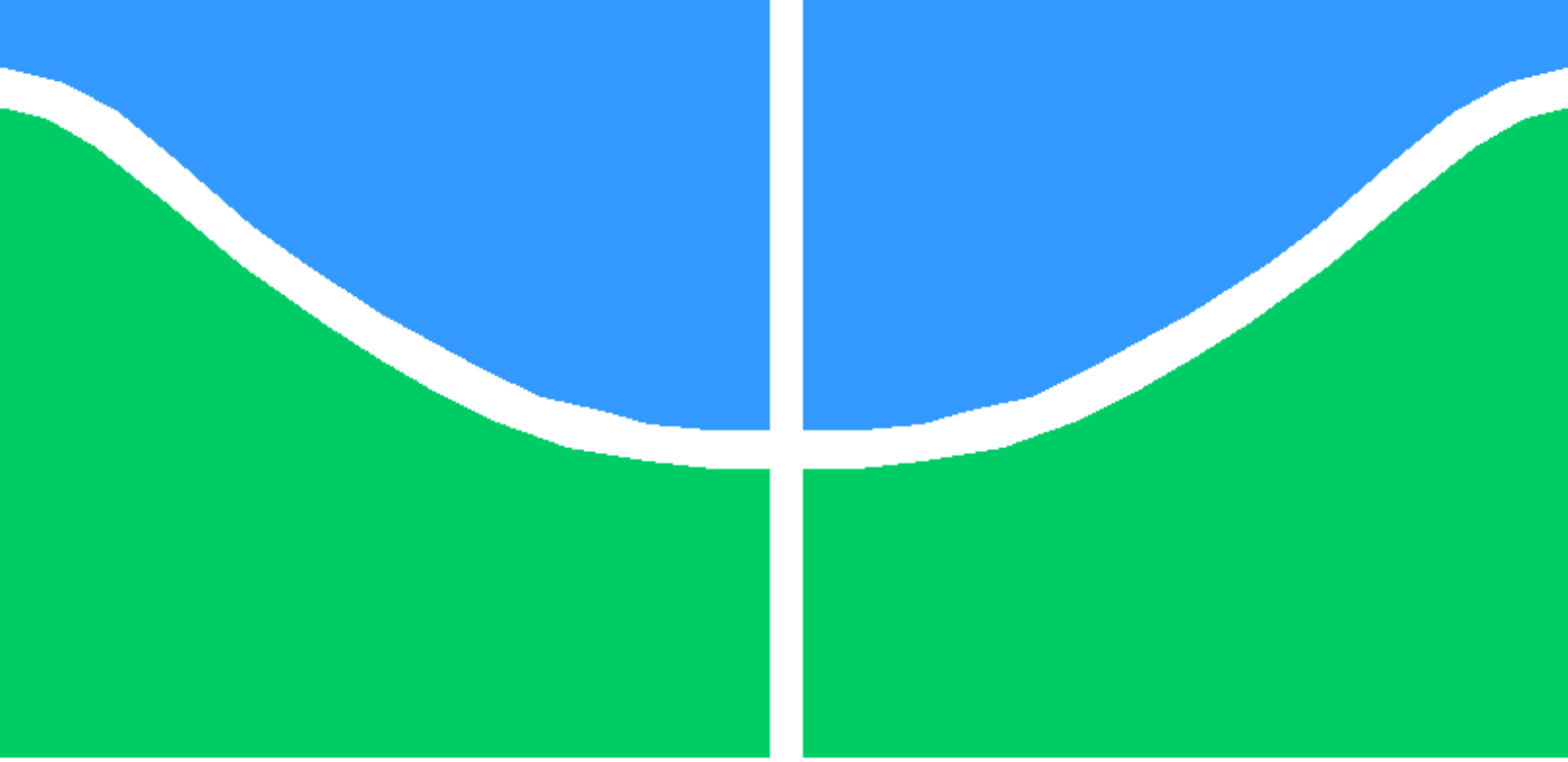


PROCESSAMENTO DIGITAL DE SINAIS DE ÁUDIO COM ELETRÔNICA SENSORIAL EMBARCADA

DANIEL CAVALLARE PIRES

PEDRO MAURÍCIO DE SOUSA

DISSERTAÇÃO DE BACHARELADO EM ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROCESSAMENTO DIGITAL DE SINAIS DE ÁUDIO COM ELETRÔNICA SENSORIAL EMBARCADA

DANIEL CAVALLARE PIRES

PEDRO MAURÍCIO DE SOUSA

Orientador: PROF. DANIEL CHAVES CAFÉ, ENE/UNB

DISSERTAÇÃO DE BACHARELADO EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO ENE - XXXXXXXXXXX/2016
BRASÍLIA-DF, 12 DE DEZEMBRO DE 2016.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROCESSAMENTO DIGITAL DE SINAIS DE ÁUDIO COM ELETRÔNICA SENSORIAL EMBARCADA

DANIEL CAVALLARE PIRES

PEDRO MAURÍCIO DE SOUSA

DISSERTAÇÃO DE BACHARELADO ACADÊMICO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRICISTA EM ENGENHARIA ELÉTRICA.

APROVADA POR:

Prof. DANIEL CHAVES CAFÉ, ENE/UnB
Orientador

Prof. Dr. Ricardo Zelenovsky, ENE/UnB
Examinador interno

Prof. Dr. Alexandre Ricardo Soares Romariz, ENE/UnB
Examinador interno

BRASÍLIA, 12 DE DEZEMBRO DE 2016.

FICHA CATALOGRÁFICA
DANIEL CAVALLARE PIRES
PROCESSAMENTO DIGITAL DE SINAIS DE ÁUDIO COM ELETRÔNICA SENSORIAL EMBARCADA
2016xv, 00000000000000p., 201x297 mm
(ENE/FT/UnB, Engenheiro Eletricista, Engenharia Elétrica, 2016)
Dissertação de Bacharelado - Universidade de Brasília
Faculdade de Tecnologia - Departamento de Engenharia Elétrica

REFERÊNCIA BIBLIOGRÁFICA

DANIEL CAVALLARE PIRES (2016) PROCESSAMENTO DIGITAL DE SINAIS DE ÁUDIO COM ELETRÔNICA SENSORIAL EMBARCADA. Dissertação de Bacharelado em Engenharia Elétrica, Publicação XXXXXXXXXXX/2016, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 00000000000000p.

CESSÃO DE DIREITOS

AUTORES: DANIEL CAVALLARE PIRES; PEDRO MAURÍCIO DE SOUSA.
TÍTULO: PROCESSAMENTO DIGITAL DE SINAIS DE ÁUDIO COM ELETRÔNICA SENSORIAL EMBARCADA.
GRAU: ENGENHEIRO ELETRICISTA ANO: 2016

_____

DANIEL CAVALLARE PIRES
Universidade de Brasília

_____

PEDRO MAURÍCIO DE SOUSA
Universidade de Brasília

Agradecimentos

Aos meus pais, por sempre terem me acompanhado nesses anos de jornada na universidade, nos momentos bons e ruins.

Aos meus amigos, por terem sempre estado ao meu lado quando precisei de algum tempo a mais de conversa ou ao sentar à mesa de um bar e poder encontrá-los de braços abertos.

À minha namorada, que, independentemente da situação pela qual eu estava passando, sempre me apoiou e me encorajou a continuar sabendo que tudo vai ficar certo.

Agradeço especialmente ao pessoal do Movi, encabeçados pelo Dhyan Kapish. Vocês me proporcionaram experiências inenarráveis, me guiaram a me conhecer, me possibilitaram seguir de cabeça erguida e me ensinaram que o mais valioso é o respeito - não só aos outros, mas principalmente a mim mesmo.

E a todos os anos na Universidade de Brasília, que para além do academicismo me ofereceram a possibilidade de evoluir como ser humano das mais diversas maneiras possíveis.

Daniel

Aos meus pais e minha irmã, que muito me inspiram e em tudo me apoiam e acreditam em mim.

Aos meus amigos, que sempre me acompanham trazendo alegria e deixando a vida mais leve.

À minha namorada que me apoia e me incentiva a ser uma pessoa melhor.

A todos os professores que ja tive e que me ensinaram muito além de conteúdos mas também a beleza do aprender.

Pedro

Este documento descreve a base teórica do funcionamento de um sistema microcontrolado pelo chip MSP430, com aplicação em áudio. Relata também a parte de idealização e implementação do projeto de um sistema misturador de áudio com eletrônica sensorial embarcada, controlado por meio de um sensor de distância ultrassônico.

Palavras-chave: Eletrônica; Áudio; MSP430.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE CÓDIGOS FONTE

Introdução

Tecnologia e áudio são dois temas intrinsecamente relacionados e a utilização de um sistema microcontrolado no processamento de sinais de áudio se mostra cada vez mais frequente. O objetivo deste trabalho é relatar uma nova técnica de expressão musical por meio de um sistema de eletrônica embarcada imbuído de um sensor de proximidade.

A ideia do projeto adveio da observação da dificuldade de um deficiente físico (cadeirante) durante uma performance com guitarra elétrica quando da necessidade de mudar os diferentes *patches* de efeito de sua pedaleira. Para tanto, o músico tinha de parar de performar, retirar a mão do instrumento, reconfigurar o efeito que desejava e voltar a tocar - nesse processo havia uma perda significativa do tempo da música, o que limitava a execução da peça a poucas ou nenhuma troca de efeitos.

A proposta, então, foi realizar a construção de um protótipo que envolvesse o mapeamento da posição da mão de um instrumentista no braço de um instrumento musical por meio de um sensor, para então ser realizada uma mistura ponderada de sinais advindos de diferentes unidades de efeitos de áudio ou com o sinal puro do próprio instrumento - a aplicação prática remete a uma vasta gama de possibilidades, possibilitando o controle de efeitos, de volume e de outras características do som com menor necessidade do uso de outros membros do corpo (notadamente, os pés).

O trabalho foi segmentado da seguinte forma: o capítulo 1 apresenta o embasamento da ideia do projeto, considerando a parte de tecnologia musical, demonstrando alguns dispositivos já existentes e como isso pode ser benéfico para criação musical. O capítulo 2 trata de processamento de sinais via MSP430, abordando alguns conceitos fundamentais como a diferença entre microprocessador e microcontrolador e detalhando minuciosamente o próprio controlador. O capítulo 3 apresenta a elaboração teórica do projeto, desde o diagrama de blocos até a escolha dos sensores utilizados. No capítulo 4 é apresentada a implementação do projeto elaborado no capítulo 3, exibindo cada parte do sistema. Ao final, o capítulo 5 aborda a conclusão dos trabalhos, citando problemas encontrados e as ideias propostas para a melhora do produto.

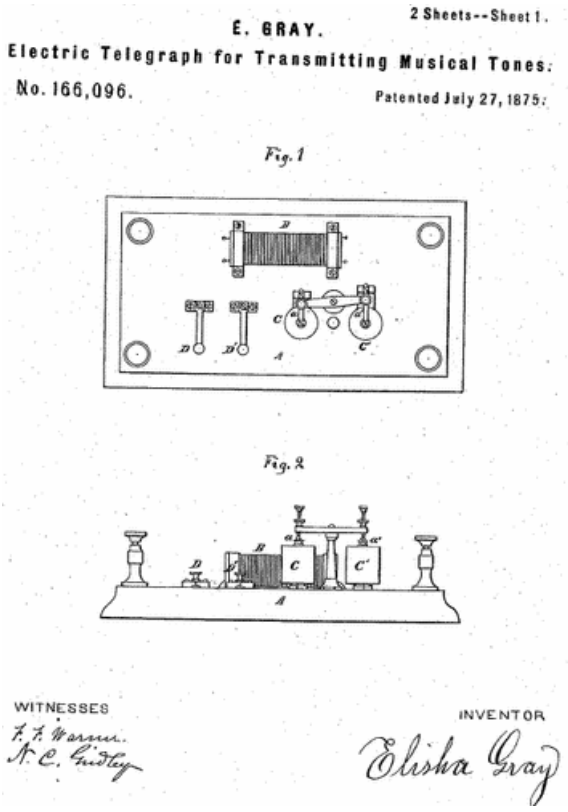
Capítulo 1

Música e Tecnologia

Desde que o homem começou a explorar e dominar a eletricidade, a quantidade de aplicações e utilidades agregadas cresceu em uma dimensão estratosférica. São tantas as possibilidades existentes que pouco a pouco o seu estudo teórico e de aplicações, como o próprio curso superior de engenharia elétrica, se expande e se divide em várias áreas mais específicas.

É no âmbito dessas numerosas aplicações que a eletricidade começou a ser utilizada na música. E mesmo assim, hoje essa área se divide em diversas aplicações. Desde a síntese, o tratamento, a reprodução e a captação do áudio, tudo tem eletricidade envolvida nos dias de hoje.

Figura 1.1 – Desenho-esquemático do Telégrafo Musical, de Elisha Gray.



Fonte: [The Palatin Project]

Antes mesmo do início do século XX, um cientista chamado Elisha Gray trabalhava na Faculdade de Oberlin, em Ohio, nos Estados Unidos, ensinando eletricidade, montando laboratórios e realizando experimentos. Gray Obteve várias patentes relacionadas à engenharia elétrica, na época ainda parte do departamento de ciências, sendo uma delas referente ao seu sintetizador, também chamado de telégrafo harmônico, assim denominado porque utilizava a tecnologia dos telégrafos para gerar um tom. A principal diferença é que seriam vários acoplamentos, possibilitando sintetizar um tom diferente ao controle de cada tecla exclusiva, da mesma maneira que já eram produzidos os sons pelas teclas de um piano. Foi em 1874 que Gray começou a fazer demonstrações públicas da sua invenção, que atualmente foi popularizada na forma de teclados musicais. A Figura 1.1 ilustra a patente mencionada.

Com o tempo, muitas novidades surgiram e, dentre elas, uma das mais revolucionárias de todas: a tecnologia digital. Porém, da mesma maneira que houve uma aversão no início da música eletrônica quando esta era novidade, a transição do analógico para o digital ainda é tabu para muitos, principalmente no histórico da guitarra elétrica. Talvez pelo grande sucesso que o instrumento fez a desde os anos 50, antes da popularização dos transistores para amplificadores de áudio, o "som da válvula"ficou marcado junto ao instrumento. A tradição fala mais alto também em outros âmbitos, como o da música erudita. Segundo [Rocha 2010]:

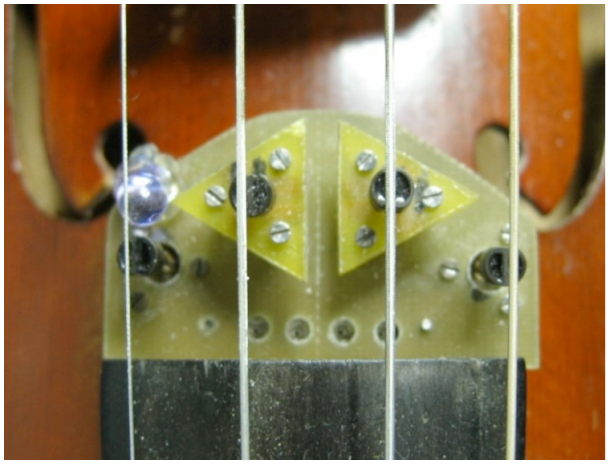
> [...] muitos instrumentistas ainda se sentem inseguros de tocar com eletrônica, pois acreditam ser muito complicado e até mesmo arriscado. Por outro lado, hoje há vários instrumentistas que se dedicam a este repertório e, no seu estudo, identificam características desta prática, como a importância de se entender quais aspectos da performance são influenciados pelo uso de equipamentos eletrônicos e pela relação entre sons acústicos e eletrônicos.

A criatividade musical sempre foi diretamente afetada pela tecnologia que é empregada à musica. De fato, vários são os profissionais que de uma maneira ou outra trabalham com o emprego ou a criação de novas tecnologias de áudio, de *luthieria*, de sonorização, dentre outras atividades.

Existem corporações pesquisando novas tecnologias para serem vendidas (normalmente *design* de produtos) e inclusive laboratórios, em sua grande maioria vinculados a universidades, cuja pesquisa é voltada para esse ramo. Também existem institutos independentes, como é o caso do IRCAM (*Institut de Recherche et Coordination Acoustique/Musique*, que pode ser traduzido do francês como Instituto de Pesquisa e Coordenação de Acústica e Música). Sua fundação é datada de 1969 (ainda que a construção da sede tenha sido terminada apenas no ano de 1978), começo do período de governo de Georges Pompidou como primeiro-ministro da França [Fondazione Renzo Piano]. A observação desse instituto é importante por dois motivos: primeiramente, deixa visível a importância desse campo de pesquisa para a sociedade, uma vez que foi o próprio Pompidou que idealizou um instituto de pesquisas de tecnologia musical; segundo, a data, por sua vez, demonstra que é um ramo reforçado de pesquisas há bastante tempo.

Atualmente, a maior parte das pesquisas é feita com tecnologia digital. Há a possibilidade, antes inimaginável, de o artista reproduzir e controlar sons em tempo real que antigamente só era possível em estúdio. Isso é devido, principalmente, à tecnologia dos sensores. Uma dessas pesquisas com sensoriamento é realizada pelo próprio IRCAM, com a criação na prática de uma ferramenta de captação de som eletroacústica de um violino com um dispositivo ótico [Leroy et al. 2006], na qual utilizou-se um LED infra-vermelho acoplado a um fototransistor para captar a vibração das cordas, conforme a Figura 1.2.

Figura 1.2 – Dispositivo ótico de captação eletroacústica de um violino (projeto do IRCAM)



Fonte: [Leroy et al. 2006]

O controle de áudio em tempo real por gestos é outra tecnologia bastante pesquisada. Segundo [Wanderley 2001]:

> The evolution of computer music has brought to light a plethora of sound synthesis methods available in general and inexpensive computer platforms, allowing a large community direct access to real-time computer-generated sound. Both signal and physical models have already been considered as sufficiently mature to be used in concert situations, although much research continues to be carried on in the subject, constantly bringing innovative solutions and developments.

Um outro projeto do IRCAM é baseado exatamente neste conceito. Utilizando-se de um dispositivo modular, uma espécie de controle sem fio [Rasamimanana et al. 2011], é possível controlar um programa de síntese e, portanto, utilizá-lo para criação musical [Françoise et al. 2013]. Há ainda a possibilidade de ampliar as possibilidades de interação com outros sensores no mesmo dispositivo, utilizando cristais piezoelétricos, *pads* e até encapando-o com um sensor de pressão, de modo que além de sua aceleração nas 3 dimensões pode-se utilizar um controle em quarta dimensão. Isso abre inúmeras possibilidades de expressão musical.

Um outro tipo de aplicação de tecnologia de captura gestual para síntese e controle musical é a empregada pela artista britânica Imogen Heap, numa parceria com a universidade de Western England. Ela utiliza um par de luvas (conforme a Figura 1.4) com vários sensores

Figura 1.3 – O módulo sem fio de controle gestual (projeto do IRCAM)

Fonte: [Rasamimanana et al. 2011]

e transmissor de radiofrequência, que captam a posição e aceleração de suas mãos, a flexão dos punhos e das falanges dos dedos controlando, assim, um software em tempo real. O interessante é que as luvas são utilizadas tanto em trabalhos de estúdio como ao vivo e a otimização do produto é tamanha que é possível tocar instrumentos musicais as tendo vestidas nas mãos.

Figura 1.4 – As *magic gloves* utilizadas por Imogen Heap

Fonte: [Said-Moorhouse 2015]

A própria cantora revela um dos desafios que surgem ao se começar a utilizar um tipo de tecnologia muito diferente, que é o de aprender a operar o sistema. De fato, saber como o sistema funciona não é suficiente para que se tenha fluência em seu uso. Segundo [Website Imogen Heap 2012]:

> The first hurdle was to tame the data coming from the gloves into an adaptable, controllable set of numbers, defining with them the hand postures we'd use. The more we did, the more I began to think in 3D, the more ideas I got and so on and so on! Very soon I needed more data to play with, to be more expressive and have more control.

Este trabalho não está isolado apenas no que se refere ao do uso de tecnologia de sensores para a música. Apesar de certa raridade, existem outros casos de aplicação musical de um dispositivo conhecido como *kit* didático de desenvolvimento e prototipagem. Tal questão foi abordado por [Carvalho et al. 2014], em um trabalho que envolve síntese de áudio com outro tipo de dispositivo também muito utilizado como ferramenta didática, as FPGAs:

> O uso destes dispositivos permite reduzir significativamente o tempo necessário às etapas de projeto, implementação e testes de novos circuitos e sistemas digitais complexos. No entanto, o uso de FPGAs para implementação de sintetizadores digitais de áudio ainda é um tema pouco explorado.

Capítulo 2

O microcontrolador MSP430

2.1 Sistemas embarcados e microprocessadores

O mercado tecnológico atual é repleto de sistemas microcontrolados, o que foi possível com o advento dos transistores há poucas décadas e com a maior facilidade de acesso a circuitos cada vez mais integrados e mais baratos. Antes de se aprofundar o estudo sobre microcontroladores, é necessário que se entenda o que é um microcontrolador e que se saiba a relação destes com os microprocessadores.

Os processadores são unidades de processamento de dados utilizadas para realizar o tomada de decisões e operações com diferentes sinais. Para que isso seja realizado, é necessário que haja uma memória que guarde as configurações (um *firmware*) desse processo, além de notadamente alguns elementos periféricos em um barramento que ofereçam suporte à atividade do processador. Também é necessário que haja um *clock* (literalmente um relógio, que alterna entre um nível lógico alto e um nível lógico baixo a cada ciclo) para que o processador consiga realizar o trabalho para o qual é configurado em sincronia.

Os microcontroladores nada mais são do que um ecossistema completo para que um processador seja utilizado principalmente para controle digital, ou seja, agregam junto a um processador elementos periféricos essenciais como o *clock*, temporizadores (*timers*), memórias, conversores analógico-digitais e portas de entrada e saída de sinais, tornando-o autossuficiente. São comercializados como CIs (circuitos integrados), ou seja, são constituídos de diversos componentes reunidos em um único encapsulamento, contendo terminais que permitem acessar os diferentes módulos contidos dentro do *chip*. Segundo [Mendonça and Zelenovsky 2004]:

> [...] O microprocessador (μP) é dedicado ao processamento, oferece uma grande quantidade de modos de acesso a dados, permite uma série de operações sobre esses dados e trabalha com operações em ponto-flutuante. Resumindo, serve para sistemas que fazem grandes quantidades de operações sobre os dados. Já com o microcontrolador (μC) a ideia é outra. Sua finalidade principal é o controle digital. Deve oferecer uma grande quantidade de

> recursos para entradas e saídas digitais, possibilidade de medir intervalos de tempo e viabilizar sistemas de pequeno tamanho físico. Ele não precisa de realizar operações sofisticadas sobre os dados. Resumindo, nunca um microcontrolador será usado para construir um computador.

A utilização de microprocessadores e microcontroladores é muito comum e extremamente diversificada, tendo aplicação em produtos com baixa necessidade de processamento (como no caso de um controle remoto, por exemplo) até em unidades com poder de processamento extremamente complexo (como é o caso de um processador utilizado em um computador-servidor de uma base de dados).

A vasta aplicabilidade de microcontroladores é devida principalmente em função de sistemas embarcados, que por vezes possuem microcontroladores como núcleo principal de seu funcionamento. O conceito de sistema embarcado, entretanto, vai além no que se refere ao uso da unidade de processamento: o usuário final tem acesso restrito às suas funcionalidades; ou seja, o controlador desempenha tarefas pré-definidas e dá pouco ou nenhum acesso à sua configuração e à sua memória, se tornando parte essencial do produto. Segundo [Davies 2008]:

> There is hardly an electrical consumer product nowadays that does not rely on digital control. This seems reasonable for washing machines and video recorders, but one might wonder why a toaster or a kettle needs any digital electronics. These products contain embedded electronic systems: The processor supports the operation of the product but is not the main reason for purchasing it. There are said to be about 100 embedded processors for each computer, so high-profile, leading-edge microprocessors make up a small part of the market in terms of volume. Fancy modern cars have approaching 100 processors and even a personal computer has embedded processors in its keyboard, mouse, screen, disk drives, and so on. The snag for an engineer is that embedded seems synonymous with invisible and few people appreciate the extent to which they rely on electronics.

A configuração dos microcontroladores é feita por meio de sua programação. Existem desde *softwares* escritos em linguagens de baixo nível de programação até *softwares* mais complexos, escritos em linguagens de alto nível (como é o caso da família de controladores MSP430, que pode ser programada em C).

2.2 Programação e rotina de um microcontrolador

As unidades de memória mais importantes (em termos de hierarquia de memória) são chamadas de registros, que são vetores de dados (ou seja, contém campos de valores que são utilizados como armazenamento de dados). Esses registros são a memória de acesso mais rápido para o processador, e geralmente existem em pequena quantidade (já que costumam ser um tipo de memória cara). O *firmware* do controlador tem como base a escrita e compilação de seu código-fonte e, então, sua execução. Assim, o programa fica gravado na memória

acoplada ao processador de modo que o microcontrolador, contanto que esteja alimentado, possa funcionar em modo *standalone*.

Há o bloco principal do código (em C, é a função *main{}*), que será executado primariamente. Este bloco pode (e costuma) chamar outras funções durante sua execução, seja para configurações do próprio controlador, seja para realizar outras atividades. O *hardware* interpreta o código de maneira linear, executando, assim, linha após linha, desde que as condições de execução sejam verdadeiras (e.g. no caso de um comando *if(condição){instruções}*, caso a condição não seja cumprida, as instruções não serão executadas e a execução do código será desviada para depois do campo *{instruções}*, continuando a partir dali).

2.2.1 *Interruptions* e *flags*

Uma exceção ao disposto no último parágrafo da seção 2.2 são as interrupções e as chamadas *flags* do processador. Isso significa que há momentos em que a leitura do código principal é interrompida, não seguindo para o próximo comando válido dentro de sua estrutura.

As interrupções podem ser entendidas como condições inesperadas (ainda que se espere que uma interrupção seja obtida para que o código tenha certa funcionalidade) de funcionamento do sistema, que ocorrem em momentos imprevisíveis no tempo e requerem um tratamento rápido. Desse modo, a parte do código destinada a esse tratamento tem prioridade em execução, e a ocorrência da interrupção gera uma pausa na execução da rotina para que a exceção em questão seja tratada. As interrupções podem ser reconhecidas tanto por *software* como por *hardware*. No caso do projeto em análise, as interrupções utilizadas foram todas obtidas via *hardware*.

Assim que uma condição de interrupção pré-estabelecida é verdadeira (a dita condição "inesperada"), há a ativação de uma *flag* (do inglês, bandeira), que sinaliza para o processador que uma interrupção foi alcançada. Essa *flag* é guardada em um registro, de modo que sua leitura seja acessível ao programa. Vale ressaltar que diferentes interrupções têm origens distintas (por exemplo, podem sinalizar que houve um flanco de descida do *clock* ou que uma conversão analógico-digital de uma amostra foi realizada) e, por isso, pode-se distinguir as *flags* de sinalização, de forma que ao haver uma interrupção pode-se saber exatamente qual foi sua causa. Além disso, é possível mascarar uma interrupção, ou seja, não permitir que uma *flag* possa ser ativada ou, caso essa mesma *flag* esteja ativa, que ela não interfira em nada no processamento. Portanto, para que uma *flag* possa ser utilizada e tratada pela CPU, deve-se garantir que exista a possibilidade de haver interrupções, habilitando-as.

O sistema automaticamente guarda o valor dos registros atuais e o valor da instrução que está sendo executada em memória, interrompe a função em execução e procura na estrutura do código uma solução para essa interrupção. Assim, o programador deve realizar o *tratamento* da interrupção por meio de uma rotina ISR (*Interrupt Service Routine*, traduzido

como rotina de serviço de interrupção). Cada rotina tem de ser específica para cada fonte de interrupção, e o fabricante do microprocessador pode determinar para qual endereço apontará o vetor de interrupção e de sua rotina (onde essa informação ficará guardada), para que seja realizado o seu tratamento.

Como um exemplo de interrupção pode-se citar o final do ciclo de um *timer*: quando o temporizador atinge seu valor máximo, uma *flag* é ativada no registro do vetor de interrupções desse *timer* e o programa passa, então, para uma rotina de interrupção no código. Assim que essa função de rotina é encontrada, as instruções contidas nela são executadas, a *flag* é desativada e o programa volta ao comando no código onde havia parado anteriormente, seguindo conforme estipulado na seção anterior. Caso haja mais de uma interrupção pendente, o programa executa cada rotina conforme sua prioridade (também estipulada pelo fabricante) e, só quando não há mais *flags* ativas, o programa volta a executar a função principal.

Caso o leitor tenha interesse em aprofundar o conhecimento acerca dos microcontroladores MSP430 e dos módulos utilizados neste projeto, é recomendada a leitura dos Apêndices A e B.

Capítulo 3

O Projeto do Misturador de FX

3.1 O projeto

A ideia básica do projeto é adaptar tecnologias disponíveis para controlar um parâmetro de efeito de guitarra elétrica sem o contato direto com algum dispositivo, como é feito na maioria das vezes com um potenciômetro ou com uma chave ao alcance dos pés. Para substituí-los e se fazer o controle desejado, utiliza-se um sensor de proximidade.

O parâmetro sobre o qual é feito o controle pode ser o da intensidade do efeito. Essa intensidade significa a quantidade de sinal que passa pelo efeito (*fxSignal*) em relação à quantidade de sinal original (*clSignal*), limpo, na soma desses sinais. Esse controle é encontrado na maioria dos efeitos de guitarra, seja distorção, eco, *chorus*, dentre outros. Nos produtos de diversos fabricantes, esse parâmetro recebe o nome de mistura, ou na língua inglesa *blend* ou *mix*, ou ainda *level*, que se refere ao nível do sinal do efeito em relação ao original.

O efeito de guitarra mais visado aqui e talvez o mais comumente usado dentre os guitarristas é o de distorção. Ele age ao saturar de alguma forma o sinal do instrumento, em que ele perde sua linearidade em relação ao sinal original (*clSignal* ou sinal limpo, como é conhecido) e gera mais harmônicos, alterando bastante a sonoridade em várias características como timbre e dinâmica, por exemplo.

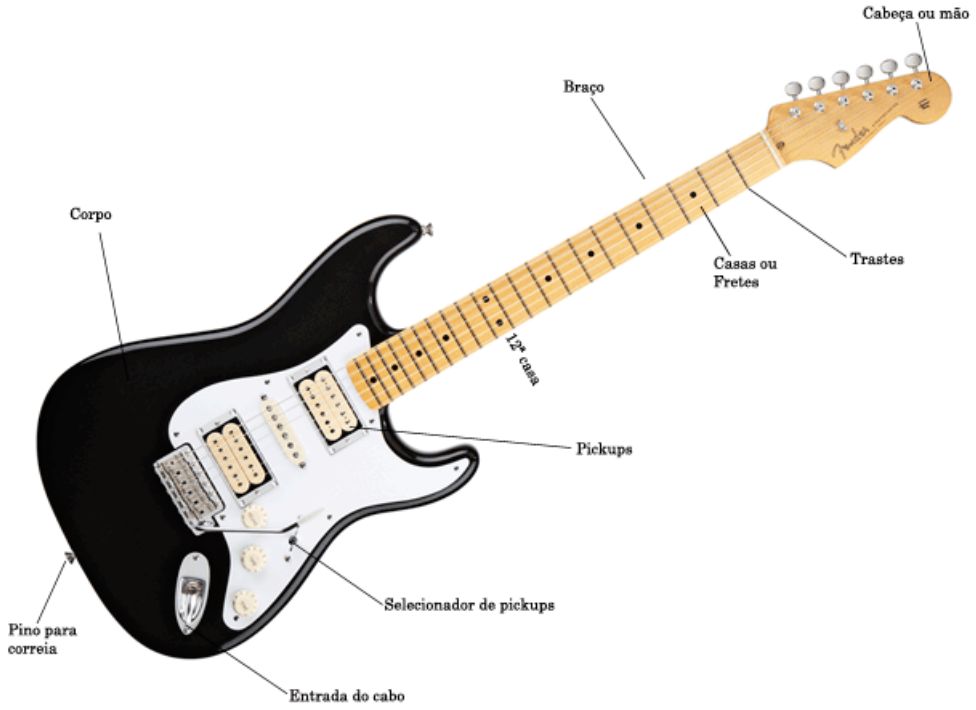
A sonoridade do sinal distorcido, diferente do sinal limpo de uma guitarra elétrica, é mais intensa, possui mais harmônicos e consequentemente um espectro mais cheio. De certa forma é mais impactante ao ouvinte, chama mais atenção, é mais melodioso por possibilitar a sustentação das notas por mais tempo e, portanto, é mais uma ferramenta disponível à expressão musical.

Outro efeito em que esse parâmetro pode ser modificado é o eco. Assim o controle de intensidade mistura o sinal das repetições (eco) com o sinal limpo. A sonoridade do instrumento muda completamente com a variação desse parâmetro. Com a presença mais notável das repetições, o eco pode dar mais ambiência ao som, um efeito mais espacial,

adicionando uma profundidade ao som. Pode-se ainda pôr a perder a definição do som executado em meio a muitas repetições com bastante intensidade de volume se sobrepondo ao sinal limpo da execução.

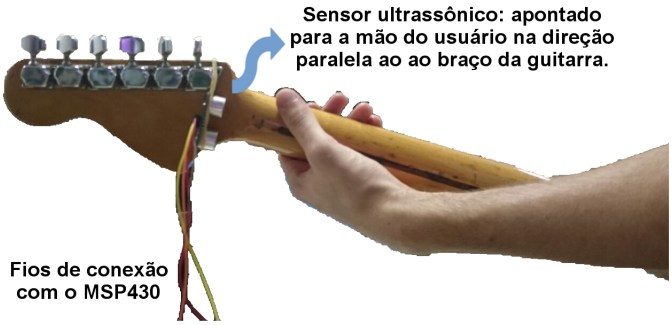
3.1.1 Concepção

Figura 3.1 – A anatomia da guitarra



Fonte: [Academia Musical 2014]

Figura 3.2 – Sensor ultrassônico na guitarra



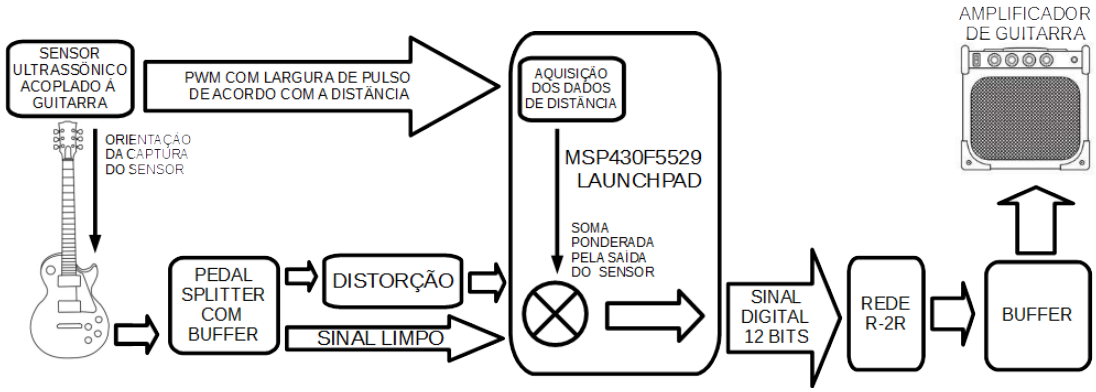
Fonte: Elaborado pelo próprio autor.

Baseando-se nessas ideias, o projeto consiste em um sensor de proximidade fixado na extremidade do braço da guitarra (mão, cabeça ou *headstock*, ilustrada com outras partes da

guitarra na Figura 3.1), detectando a posição da mão do usuário no eixo do braço da guitarra, como demonstrado na Figura 3.2. Assim, quanto mais perto do corpo da guitarra a mão do instrumentista estiver, alcançando notas mais agudas, maior seria a presença do efeito no sinal final. Em contrapartida, quanto mais perto a mão do usuário está do sensor, maior é o ganho do sinal limpo em relação ao do sinal com efeito. Assim, com a escolha da posição da mão sobre o braço da guitarra o executante tem a opção de mais variedades sonoras.

Desse modo, a realização do projeto é baseada na captura em separado dos sinais da guitarra limpo e alterado pelo efeito, e suas subsequentes digitalizações (conversão analógico-digital); na aquisição da distância medida pelo sensor; na mistura de ambos os sinais, ponderada pela reposta de distância do sensor; e, finalmente, na conversão final do sinal digital em analógico. As partes de captura dos sinais da guitarra, interpretação do sensor e soma ponderada dos sinais são todas feitas por meio do *LaunchPad*. A parte da obtenção de dois sinais distintos do mesmo instrumento (e a posterior aplicação de efeito em um dos sinais) e a conversão digital-analógico para o sinal de saída em nível de linha são totalmente externas. Cada uma dessas etapas pode ser observada em detalhe na Figuras 3.3. Primeiramente será analisado o esquema geral do projeto e, em seguida, a relação entre os módulos periféricos e o processador do MSP430F5529 detalhadamente.

Figura 3.3 – Diagrama de blocos do projeto



Fonte: Elaborado pelo próprio autor.

3.2 Os sensores

A parte sensorial do projeto é orientada a capturar a informação de posição da mão do usuário. Para isso foram consideradas diferentes tecnologias: sensores do tipo infravermelho e do tipo ultrassônico. Devido à natureza distinta dessas tecnologias, elas possuem características que podem ser vantajosas ou não para diferentes aplicações.

Na tecnologia de sensores infravermelhos, há duas possibilidades: um sensor de proximidade implementado com um par de LED infravermelho e um fotodiodo ou, no lugar do fotodiodo, um foto-resistor (LDR). Com base nas leituras e pesquisas feitas, a ideia com o foto-resistor foi prontamente descartada. Esse tipo de sensor apresenta uma resposta mais

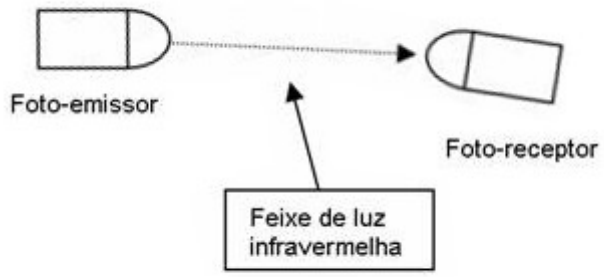
lenta e grosseira, que não seria interessante para a aplicação desejada, já que o movimento do braço do músico pode ser visivelmente mais rápido que a resposta do sensor.

Diante de tais circunstâncias, foram escolhidos para serem testados o sensor infravermelho com fotodiodo e o sensor ultrassônico.

3.2.1 Sensor infravermelho

O sensor infravermelho com fotodiodo funciona a partir da variação da incidência de raios infravermelhos no fotodiodo, onde esses raios são transformados em uma pequena corrente elétrica. Assim, a ideia do sensor é registrar o aumento e a diminuição da incidência de raios infravermelhos no fotodiodo com a mão, de duas maneiras: por incidência direta ou por reflexão. Na incidência direta (ou ponto a ponto), como pode ser esquematizada na Figura 3.4, o LED infravermelho é acoplado à mão do usuário e posicionado a diferentes distâncias do fotodiodo. Mede-se, assim, a resistência no fotodiodo para o teste. No sistema de reflexão, visualizado na Figura 3.5, o LED é posicionado junto ao fotodiodo, os raios são emitidos pelo primeiro, refletidos na mão do usuário para depois incidirem no fotodiodo que novamente tem sua resistência medida para o teste.

Figura 3.4 – Esquema de montagem do sensor ponto a ponto

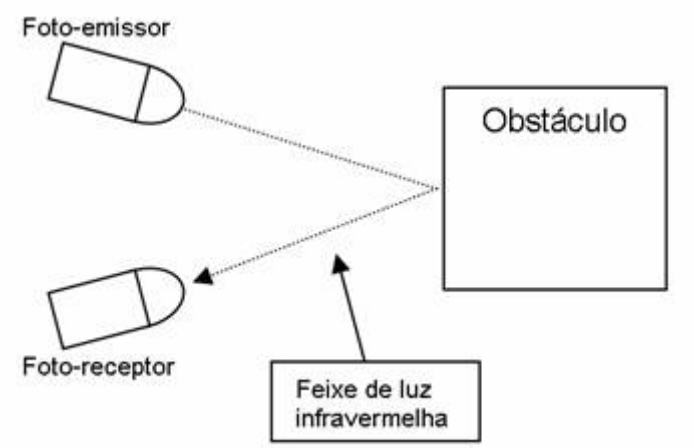


Fonte: Elaborado pelo próprio autor.

Em ambos os casos a quantidade de raios incidentes no fotodiodo é influenciada pela distância devido ao fato da emissão do LED infravermelho não ser unidirecional, fazendo com que haja uma menor absorção quanto maior for essa distância. Para os testes, foram consideradas algumas condições de luz do ambiente que pudessem causar interferência no resultado desejado.

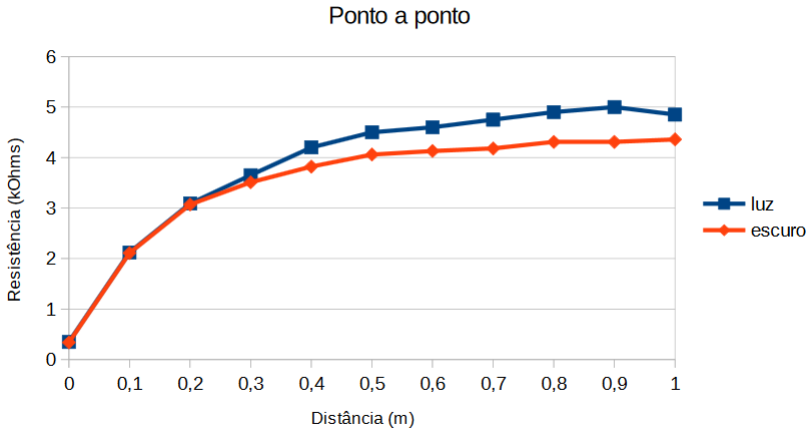
Com base nos resultados das figuras 3.6, 3.7 e 3.8, nota-se que o sensor IR pode funcionar bem se utilizado ponto-a-ponto, porém nem tão bem se usado por reflexão. Isso já é previsto por [Rockwell Automation and Allen-Bradley 1999], que relata: "A quantidade e o tipo de reflexão dos objetos alvos são considerações importantes para as aplicações [...]".

Figura 3.5 – Sensor infravermelho em reflexão



Fonte: Elaborado pelo próprio autor.

Figura 3.6 – Gráfico da Resistência vs. Distância do sensor infravermelho no esquema ponto a ponto



Fonte: Elaborado pelo próprio autor.

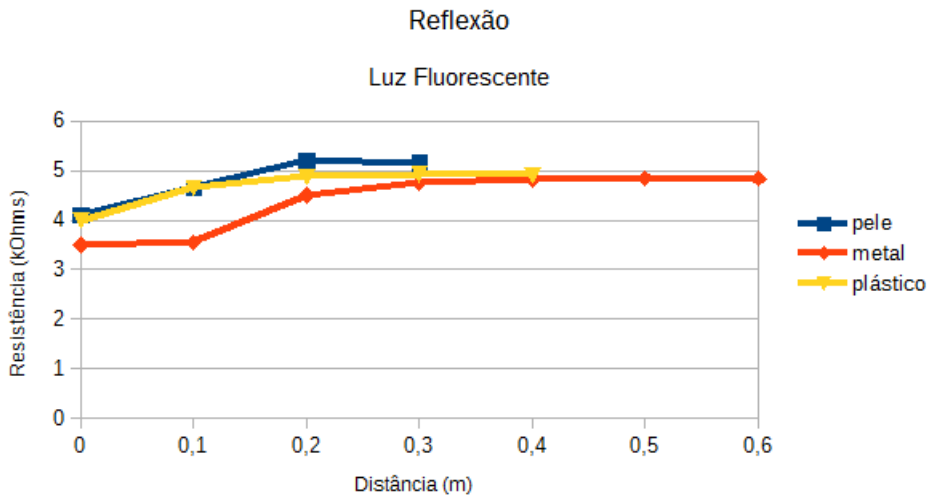
Os gráficos mostram problemas de instabilidade e imprecisão dos resultados para a reflexão. Mesmo assim, nenhum desses sistemas testados se mostram minimamente utilizáveis, já que apresentam: (i) imprecisões, no caso da reflexão; e (ii) a dificuldade de acoplar o LED infravermelho na mão do usuário no caso do ponto a ponto.

3.2.2 Sensor ultrassônico

O sensor ultrassônico consiste em um emissor que envia 8 pulsos de uma onda ultrassônica à frequência de 40 kHz, e um receptor que capta esses mesmos pulsos depois de refletidos, com um natural atraso em relação ao envio. Esse atraso pode ser convertido em distância considerando a velocidade do som constante.

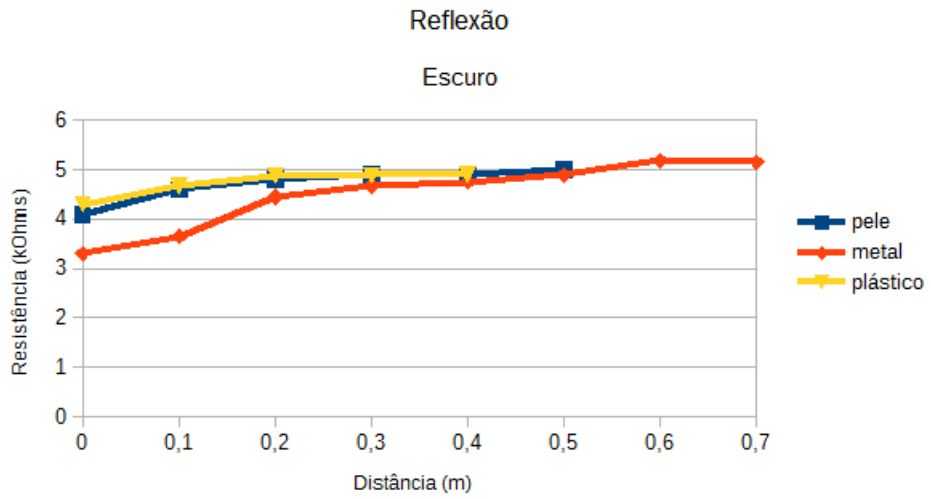
Para funcionar, o sensor deve receber um sinal (*trigger*) de onda PWM com período de 60 ms e largura dos pulsos de pelo menos 10 μs. Esse sinal provoca os disparos dos pulsos

Figura 3.7 – Gráfico Resistência x Distância do sensor infravermelho no esquema de reflexão sob luz fluorescente



Fonte: Elaborado pelo próprio autor.

Figura 3.8 – Gráfico Resistência x Distância do sensor infravermelho no esquema de reflexão em sala escura



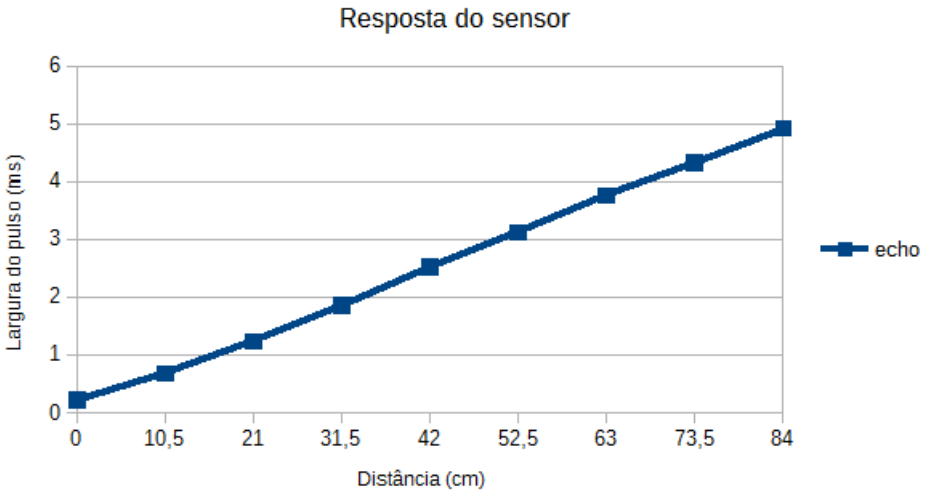
Fonte: Elaborado pelo próprio autor.

de 40 kHz e, a partir daí, o sensor retorna uma saída (*echo*) em alto nível lógico que dura um tempo proporcional à distância. Isso ocorre devido ao fato dos pulsos de alta frequência serem emitidos, refletidos e captados, respectivamente.

Nos testes, com a ajuda de um osciloscópio, pôde-se medir a largura do pulso na saída do sensor (*echo*) sabendo a distância entre o obstáculo e o sensor. A largura do pulso é bem linear em relação à distância, como pode ser observado na Figura 3.9.

O sensor de ultra-som SR04 trabalha apenas com reflexão, condição ideal para a utilização desejada e, como pode ser observado nos resultados, esse sistema funcionou muito

Figura 3.9 – Resposta do sensor, largura do pulso x distância



Fonte: Elaborado pelo próprio autor.

bem para a aplicação esperada. Para se trabalhar com esse sensor ultrassônico a sua saída (em pulsos de largura correspondente à distância) deverá ser adquirida e interpretada pelo microcontrolador, para que este faça o processamento dessa informação e a transforme na ponderação dos sinais amostrados pelo *LaunchPad*.

3.2.3 A escolha do sensor

A escolha do sensor leva em conta diversos fatores. Primeiramente é desejado que o sensor seja capaz de oferecer dados interpretáveis pelo microcontrolador, além de que esses dados sejam em via de regra corretos e com uma precisão mínima. É necessário ainda que o sensor funcione bem para detectar a mão do usuário da guitarra para que não haja erros graves na leitura do valor da posição.

O sensor infravermelho, pelo seu princípio, oferece os dados por meio de um modelo de resistência que varia de acordo com a distância medida. Esses dados podem ser interpretados facilmente pelo microcontrolador se essa resistência for utilizada para variar uma tensão, que pode ser lida pelo conversor AD do microcontrolador.

No que se refere à detecção da distância, o sensor infravermelho não funcionou bem no modo por reflexão e funcionou razoavelmente no esquema ponto a ponto. Esse resultado não é vantajoso para a aplicação desejada porque implica na necessidade de acoplar um LED à mão do usuário, além de que esse LED não poderia se mover em outras direções se não paralelamente ao braço da guitarra. Tais restrições exigidas são quase que impossíveis de serem cumpridas pois atrapalhariam o uso do instrumento para sua finalidade principal, visto que é necessário muitas vezes que o usuário movimente sua mão em várias direções para executar o que é desejado, impossibilitando, assim, a manutenção do alinhamento do LED.

O sensor ultrassônico oferece sua resposta por meio de um pulso digital cuja largura (duração) varia de acordo com a distância captada. Essa largura de pulso é facilmente medida pelo microcontrolador com o uso de *timers* e um sistema de detecção de flanco de subida e de descida.

Na detecção da distância da mão do usuário, o sensor ultrassônico foi capaz de oferecer uma boa resposta, sendo que funciona com reflexão, sem a necessidade de que algo seja acoplado à mão do usuário e assim interfira na execução do instrumento. Sua resposta foi inclusive mais linear em relação à distância do que a do infravermelho.

Analisando tais resultados, não fica dúvidas quanto a superioridade do sensor ultrassônico sobre o infravermelho para essa aplicação, nas configurações adotadas. Pode ser facilmente acoplado ao *headstock* da guitarra e detectar a mão do usuário sem interferir na execução do instrumento e sua resposta pode ser facilmente interpretada pelo microcontrolador.

Capítulo 4

Implementação do projeto

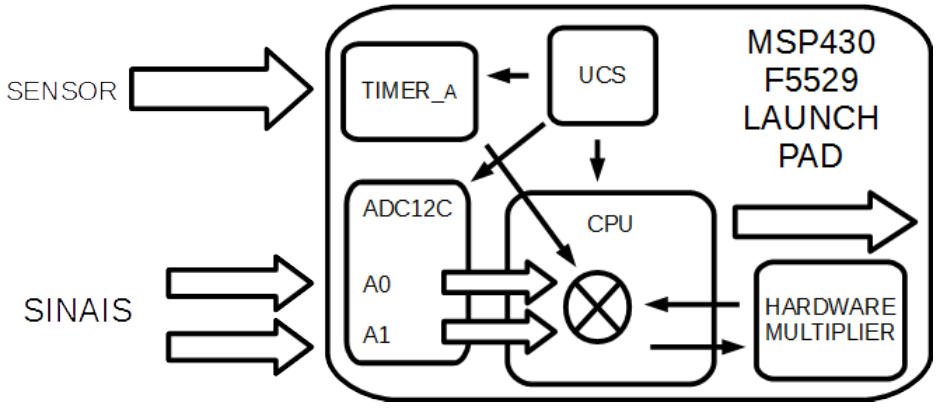
Segue aqui o detalhamento das implementações feitas para este trabalho. Dentre todas as partes e etapas aqui tratadas, podemos agrupá-las em três grandes grupos:

1. Primeiramente a implementação que antecede o *LaunchPad* na sequência do sinal com algumas montagens e a do próprio *LaunchPad* ilustrada no diagrama de blocos da Figura 4.1, onde se tem as configurações dos *clocks*, *timers*, conversor AD e portas utilizadas. São detalhadas também as rotinas de interrupção e o *Hardware Multiplier* para realizar as tarefas necessárias integradas aos blocos;

2. Posteriormente a atenção se volta à implementação na saída do *LaunchPad*, após o processamento até a reprodução do som, conforme pôde ser observado na Figura 3.3. Nesta parte o foco é a conversão digital-analógico (DAC) e em seguida o *buffer* de saída do sinal;

3. Finalizando o capítulo tem-se uma visão mais externa e geral do protótipo criado neste trabalho com todas as partes implementadas, em conjunto com alguns testes, e resultados obtidos.

Figura 4.1 – Diagrama de blocos em detalhe do LaunchPad



Fonte: Elaborado pelo próprio autor.

4.1 Configurações (*setup*) do UCS

A primeira configuração a ser feita envolve o UCS, cujo código é detalhado na Listagem 4.1. Basicamente, o UCS controla todos os *clocks* disponíveis para adequá-los à utilidade desejada, portanto a configuração aqui feita é válida não só para a implementação do sensor, mas sim de todo o sistema envolvendo o ADC. São 10 registros de controle (UCSCTL0 até UCSCTL9) mas utilizou-se apenas os seis primeiros para a configuração desejada.

Listagem 4.1 – Código de configuração do UCS

```
1  void setupUCS() {
2
3  WDTCTL = WDTPW | WDTHOLD; // Hold Watchdog Timer
4
5  UCSCTL0  = 0x0000;        // Set lowest DCOx, MODxw
6  UCSCTL1  = DCORSEL_5;     // Select DCO range to 16MHz (2.5MHz < Range5 < 54.1MHz)
7  UCSCTL2 |= 244 |          // Multiplier N: (N+1) * FLLD' * 32.768kHz = 16.056320
8                            //;MHz, FLLD'=2, FLLD__2;
9  UCSCTL3  = FLLREFDIV_0 |// FLL divider 1;
10           SELREF__XT1CLK;//DCO FLL source = XT1
11 UCSCTL4  = SELA_0 |      // ACLK source = XT1;
12             SELS_3 |     //;SMCLK source = DCOCLK;
13             SELM_3;      //;MCLK source = DCOCLK
14 UCSCTL6 &= ~XT1OFF;      // Ensure XT1 is on
15
16 __delay_cycles(250000); // Accomodation cycles
17
18 }                         // Close setupUCS()
```

Antes de tudo, o *watchdog timer* é desligado. Já no UCS, o primeiro registro de controle, o UCSCTL0 é composto de alguns bits reservados sempre lidos em 0 e os campos do DCO e do MOD. Ambos (DCO e MOD) são ajustados no mínimo possível, pois a modulação será utilizada e assim serão modificados automaticamente durante a operação do FLL.

O UCSCTL1 possibilita o ajuste da faixa de frequência em que o DCO será operado. Para obter 16 MHz, a sexta faixa de frequência possível nos bits DCORSEL_5, que vai aproximadamente de 4 MHz até 40 MHz, é selecionada. Neste registro a modulação pode ser desligada, mas não é o caso. A modulação aqui é útil para manter uma frequência melhor regulada com ajuda do FLL.

Em sequência, o UCSCTL2 é composto de multiplicadores da frequência de referência usada pelo DCO para chegar no valor desejado. O *clock* do DCO será utilizado pelo processador, e é o mais alto dentre os necessários para este projeto, aproximadamente 16 MHz, já que mais ciclos são necessários para processar certas tarefas enquanto módulos específicos têm suas arquiteturas especializadas para determinadas funções. Sabendo que o *clock* final do DCO é calculado pela equação 5.2, segue a escolha dos próximos bits aplicados nessa equação resultando na expressão 4.1. O *clock* de referência a ser selecionado no próximo registro será o XT1CLK, que pulsa a 32768 Hz. Para este caso, o FLLN é ajustado em 244

para obtermos N+1 = 245, e o FLLD recebe o valor de multiplicador 2. Com o FLLREFDIV escolhido como 1, tem-se:

$$DCOCLK = (244 + 1) \cdot (2) \cdot \frac{32768}{1} = 16056320 \tag{4.1}$$

Como dito no parágrafo anterior, o registro UCSCTL3 seleciona a fonte do *clock* de referência para o FLL como SELREF_XT1CLK, correspondendo ao XT1CLK. Além disso este registro controla também um divisor do *clock* que no caso é 1, já que o *clock* final desejado já teve sua frequência alcançada.

Do registro UCSCTL0 até o UCSCTL3, as configurações feitas foram todas referentes ao DCOCLK, que pelo seu funcionamento possui algumas opções exclusivas como FLL, modulação, dentre outras. No UCSCTL4, escolheu-se a origem do *clock* das saídas ACLK, MCLK e SMCLK. Para o sensor ultrassônico, selecionou-se as fontes do ACLK, SMCLK e do MCLK, sendo o primeiro usado no *timer* para o LED demonstrativo, o segundo no *timer* do sensor e o terceiro na CPU. Assim, o ACLK recebe o *clock* diretamente do XT1CLK enquanto o MCLK e o SMCLK são alimentados pelo DCOCLK.

Logo abaixo das configurações do UCS, é importante a função "__delay_cycles(250000)" para interromper a sequência do código até que o número determinado de ciclos de *clock* seja completado e a FLL trave na frequência desejada para não causar nenhum problema adiante, ou seja, para que o sistema se estabilize.

4.2 Implementação do sensor ultrassônico

Para implementar o sensor ultrassônico no sistema embarcado do MSP430, são necessários, além de sua alimentação, o tratamento de dois sinais. É necessário um sinal de *trigger* saindo do MSP430 para que o sensor dispare os pulsos ultrassônicos e, finalmente, a interpretação da saída do sensor que entra no MSP430.

Primeiramente, o código deve gerar o sinal de *trigger*. Esse sinal é uma onda PWM de período e largura de pulso fixos, que pode ser gerada com o uso de *timers* e suas interrupções para controlar uma saída e formar o sinal desejado. Para obter os resultados, um procedimento parecido é feito. Usando o mesmo *timer*, são configuradas as interrupções de borda de subida e descida da porta que recebe o sinal do sensor (*echo*), para que os valores de contagem do *timer* sejam armazenados temporariamente e subtraídos um do outro, já que a diferença no tempo entre o bordo de subida e de descida é necessariamente o tempo medido da largura do pulso (e essa diferença no tempo é modelada como ciclos do *timer*).

4.2.1 Configurações (*setup*) do sensor

Abaixo é apresentado o código utilizado (mostrado na Listagem 4.2.1) e também são detalhadas as configurações necessárias para implementar essa funcionalidade. No todo, para a implementação do sensor ultrassônico é necessário configurar, além do *Unified Clock System* (UCS), as portas (GPIO) e os *timers*.

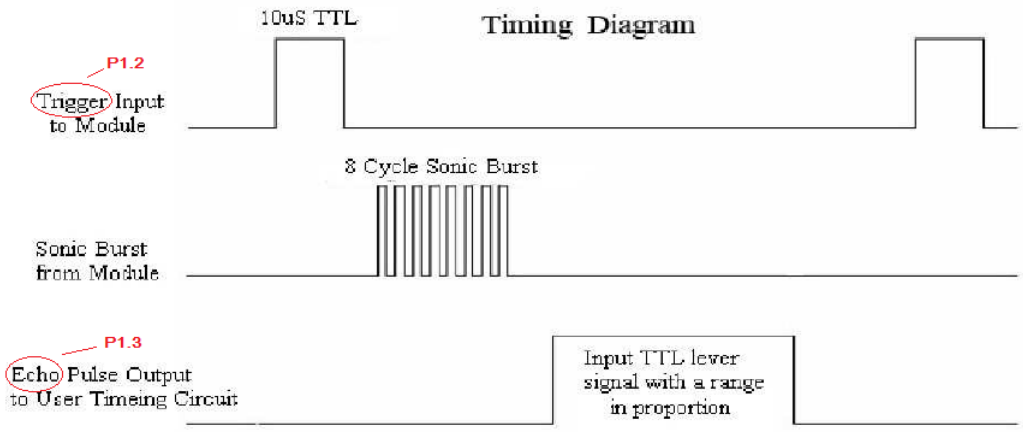
Listagem 4.2 – Código de configuração do sensor ultrassônico envolvendo GPIO e *timers*

```
1  void setupUS() {
2
3  // TRIGGER configure
4  P1DIR |=  BIT2;          // P1.2 Timer Capture Output (TRIGGER)
5  P1OUT &= ~BIT2;          // Trigger starts low
6
7  // ECHO configure
8  P1SEL |=  BIT3;          // P1.3 is PERIPHERAL (TimerA CCI2A)
9  P1DIR &= ~BIT3;          // P1.3 Timer Capture Input (ECHO)
10 P1REN |=  BIT3;          // Resistors on for ECHO
11 P1OUT &= ~BIT3;          // Pull-down resistor for P1.3
12
13 // TEST LED configure
14 P1DIR |=  BIT0;          // P1.0 RED LED
15 P1OUT &= ~BIT0;          // RED LED off
16
17 // TIMERS configure
18
19 TA0CTL = TASSEL_1      | // ACLK = 32768 Hz;
20           MC_1         | // UP mode;
21           TACLR        | // Clear;
22           TAIE;          // Interrupt enable;
23           TA0CCR0 = 1966;// CCR0 = 60ms --> (1/32768)*1966
24           TA0CCR1 = 4;   // CCR1 = 12,207us --> (1/32768)*4
25
26 TA0CCTL0 = CCIE;         // Interrupt enable
27 TA0CCTL1 = CCIE;         // Interrupt enable
28 TA0CCTL2 = CM_3    |     // Capture BOTH edges;
29            CCIS_0  |     // Input select CCI2A;
30            SCS     |     // Sync capture;
31            CAP     |     // Capture mode;
32            CCIE;         //; Interrupt enable
33
34 TA1CTL = TASSEL_1 |      // ACLK = 32768Hz;
35           MC_1     |     // UP mode;
36           TACLR    |     // Clear;
37           TAIE;          // Interrupt enable;
38 TA1CCTL0 = CCIE;         // Interrupt enable
39 TA1CCTL1 = CCIE;         // Interrupt enable
40 TA1CCR0 = 400;           // CCR0 = 15,26ms --> (1/32768)*500, LED
41                          //;blinks at 65.536Hz;
42 TA1CCR1 = 164;           // CCR1 is initialized as 5ms. Related to
43                          //;the DUTY CYCLE. Will be modified every TA0_ISR
44
```

```
45 }                                   // Close setupUS()
```

A primeira configuração necessária envolve as portas utilizadas (GPIO). Primeiramente a porta P1.2 é associada como saída (*output*) e iniciada em nível lógico baixo. Essa será a saída do sinal de *trigger*. Da mesma forma para o *echo*, a porta P1.3 é selecionada como entrada (*input*) e como *peripheral*, o que significa que estará associada ao *timer* A0, na entrada CCI2A, que dispara o módulo CCR2 para capturar o valor do *timer* no momento. Como é entrada, configura-se ainda resistores em modo *pull-down* para P1.3. A alocação de pinos realizada pode ser vista na Figura 4.2, que detalha o funcionamento do sensor ultrassônico. O led vermelho, que é a porta P1.0, será usado como teste para o sensor e para isso é configurado como saída e iniciado como desligado.

Figura 4.2 – Diagrama de funcionamento do sensor ultrassônico



Fonte: Adaptado de [Elec Freaks].

Com o GPIO (portas) e o UCS devidamente ajustados, cabe à próxima etapa de configuração do sensor preparar o *timer* de acordo com o necessário. Dentre os *timers* disponíveis foi utilizado o *timer* A0 para enviar o sinal de *trigger* e capturar o sinal de *echo*. O *timer* A1 será utilizado apenas para gerar uma PWM com *duty cycle* proporcional à distância do sensor, útil para verificar o seu funcionamento por meio do LED.

O primeiro registro a ser ajustado é o TA0CTL, que seleciona a fonte do *clock* como ACLK (TASSEL_1) como previsto na configuração do UCS. Ainda nesse registro, o *timer* é colocado em “UP mode” (MC_1). Ainda neste registro, temos um bit TACLR ajustado para fazer um *reset* no contador do *timer* e um bit TAIE que habilita a interrupção da *flag* TAIFG. Ambos recebem valor 1 para habilitarem suas respectivas funções.

Para cada um de seus módulos de captura/comparação (CCR) a serem utilizados, há um registro respectivo que o controla. Para a aplicação desejada é necessário o uso dos três primeiros módulos, sendo que o primeiro e o segundo funcionam em modo de comparação. A função deles é gerar interrupções para construir a onda PWM que alimenta a entrada do sensor (*trigger*). Para isso não são necessárias muitas alterações nos registros, porque a

configuração padrão já é bem próxima da correspondente à função desejada. Apenas são habilitadas as interrupções da *flag* CCIFG de cada um dos dois módulos.

No terceiro módulo, a função já é diferente e precisa de outras configurações. Em vez de comparação, o modo é de captura, já que será capturado o valor do contador na borda de subida e depois na borda de descida para obter-se a largura do pulso do sinal *echo* da saída do sensor. Para isso os dois bits mais significativos do registro TA0CCTL2 são configurados como CM_3 (11b) para indicar a captura tanto em borda de subida como de descida. Os próximos 2 bits mais significativos (CCIS) são atribuídos como CCIS_0 (00b) e assim selecionam a entrada do módulo que provocará a captura, que no caso é a entrada CCI2A, correspondente ao pino P1.3, de acordo com o *datasheet*. Em sequência o bit referente ao compo do SCS é ajustado para que o sinal de entrada da captura seja sincronizado com o *clock* do *timer*. Esse ajuste não é necessário mas contribui para um melhor funcionamento da captura. O campo CAP configurado como 1 seleciona o modo de captura, enquanto o valor 0 (que é o da configuração padrão) habilita o modo de comparação. Finalmente, assim como nos registros anteriores de controle dos blocos de captura/comparação, o bit referente ao CCIE é ajustado para ativar a interrupção da *flag* CCIFG referente à captura neste caso.

O nível máximo do contador do timer A0, mencionado anteriormente, é ajustado no registro TA0CCR0, o qual recebe o número máximo de ciclos do *timer*, a partir do qual é reiniciada a contagem. Tal número relaciona-se diretamente com o tempo, ou seja, para se obter o período da PWM desejado para o sinal *trigger* do sensor, foi escolhido o número de ciclos à frequência de 32768 Hz necessários para totalizar 60 ms, que equivale a aproximadamente 1966 ciclos. Esse valor encontrado foi atribuído ao TA0CCR0.

Da mesma forma usada no TA0CCR0, atribui-se ao registro TA0CCR1 o valor de 4 ciclos, que correspondem a 12,2 μs, usado como largura de pulso para o sinal *trigger* do sensor. Assim feito, tem-se duas referências de tempo no *timer*, uma para o período total da PWM do *trigger* (TA0CCR0) e a outra para a largura do pulso deste mesmo sinal. A partir daí, o mesmo *timer* é utilizado para a contagem de tempo da largura do pulso do sinal de saída do sensor (*echo*).

4.2.2 Rotinas de interrupção do sensor

Com todas as configurações necessárias prontas, segue-se para o funcionamento do sensor, a rotina a ser chamada nas ocasiões de interrupções. O código mostrado na Listagem 4.2.2 é baseado nas interrupções. A leitura do vetor de interrupções TA0IV permite saber a fonte da interrupção e assim tratá-la com a função desejada para cada ocasião de interrupção.

Listagem 4.3 – Código da interrupção dos *timers*

```
1 // Interrupt Service Routine (sensing process interrupts)
2 #pragma vector=TIMER0_A0_VECTOR // Interrupt Service Routine for TA0CCR0
3 __interrupt void TA0_CCR0_ISR(void){
4 }                                 // Close TA0_CCR0_ISR
```

```
5
6  #pragma vector=TIMER0_A1_VECTOR // Interrupt Service Routine for TA0 non-CCIFG0
7   __interrupt void TA0_ISR(void){
8    switch(TA0IV) {
9    case 2:                        // Interrupt source TA0CCR1
10      P1OUT &= ~BIT2;             // Set P1.2 off (STOP TRIGGER)
11      break;
12   case 4:                        // Interrupt source TA0CCR2
13                                  //;(RISING AND FALLING EDGES)
14      if (TA0CCTL2 & CCI)         // CCR2 interrupt (RISING edge)
15       riseEdge = TA0CCR2;        // Store rising edge time value
16      else {                      // CCR2 interrupt (FALLING edge)
17       fallEdge = TA0CCR2;        // Store falling edge time value
18       pulseWidth =               // Number of timer cycles taken
19           (fallEdge-riseEdge);
20      }                           // Close if-else
21      break;
22   case 0xE:                      // Interrupt source TAIFG (CCR0)
23      P1OUT  |=  BIT2;            // Set P1.2 on (START TRIGGER)
24      break;
25   default: break;
26   }                              // Close Switch-Case
27 }                                // Close TA0_ISR
28
29
30 #pragma vector=TIMER1_A0_VECTOR // Interrupt Service Routine for TA1CCR0
31 __interrupt void TA1_CCR0_ISR(void){
32 }                                // Close TA1_CCR0_ISR
33
34
35 #pragma vector=TIMER1_A1_VECTOR // Interrupt Service Routine for TA1 non CCIFG0
36 __interrupt void TA1_ISR(void){
37   switch(TA1IV) {
38   case 2:                        // Interrupt source TA1CCR1 (pulse ending)
39      P1OUT &= ~BIT0;             // Reset P1.0 (RedLed OFF)
40      break;
41   case 0xE:                      // Interrupt source TA1CCR0 (PWM Overflow,
42                                  //;pulse beginning)
43      P1OUT |=  BIT0;             // Set P1.0 (RedLed ON)
44      break;
45   default: break;
46   }                              // Close switch-case
47
48   TA1CCR1 = pulseWidth;          // TA1CCR1 is related to last DutyCycle
49 }                                // Close TA1_ISR
```

Para toda interrupção proveniente do bloco de captura/comparação CCR1 o sinal do *trigger* é colocado em nível baixo, marcando o fim do pulso da PWM. Para interrupção vinda do CCR2, uma função *if* analisa a entrada da captura (sinal *echo* do sensor), para saber se a borda é de subida ou de descida. Assim pode-se, então, atribuir o valor do contador (à ocasião do momento da captura) a diferentes variáveis: para o caso de subida (*riseEdge*) e, de descida, (*fallEdge*). A partir desse ponto é feita a diferença entre essas duas variáveis

para obter o tamanho da largura de pulso da resposta do sensor (*echo*) e por consequência a posição do objeto em relação ao sensor. Finalmente para a interrupção causada pelo bloco CCR0 o sinal destinado ao *trigger* é colocado em nível alto iniciando o pulso da PWM.

A rotina que segue utilizando outro *timer* tem utilidade exclusiva de reproduzir uma PWM de *duty cycle* proporcional à distância medida no sensor. Essa PWM é destinada ao LED vermelho da porta P1.0, que dessa forma terá sua intensidade de brilho variando em dependência do sensor. Isso é feito com um período do *timer* (CCR) fixo e o primeiro módulo de captura/comparação com valor variável proporcional à medida feita pelo sensor. Isso foi feito dessa forma porque o brilho aparente do LED varia conforme a potência RMS aplicada a ele: uma onda quadrada com menor *duty cycle* apresenta uma menor potência RMS, fazendo com que o brilho aparente seja menor. O contrário, quando o *duty cycle* é maior, resulta em um brilho aparente maior. Assim, uma PWM é uma técnica eficaz para esse tipo de controle, já que muda constantemente a largura do pulso, portanto variando a potência RMS no LED e, por conseguinte, seu brilho.

4.3 Implementação do conversor AD (ADC12_A)

O MSP430 possui um conversor analógico-digital com 12 canais de entrada externos. No entanto, para o presente projeto há a necessidade de utilização de apenas dois deles, um para o sinal original proveniente da guitarra e outro para o sinal que passa pelo pedal de efeito. Deste modo, é feita a conversão sequencial dos canais. Para configurar o ADC12 para realizar a amostragem dos dois sinais e ser possível posteriormente somá-los será necessário configurar o GPIO, o UCS (já configurado anteriormente), a tensão de referência e o próprio ADC12_A. Seguindo os códigos nas listagens 4.3.1 e 4.3.2, é feita em seguida uma explicação sobre as configurações escolhidas para o projeto.

4.3.1 Configurações (*setup*) do ADC

Listagem 4.4 – Código de configuração do ADC12 incluindo GPIO UCS e REF

```
1  void setupADC() {
2
3  // GPIO configure
4  // ADC INPUT CHANNELS
5  P6SEL     |= BIT0 |          // P6.0 peripheral - ADC A0;
6               BIT1;          //;P6.1 peripheral - ADC A1
7  P6REN     |= BIT0 |          // Resistors on for A0
8               BIT1;          // Resistors on for A1
9  P6OUT     &=~BIT0 |          // Pull-down resistor for A0
10              BIT1;          // Pull-down resistor for A1
11
12 // ADC OUTPUT CHANNELS
13 P1DIR     |= BIT6;          // P1.6 is output (ADC BIT 00)
```

```
14 P2DIR      |= BIT7;                 // P2.7 is output (ADC BIT 03)
15 P3DIR      |= BIT2 |                // P3.2 is output (ADC BIT 02)
16                BIT5 |                // P3.5 is output (ADC BIT 06)
17                BIT6;                 // P3.6 is output (ADC BIT 07)
18 P4DIR      |= BIT1 |                // P4.1 is output (ADC BIT 05)
19                BIT2;                 // P4.2 is output (ADC BIT 04)
20 P6DIR      |= BIT2 |                // P6.2 is output (ADC BIT 11)
21                BIT3 |                // P6.3 is output (ADC BIT 10)
22                BIT4 |                // P6.4 is output (ADC BIT 09)
23                BIT6;                 // P6.6 is output (ADC BIT 01)
24 P7DIR      |= BIT0;                 // P7.0 is output (ADC BIT 08)
25 P1DS       |= BIT6;                 // P1.6 high drive strength (ADC BIT 00)
26 P2DS       |= BIT7;                 // P2.7 high drive strength (ADC BIT 03)
27 P3DS       |= BIT2 |                // P3.2 high drive strength (ADC BIT 02)
28                BIT5 |                // P3.5 high drive strength (ADC BIT 06)
29                BIT6;                 // P3.6 high drive strength (ADC BIT 07)
30 P4DS       |= BIT1 |                // P4.1 high drive strength (ADC BIT 05)
31                BIT2;                 // P4.2 high drive strength (ADC BIT 04)
32 P6DS       |= BIT2 |                // P6.2 high drive strength (ADC BIT 11)
33                BIT3 |                // P6.3 high drive strength (ADC BIT 10)
34                BIT4 |                // P6.4 high drive strength (ADC BIT 09)
35                BIT6;                 // P6.6 high drive strength (ADC BIT 01)
36 P7DS       |= BIT0;                 // P7.0 high drive strength (ADC BIT 08)
37
38 // Voltage Reference configure
39
40 REFCTL0  = REFMSTR | REFON;        // 1,5V voltage reference
41
42 // ADC configure
43
44 ADC12CTL0 &= ~ADC12ENC;            // Ensure enable conversion is off
45
46 ADC12CTL0 |= ADC12SHT0_2  |        // A0 sample time = 16 ADC12CLK cycles;
47               ADC12SHT1_2  |        //;A1 sample time = 16 ADC12CLK cycles;
48               ADC12ON      |        //;ADC12 on;
49               ADC12OVIE    |        //;ADC12 overflow interrupt enable;
50               ADC12TOVIE;           //;ADC12 conversion time interrupt en.
51 ADC12CTL0 &=~ADC12MSC;              // Further S&C are not automatic;
52 ADC12CTL1 |= ADC12SHS_0   |        // SC source = Timer_B
53               ADC12SHP     |        // S&H pulse sourced from timer
54               ADC12DIV_0   |        //;Clock divider = 1;
55               ADC12SSEL_2  |        //;Clock source = MCLK = 16MHz;
56               ADC12CONSEQ_1;        //;Sequence of channels conversion
57 ADC12CTL2 |= ADC12TCOFF   |        // Temperature sensor off;
58               ADC12RES_2 ;          //;12 bit resolution;
59 ADC12MCTL0|= ADC12SREF_0  |        // Reference = AVss-Vref(-);
60               ADC12INCH_0;          //;MEM0 input channel = A0 (P6.0)
61 ADC12MCTL1|= ADC12EOS     |        // End of sequence (last channel);
62               ADC12SREF_0  |        //;Reference = AVss-Vref(-);
63               ADC12INCH_1;          //;MEM1 input channel = A1 (p6.1)
64 ADC12IE     = 0x03;                 // Enable interrupt for A0 and A1
65
66 // Timer Configure
67
```

```
68 TA2CTL      = TASSEL_2 |          // SMCLK = 16MHz;
69               MC_1   |          //;UP mode;
70               TACLR  |          //;Clear counter;
71               TAIE;             //;Interrupt enable.
72 TA2CCR0     = 180;               // At 16Mhz 180 cycles = [1/(2(44100Hz))]s
73 }                                // Close setupADC()
```

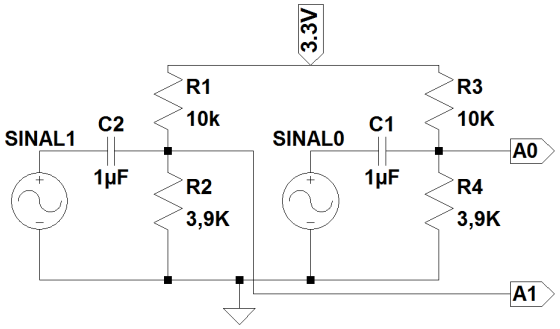
A configuração do GPIO é bem simples pois envolve apenas selecionar as portas já dedicadas à entrada do ADC como função de *peripheral mode*. Das entradas mostradas no esquema do ADC12_A, usaremos as entradas A0 e A1, que correspondem aos pinos P6.0 e P6.1 respectivamente. Então, esses pinos são configurados como periféricos ao invés da função de I/O.

Da mesma maneira, são configuradas todas as portas de saída dos bits que serão posteriormente convertidos em sinal analógico. Para isso os campos PDIR são configurados como 1, ou seja, para *output*. Como essas saídas serão acopladas a um circuito RLC em seguida, é modificado ainda o campo PxDS, que dá à porta a liberdade de oferecer maior corrente de saída.

A configuração do UCS feita para o sensor também serve ao ADC pois se encontram no mesmo programa. Ou seja, o ajuste do UCS já foi realizado para o funcionamento adequado do ADC também.

Antes de se ajustar o ADC em si, é designada a referência a ser por ele utilizada. Foi escolhida uma tensão de referência com Vref(+)=AVcc e Vref(-)=AVss. Como os sinais de áudio não possuem *offset* e sabendo que a tensão de referência mínima AVss é de 0V, há um problema na amostragem desses sinais quando atingem valores menores que 0 (semiciclos negativos). Para resolver esse problema foi acrescentado a A0 e A1 um sinal DC de aproximadamente 1 V (chamado *terra virtual*), utilizando-se o pino de 3,3 V do LaunchPad com um divisor de tensão formado por resistores de 10 kΩ e 3,9 kΩ, respectivamente, conforme a Figura 4.3. O capacitor de entrada de 1 μF foi utilizado para remover qualquer *offset* prévio do sinal (geralmente causado por espúrios).

Figura 4.3 – Implementação de um sinal DC na entrada dos pinos A0 e A1



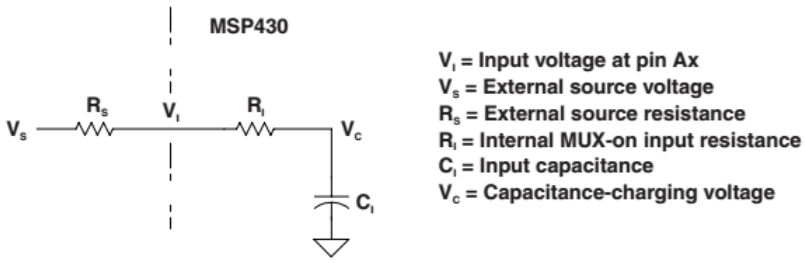
Fonte: Elaborado pelo próprio autor.

Dando continuidade à implementação do código do ADC, é necessário garantir que as conversões sejam desabilitadas antes da configuração dos registros do próprio módulo

ADC12_A, para que só depois de tudo ajustado elas sejam habilitadas. Caso isso não seja feito, existe o risco das configurações não serem corretamente efetivadas e o ADC não funcione como o esperado. Assim, logo no primeiro registro de controle, o campo que habilita as conversões no ADC12CTL0 (ADC12ENC) recebe valor 0.

Ainda no registro ADC12CTL0, são atribuídos aos bits referentes aos campos ADC12SHT0 e ADC12SHT1 o valor 2 (10b), para que o tempo do processo de *Sample and Hold* seja fixado em 16 ciclos do ADC12CLK. Há a necessidade desse tempo para que a medida seja correta porque a entrada do ADC pode ser modelada como na figura 4.4, onde a resistência da entrada interfere na velocidade de carga do capacitor.

Figura 4.4 – Modelo da entrada do ADC



Fonte: [Texas Instruments 2015].

Esse número de ciclos pode ser calculado pela fórmula 4.2 presente no *user's guide* [Texas Instruments 2014]:

$$t_{sample} > (R_S + R_I)\ln(2^{n+1})C_I + 800[ns] \tag{4.2}$$

Tal fórmula, considerando R_s no valor de 1 Mohm, resulta em um período T_{sample} de 0,226 ms aproximadamente, que corresponde a 7,4 batidas de *clock*, sendo necessários então 8 ciclos. Além desses 8 ciclos, considera-se uma folga para que diminua a chance de erro. Com 16 ciclos é dado um espaço mais do que suficiente para a o processo de amostragem.

O bit ADC12MSC ajusta o *sample timer* para esperar uma borda de subida do sinal SHI para disparar cada amostragem e conversão. Caso contrário, as conversões são feitas uma logo após a anterior ter sido finalizada, sem a sincronia necessária para amostragem de áudio.

Ainda no primeiro registro do ADC12_A, o ADC é ligado pelo bit ADC12ON e mais duas interrupções são habilitadas. A interrupção de *overflow* dos registros ADC12MEMx é ativada no bit ADC12OVIE e a interrupção do *overflow* do tempo de conversão também mas pelo bit ADC12TOVIE.

No segundo registro do ADC, primeiramente seleciona-se a fonte do sinal de amostragem (SAMPCON) como o *sampling timer*. Em seguida, o ADC12CLK é dividido por um no campo ADC12DIV_0 e a fonte do *clock* é escolhida como sendo MCLK nos bits referentes ao campo ADC12SSEL_2 (10b), como já fora preparado para isso. Em seguida é selecionada o tipo de conversão, que ocorre sequencialmente nos canais e sem repetição, no modo

Sequence-of-channels (ADC12CONSEQ_1).

Dando sequência ao próximo registro, o sensor interno de temperatura é desligado no campo ADC12TCOFF e a resolução da conversão é escolhida em sua máxima qualidade disponível de 12 bits (ADC12RES_2).

Diferente dos anteriores, os próximos registros referem-se à memória do ADC12_A, sendo um para cada um dos 16 *buffers* de memória. Como utilizamos apenas dois canais do ADC, necessitamos apenas de dois *buffers* de memória. O registro ADC12MCTL0 controla o primeiro *buffer* e é selecionada a referência para leitura do valor como sendo AVcc para o valor máximo e AVss, o valor mínimo (ADC12SREF_0). O campo ADC12INCH não necessitaria ser modificado, permanecendo, assim, na configuração padrão, que lhe atribui o valor 0 e seleciona o canal de entrada A0 para esse *buffer* de memória. No caso foi atribuído zero apenas para uma melhor visualização dos registros.

O registro seguinte é igual ao anterior porém refere-se ao segundo *buffer* de memória, portanto recebe a mesma configuração para tensão de referência, com ADC12SREF_0. Porém este *buffer* é o último da sequência a ser utilizado, sendo então modificado o bit referente ao campo ADC12EOS, que significa *end of sequence*, para que o módulo não faça mais conversões nas entradas subsequentes de modo automático. Finalmente tem seu canal de entrada escolhido como o A1, portanto é dado o comando ADC12INCH_1. O último registro configurado habilita as interrupções dos canais utilizados, de modo que o ADC12IE é configurado como 1 nos campos ADC12IE0 e ADC12IE1.

4.3.2 Rotinas de interrupção do ADC

Listagem 4.5 – Código das rotinas de interrupção para o ADC

```
1  // Interrupt Service Routines (ADC process interrupts)
2
3  #pragma vector=TIMER2_A0_VECTOR // Interrupt Service Routine for TA2CCR0
4  __interrupt void TA2_CCR0_ISR(void){
5  }                                // Close TA2_CCR0_ISR
6
7  #pragma vector=TIMER2_A1_VECTOR
8  __interrupt void TA2_ISR(void){
9    switch(__even_in_range(TA2IV,14)){
10   case 0: break;
11   case 2: break;
12   case 4: break;
13   case 6: break;
14   case 8: break;
15   case 10: break;
16   case 12: break;
17   case 14:
18      ADC12CTL0 |= ADC12SC;              // Pulso do sinal ADC12SC
19      ADC12CTL0 &=~ADC12SC;
20      break;
```

```
21    default: break;
22    }
23
24  }
25
26  #pragma vector = ADC12_VECTOR
27  __interrupt void ADC12_ISR(void){
28    switch(__even_in_range(ADC12IV,36)){
29    case  0: break;                          // No interrupt pending
30    case  2: break;                                 // ADC overflow
31    case  4: break;                          // ADC timing overflow
32    case  6:                                 // ADC12IFG0
33       ADC12IFG &=~ ADC12IFG0;
34       break;
35    case  8:                                 // ADC12IFG1
36       clSignal = (unsigned int)             // clSignal assumes A0
37       ADC12MEM0;
38       fxSignal = (unsigned int)             // fxSignal assumes A1
39       ADC12MEM1;
40
41       if (pulseWidth>256)                   //(1/32768)*128*(1/2)=
42            pulseWidth=128;                  //;=DIST/[VEL. AR (340.29m/s)] =~66,5cm
43       multFactor= pulseWidth>>1;
44       cl        = (clSignal)>>2;            // deslocamento de bit do sinal cl
45       fx        = (fxSignal)>>2;            // deslocamento de bit do sinal fx
46       MPY       = multFactor;               // multiplicador 1: pulseWidth
47       OP2       = cl;                       // multiplicador 2: sinal cl deslocado
48       clPond    = (RESLO);                  // resultado 1: sinal cl na proporcao da
49                                             //;distancia
50       MPY       = 64 - multFactor;          // multiplicador 1: complemento de
51                                             //;pulseWidth
52       OP2       = fx;                       // multiplicador 2: sinal fx deslocado
53       fxPond    = (RESLO);                  // resultado 2: sinal fx na proporcao
             inversa de cl
54       outSignal= ((clPond+fxPond)>>3);     // soma balanceada dos resultados com
55                                             //;ganho 2
56
57  //Habilitacao das saidas
58       if (outSignal&BITB)
59          P6OUT |= BIT2;
60       else P6OUT &= ~BIT2;
61       if (outSignal&BITA)
62          P6OUT |= BIT3;
63       else P6OUT &= ~BIT3;
64       if (outSignal&BIT9)
65          P6OUT |= BIT4;
66       else P6OUT &= ~BIT4;
67       if (outSignal&BIT8)
68          P7OUT |= BIT0;
69       else P7OUT &= ~BIT0;
70       if (outSignal&BIT7)
71          P3OUT |= BIT6;
72       else P3OUT &= ~BIT6;
73       if (outSignal&BIT6)
```

```
74            P3OUT |= BIT5;
75          else P3OUT &= ~BIT5;
76          if (outSignal&BIT5)
77            P4OUT |= BIT1;
78          else P4OUT &= ~BIT1;
79          if (outSignal&BIT4)
80            P4OUT |= BIT2;
81          else P4OUT &= ~BIT2;
82          if (outSignal&BIT3)
83            P2OUT |= BIT7;
84          else P2OUT &= ~BIT7;
85          if (outSignal&BIT2)
86            P3OUT |= BIT2;
87          else P3OUT &= ~BIT2;
88          if (outSignal&BIT1)
89            P6OUT |= BIT6;
90          else P6OUT &= ~BIT6;
91          if (outSignal&BIT0)
92            P1OUT |= BIT6;
93          else P1OUT &= ~BIT6;
94          break;
95
96      case 10: break;                          // ADC12IFG2
97      case 12: break;                      // ADC12IFG3
98      case 14: break;                          // ADC12IFG4
99      case 16: break;                          // ADC12IFG5
100     case 18: break;                          // ADC12IFG6
101     case 20: break;                          // ADC12IFG7
102     case 22: break;                          // ADC12IFG8
103     case 24: break;                          // ADC12IFG9
104     case 26: break;                          // ADC12IFG10
105     case 28: break;                          // ADC12IFG11
106     case 30: break;                          // ADC12IFG12
107     case 32: break;                          // ADC12IFG13
108     case 34: break;                          // ADC12IFG14
109     case 36: break;                          // ADC12IFG15
110     default: break;
111     }                                    // Close switch-case
112 }
```

Para o correto funcionamento do ADC, optou-se por utilizar um *timer* que gera um pulso no bit ADC12SC que, como configurado anteriormente, ativa o sinal SHI e, portanto, realiza a amostragem e posteriormente a conversão. Assim, a frequência de amostragem depende diretamente desse *timer*. Quando há a interrupção de do CCR0, o pulso no ADC12SC é gerado.

A próxima rotina de interrupção trata o vetor ADC12IV, que detecta a origem da interrupção de memória do ADC. Ou seja, essa interrupção indica que uma memória foi preenchida com um valor de amostragem. Para este caso só é necessário que, quando o segundo sinal da sequência for convertido e armazenado, duas variáveis recebam os valores correspondentes às amostras de ambos sinais. A partir daí, é testada a distância medida pelo sensor, que se

for maior do que um certo ponto, não é útil para o sistema (sabe-se que um braço de uma guitarra geralmente não mede um metro do *headstock* até o encaixe no corpo, por exemplo). Caso não fosse limitada, uma distância muito grande causaria *overflow* na saída do sinal, pois este é diretamente proporcional à distância medida. Portanto, em caso de erro na distância aferida (para além de certo tamanho), não haverá um impacto significativo na proporção dos sinais.

Assim, tem-se ambos sinais amostrados. Logo em seguida são feitos deslocamentos de bits em ambos os sinais processados, para que quando multiplicados e somados não ocorra *overflow* na saída. Em seguida é feita a multiplicação utilizando os registros do multiplicador em *hardware* e então é obtido o resultado da saída, que será convertida para uma tensão analógica posteriormente. Esse resultado é deslocado 1 bit a mais à esquerda para, além de compensar os deslocamentos prévios, causar um ganho no sinal. Assim, ocorrendo de maneira sucessiva, temos os sinais analógicos sendo amostrados, somados e multiplicados proporcionalmente para serem encaminhados para o conversor digital analógico.

4.4 O problema e a solução para o DAC

A etapa final do projeto do misturador é converter o sinal digital de 12 bits para um sinal analógico novamente, em nível de linha, para que possa ser conectado a outros pedais de efeitos ou mesmo diretamente amplificado. Ocorre que, infelizmente, o microcontrolador MSP430F5529LP não possui um módulo de conversão digital-analógico, sendo necessário, portanto, que essa conversão seja realizada externamente, aumentando as dimensões do produto. Existem módulos conversores disponíveis no mercado de diversas tecnologias, porém devido a dificuldade de encontrá-los e seu custo, foi optado pela construção de um modelo simples porém bastante funcional de rede R-2R que será detalhado a seguir.

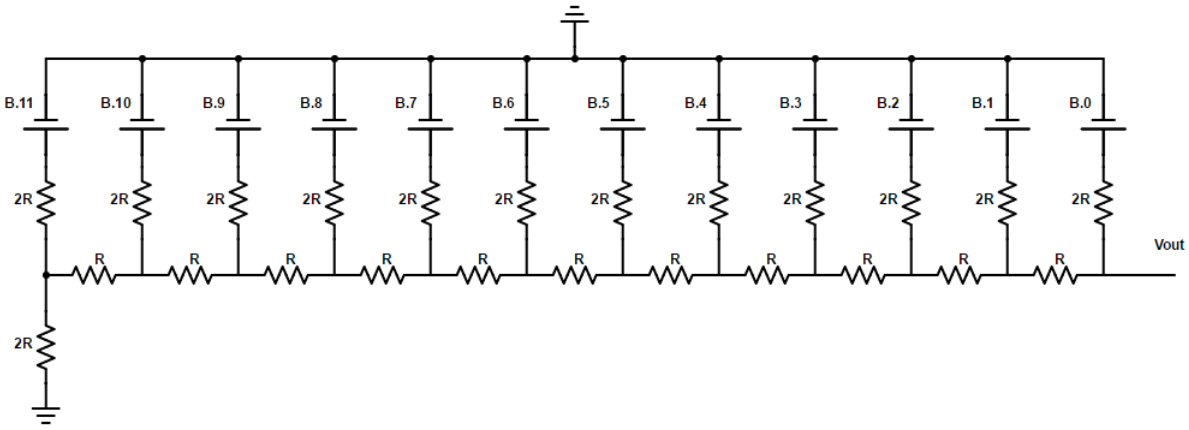
4.4.1 Rede R-2R

Um dos métodos mais simples para realizar uma conversão digital-analógico é com a chamada *rede escada R-2R*. Ele leva em conta a influência de cada bit na composição do sinal, do bit mais significativo (MSB) até o menos significativo (LSB). Para 12 bits, tem-se (2^12) posssibilidades, ou seja, 4096 degraus diferentes. Assim, a menor variação possível de tensão no sinal analógico na saída é exatamente o menor passo: ${}^1/_{4096}V_{dd}$. Essa menor variação é representada pelo bit menos significativo, nesta seção definido como o bit B.11.

Isso significa que se B.11 for ativado, será somada uma tensão equivalente ao menor degrau possível ao sinal de saída V_{out}. Por conseguinte, sendo o segundo bit (B.10) duas vezes mais significativo que o primeiro (B.11), sua variação terá um peso duas vezes maior sobre V_{out}. Assim, ativar o segundo bit é equivalente a somar ${}^2/_{4096}V_{dd}$ ao sinal de saída. Baseado nessa lógica, o circuito R-2R é esquematizado conforme a Figura 4.5.

A análise de seu funcionamento é feita com base em 2 princípios: o equivalente de

Figura 4.5 – Rede R-2R.

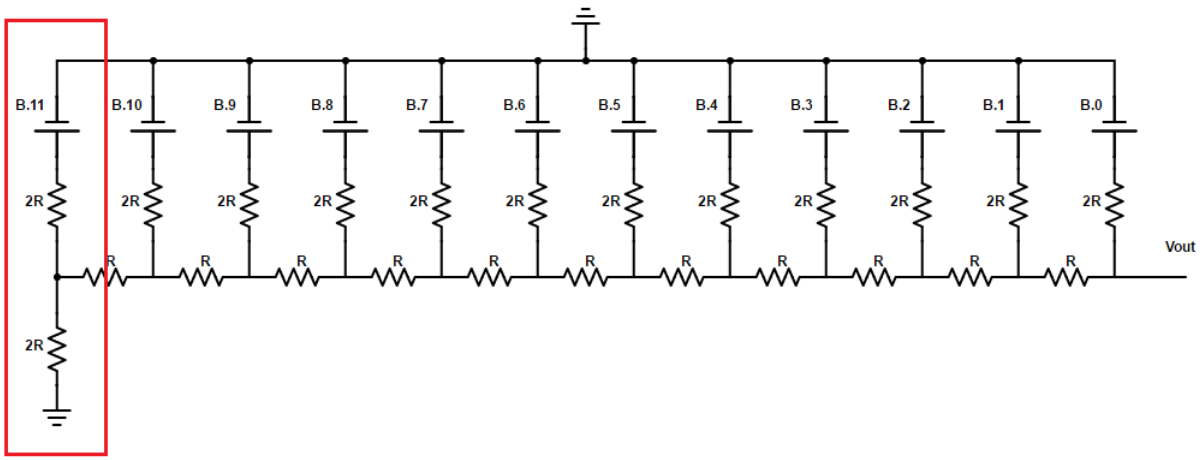


Fonte: Elaborado pelo próprio autor.

Thévenin e a superposição. Assim, será analisado cada bit separadamente (colocando-se outras fontes de tensão em curto com o comum para que não hava diferencial de potencial em seus polos) segundo o teorema de Thévenin.

Para começar a análise, espera-se obter tensão e resistência equivalente no primeiro nó do circuito, dentro da zona demarcada na Figura 4.6.

Figura 4.6 – Análise do primeiro nó da malha R-2R.



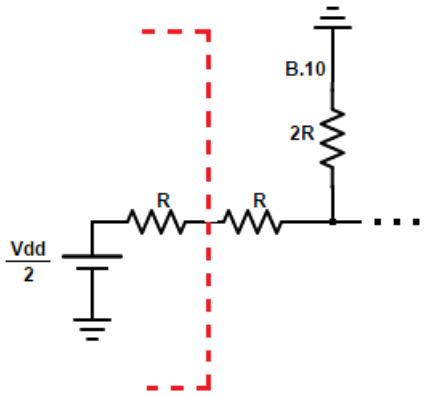
Fonte: Elaborado pelo próprio autor.

Apenas 2 valores de resistência são encontrados nesse circuito, sendo a mais baixa R, e a mais alta o dobro de R: 2R (daí o nome de rede R-2R para este conversor). Assim, o simples divisor de tensão com 2 resistores iguais contido na seção vermelha resulta em uma tensão $^{V_{dd}}/_{2}$ no nó entre os dois resistores de valor 2R.

Analisando a partir do nó em questão e aterrando-se todas as fontes, tem-se que a resistência vista da saída é o paralelo entre os dois resistores semelhantes 2R. Ou seja, a resistência equivalente vale $^{2R}/_{2} = R$.

O equivalente de Thévenin da primeira zona é demonstrado na Figura 4.7, estando à esquerda da zona delimitada em vermelho. Seu acoplamento ao resto da rede também está representado nessa figura.

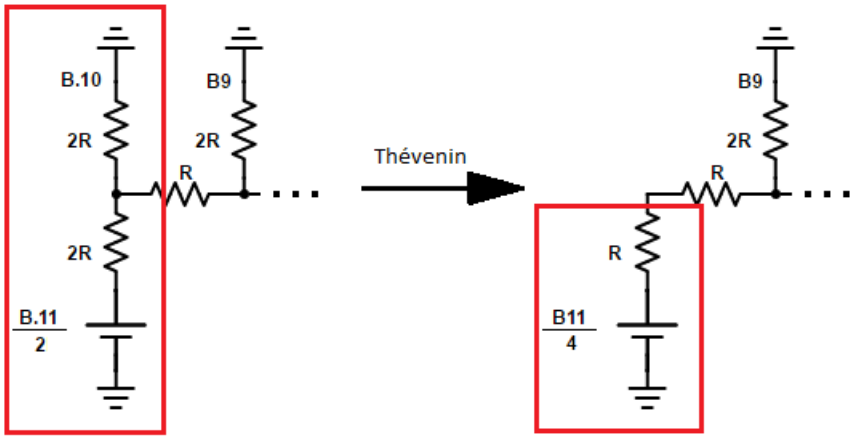
Figura 4.7 – Equivalente de Thévenin no primeiro nó da rede R-2R.



Fonte: Elaborado pelo próprio autor.

É interessante perceber que o resistor equivalente obtido pode ser associada ao próximo resistor, de mesmo valor R, tornando-se um único resistor 2R. Pela Figura 4.7, percebe-se que esse resistor equivalente é acoplado em paralelo a um resistor de valor igual, 2R, associado ao bit B.10. Portanto, a análise anterior pode ser repetida no nó da Figura 4.7, obtendo-se, assim, a metade da tensão ali aplicada, confome demonstrado na Figura 4.8.

Figura 4.8 – Equivalente de Thévenin no segundo nó da rede R-2R.



Fonte: Elaborado pelo próprio autor.

É de se esperar, portanto, que a tensão exercida pelo bit B.11 na saída ($Vout$) seja, seguindo-se a lógica das análises dos nós anteriores, igual a $\frac{V_{B.11}}{2^{12}}$. Isso significa, portanto, que B.11 tem o peso do menor degrau possível ($^{V_{dd}}/_{4096}$).

De fato, para cada bit $N, 0 < N < 11$, temos que a tensão exercida por tal bit no sinal de saída é dada por $V_{B.N|V_{out}} = \frac{V_{B.N}}{2^{N+1}}$, o que satisfaz a condição dos pesos de cada bit.

Segundo o princípio da superposição, realiza-se a análise para cada bit separadamente e somam-se os resultados respectivos. O sinal de saída é dado, então, pela Eq. 4.3.

$$V_{out} = \frac{V_{B.11}}{2^{12}} + \frac{V_{B.10}}{2^{11}} + \frac{V_{B.9}}{2^{10}} + \frac{V_{B.8}}{2^{9}} + \frac{V_{B.7}}{2^{8}} + \frac{V_{B.6}}{2^{7}} + \frac{V_{B.5}}{2^{6}} + \frac{V_{B.4}}{2^{5}} + \frac{V_{B.3}}{2^{4}} + \frac{V_{B.2}}{2^{3}} + \frac{V_{B.1}}{2^{2}} + \frac{V_{B.0}}{2^{1}}, \tag{4.3}$$

que pode ser simplificada como disposto na Eq. 4.4.

$$V_{out} = \frac{V_{B.11}}{4096} + \frac{V_{B.10}}{2048} + \frac{V_{B.9}}{1024} + \frac{V_{B.8}}{512} + \frac{V_{B.7}}{256} + \frac{V_{B.6}}{128} + \frac{V_{B.5}}{64} + \frac{V_{B.4}}{32} + \frac{V_{B.3}}{16} + \frac{V_{B.2}}{8} + \frac{V_{B.1}}{4} + \frac{V_{B.0}}{2} \tag{4.4}$$

Considerando que as tensões de cada bit só podem assumir valores V_{dd} ou 0, testamos os limites do sinal de saída. A Eq. 4.5 o faz para o valor máximo, V_{dd}; já a Eq. 4.6, para o valor mínimo, 0, como pode ser visto a seguir.

$$\begin{aligned} V_{out_M} = {} & \frac{V_{dd}}{4096} + \frac{V_{dd}}{2048} + \frac{V_{dd}}{1024} + \frac{V_{dd}}{512} + \frac{V_{dd}}{256} + \frac{V_{dd}}{128} + \frac{V_{dd}}{64} + \\ & + \frac{V_{dd}}{32} + \frac{V_{dd}}{16} + \frac{V_{dd}}{8} + \frac{V_{dd}}{4} + \frac{V_{dd}}{2} = \frac{4095}{4096} V_{dd} \approx V_{dd} \end{aligned} \tag{4.5}$$

$$V_{out_m} = \frac{0}{4096} + \frac{0}{2048} + \frac{0}{1024} + \frac{0}{512} + \frac{0}{256} + \frac{0}{128} + \frac{0}{64} + \frac{0}{32} + \frac{0}{16} + \frac{0}{8} + \frac{0}{4} + \frac{0}{2} = 0 \tag{4.6}$$

Portanto, as equações 4.5 e 4.6 confirmam as condições do sinal de saída e cada bit tem um peso específico sobre este. Confere-se, assim, a validade da rede R-2R na resolução do problema de conversão digital-analógico.

4.4.2 A resolução do problema

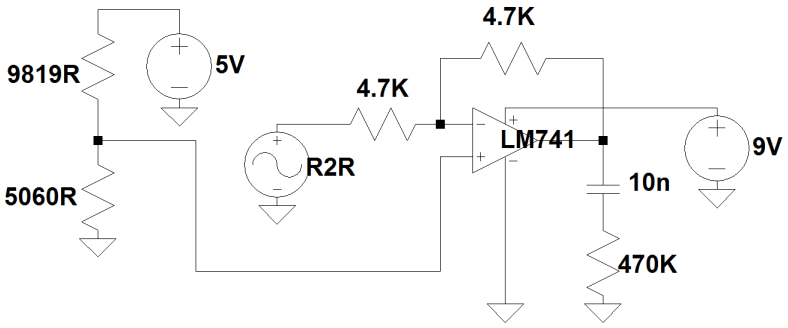
Para se lidar com a falta de um conversor digital-analógico integrado ao controlador, foi utilizada uma rede escada R-2R à saída do MSP430, de modo que cada bit resultante da conversão ADC foi alocado como saída em um pino específico de *input*/*output*. Desse modo, dois novos problemas surgiram e tiveram de ser resolvidos:

- há um ruído inerente à malha R-2R que causa espúrios significativos ao sinal; e
- a impedância de saída é muito alta para ser acoplada ao restante do sistema, resultando em baixa corrente.

Assim, optou-se por utilizar um *buffer* de saída com o circuito integrado LM741. Como

o buffer desacopla o circuito de entrada do circuito de saída, isto é, os separa, o problema da impedância passa a ser menos significativo. Além disso, a corrente na saída do circuito tende a ser maior. Nas configurações utilizadas, conforme a Figura 4.9, percebe-se que a retroalimentação negativa (amplificador operacional em configuração inversora) unitária provoca um ganho de $\left(-{}^{4.7K}/_{4.7K}\right) = -1$, ou seja, o ganho é unitário e há inversão do sinal.

Figura 4.9 – Circuito do *buffer* de saída da rede R-2R.



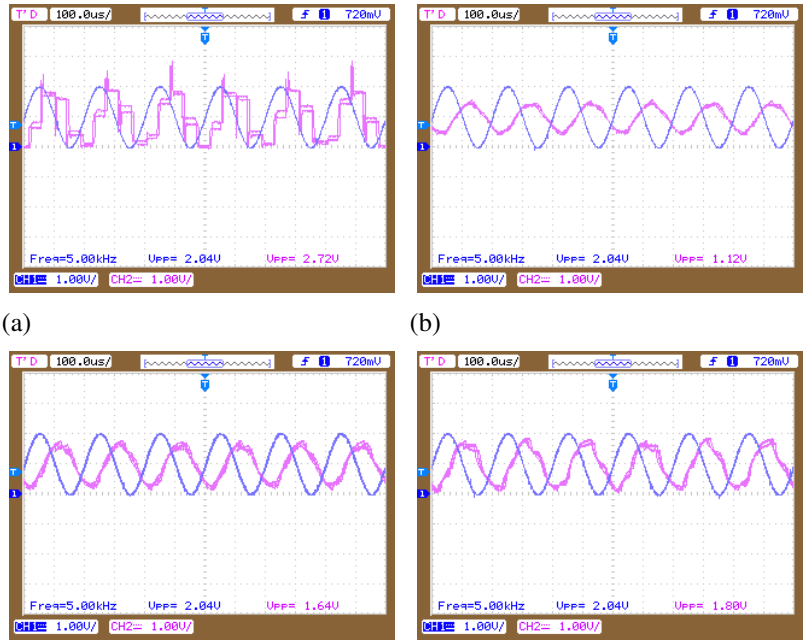
Fonte: Elaborado pelo próprio autor.

Em laboratório é fácil obter uma tensão simétrica para alimentação do LM741, porém como isso é mais difícil de ser realizado em um sistema embarcado, devido à utilização de bateria, foi dada preferência ao uso de um *offset* no sinal de entrada por meio de *virtual grounding* para que este problema fosse sanado. No caso, como o ganho é unitário, o sinal de corrente contínua em um dos lados do amplificador operacional não causa saturação da saída. Portanto, ao pino $V+$ do CI foi acoplado um terra virtual. Além disso, por mais que os níveis de tensão mínima de alimentação recomendados no *datasheet* para operação do LM741 fossem de $\pm 10V$, foi verificado em laboratório que uma tensão de $+9V$ era suficiente para um ótimo desempenho do amplificador nessas configurações, o que facilita o uso desse circuito com uma bateria de 9V.

Grande parte do ruído e da presença de imperfeições no sinal devido a todo processo de digitalização e conversão do sinal para analógico novamente, pode ser amenizado com o uso de filtros. Disponíveis no laboratório, os capacitores de 10 nF foram utilizados como um filtro RC simples, passa-baixas. Para obter-se diferentes valores, foram associados dois em série e também 3 em série, alcançando os valores de 5 nF e 3.33 nF respectivamente.

Assim foi realizado um teste com o filtro RC na saída utilizando os valores de capacitância $10nF, 5nFe3, 3nF$. O *LaunchPad* foi programado para amostrar o sinal senoidal de um gerador de funções e reproduzi-lo diretamente, sem qualquer processamento no conversor digital-analógico. Com o auxílio de um osciloscópio, foi possível observar como mostra a Figura 4.10 (a) o sinal de saída com os problemas apontados, e nas seguintes figuras, 4.10 (b), 4.10 (c) e 4.10 (d), uma visível melhora. O melhor valor observado para a capacitância do filtro é apresentado pela figura 4.10 (d), onde o valor é de $3.3nF$. Tal valor é suficiente para retirar grande parte de ruído sem prejudicar demasiadamente o sinal. Esses testes foram feitos com a taxa de amostragem um pouco inferior à utilizada no protótipo final.

Figura 4.10 – Testes com filtragem sobre o sinal de saída. Sinal de entrada no conversor ADC_12A (azul) e saída (rosa) da rede R2R (a) sem filtro, (b) com filtro RC (C = $10nF$), (c) com filtro RC e (C = $5nF$) e (d) com filtro RC e (C = $3.3nF$)



(a) (b)

(c) (d)

Fonte: Elaborado pelo próprio autor.

4.5 Produto final

Foram implementadas todas as partes propostas no diagrama de blocos, sendo:

1. A conversão analógico-digital de 12 bits de 2 sinais distintos;
2. O sensoriamento da distância por meio do sensor ultrassônico HC-SR04;
3. A mistura ponderada dos dois sinais conforme a distância; e
4. O conversor digital-analógico de 12 bits em modelo escada R-2R, junto a um *buffer* de saída com filtro passivo RC de primeira ordem.

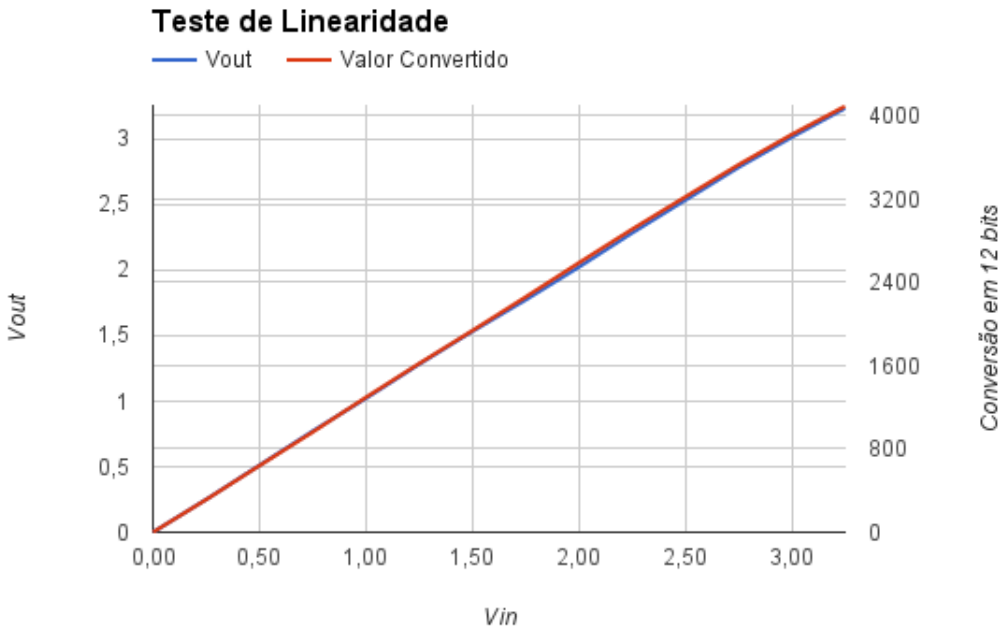
Antes que tais partes fossem testadas em conjunto como o protótipo proposto por este trabalho, alguns outros testes foram realizados. Úteis não somente para demonstração do bom funcionamento das partes do protótipo, servem também de documentação do resultado obtido com toda a implementação apresentada. Abordando aspectos relevantes de algumas partes até finalmente o protótipo inteiro, seguem os resultados obtidos.

4.5.1 Teste de linearidade do ADC e DAC

O teste de linearidade é feito alimentando a entrada do conversor AD com uma tensão contínua e tomando nota do valor por ele atribuído na memória do microcontrolador para tal

entrada. Ao mesmo tempo para conferir a linearidade do DAC, mede-se a saída do mesmo, que é programada para reproduzir o valor amostrado pelo ADC e salvo temporariamente no microcontrolador, sem qualquer processamento adicional. Esse processo é feito para vários níveis de tensão, do mínimo ao máximo aceito pelo conversor AD. Com tais medidas, pode ser conferida a linearidade do processo de conversão analógico-digital e vice-e-versa.

Figura 4.11 – Teste de linearidade do ADC e DAC, com passo de 250 mV



Fonte: Elaborado pelo próprio autor.

A Figura 4.11 mostra que tanto o valor obtido na conversão AD quanto a tensão na saída da conversão DA são lineares em relação à entrada do ADC. Há uma reta para o valor da conversão e outra para a saída do DAC, ambas em relação a tensão de entrada. As retas se sobrepõem com uma discrepância muito pequena.

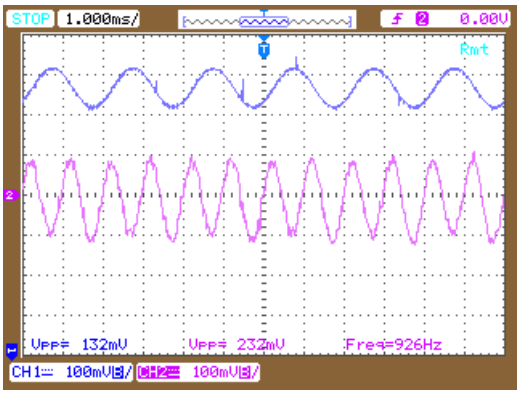
Em algumas medidas houve uma diferença de no máximo 30 mV entre o valor de tensão da entrada e da saída, que não comprometeu a linearidade dos processos de conversão. Tal erro pode ser atribuído à precisão dos resitores utilizados, que é de até 5% segundo o fabricante. Supondo que alguns desses resistores possuam resistência distinta da nominal próxima ao limite da precisão especificada enquanto outros teriam seu valor real mais próximo do nominal, é plausível pensar que para certos valores de conversão, onde há certo bit em nível alto que passa pelos resistores de valores menos precisos, ocorra tal erro na saída do DAC. Para outros valores de conversão cujo bit que passa pelos resistores mais errôneos seja de nível baixo, tal erro seria inexistente ou menor. Por isso então seria possível observar o erro em apenas algumas medidas.

4.5.2 Teste da saída: soma dos sinais ponderada pelo sensor e taxa de amostragem

Para testar a parte relacionada à soma ponderada dos sinais, é necessário processar os sinais amostrados diante da resposta do sensor fornecida ao microcontrolador, antes de enviá-lo ao DAC para se tornar analógico novamente. Com a parte de conversões tanto AD como DA funcionando bem, como foi mostrado no teste de linearidade, adicona-se o processamento da informação dada pelo sensor de modo a ponderar a soma dos sinais amostrados.

Com tal implementação preparada como foi detalhada anteriormente neste mesmo capítulo, o teste foi feito utilizando-se dois geradores de função para fornecer os sinais às entradas, e um osciloscópio para analisar a saída. Ambos geradores de função foram ajustados para ondas senoidais, uma com o dobro de frequência da outra, em 500 Hz e 1 kHz. Assim, movimentando-se um anteparo diante do sensor ultrassônico, pode ser observada a saída como uma onda senoidal de 500 Hz com a mão na distância máxima do sensor, uma onda de 1 kHz, na distância mínima, ou ainda a soma das duas em diferentes proporções de acordo com a distância da mão dentre os limites máximo e mínimo. Como esperado, o resultado é mostrado nas figuras 4.12, 4.13, 4.14 e 4.15, para as distâncias respectivas de 2, 20, 30 e 60 cm do sensor ao anteparo. Nessas figuras, o sinal de saída observado está centralizado na tela do osciloscópio na cor rosa, enquanto o sinal mostrado na parte superior em azul é a entrada de 500 Hz, utilizada apenas como referência para a comparação com o sinal de saída.

Figura 4.12 – Sinal de entrada e de saída com objeto a 2 cm do sensor

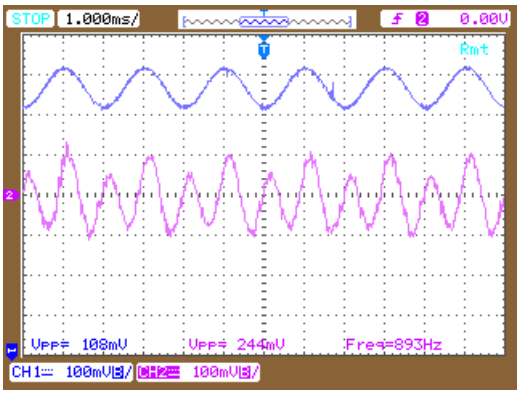


Fonte: Elaborado pelo próprio autor.

Antes que a alimentação das entradas com os sinais senoidais fosse ligada, foram medidas a entrada e a saída para se constatar os ruídos gerados pelo sistema. Como pode ser visto na Figura 4.16, os ruídos medidos na entrada e na saída são praticamente iguais, com tensão máxima de 40 mV. Logo, foi observado que o ruído presente na saída está presente em grande parte também na entrada e tem origem de todo o sistema em geral, além de que é tolerável e não interfere no objetivo da aplicação.

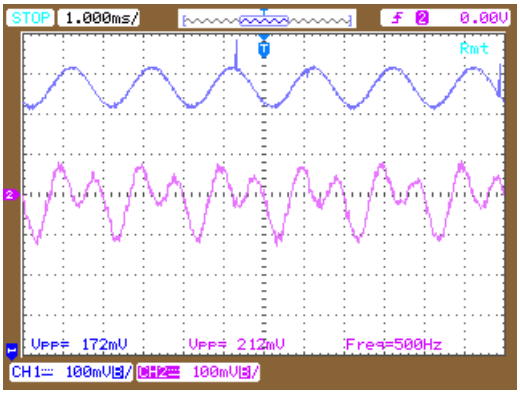
Ainda analisando a saída, há o interesse em medir a taxa de amostragem obtida. A

Figura 4.13 – Sinal de entrada e de saída com objeto a 20 cm do sensor



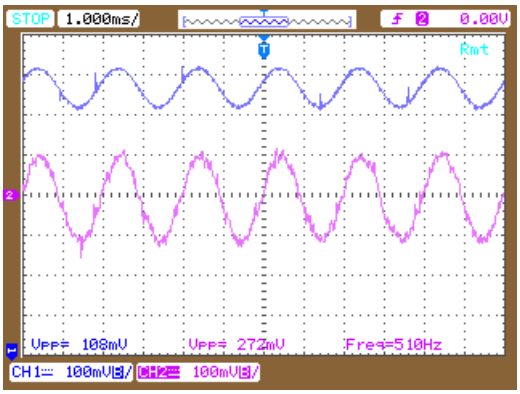
Fonte: Elaborado pelo próprio autor.

Figura 4.14 – Sinal de entrada e de saída com objeto a 30 cm do sensor



Fonte: Elaborado pelo próprio autor.

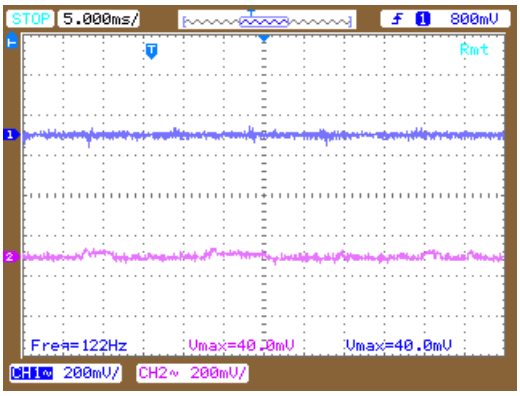
Figura 4.15 – Sinal de entrada e de saída com objeto a 60 cm do sensor



Fonte: Elaborado pelo próprio autor.

amostragem tem sua frequência definida pelo *timer* (pelo qual é acionado o pulso SHI), mas pode sofrer influência de outros fatores como o número de ciclos de *clock* na amostragem definidos na fórmula 4.2 e o número de canais utilizados, por exemplo, além da duração das rotinas de interrupção. Utilizando apenas uma das entradas com a tensão senoidal de 1 kHz, é possível perceber os degraus de cada valor amostrado e medir o real período de amostragem e por consequência, a frequência. Dessa maneira a Figura 4.17 foi obtida, onde

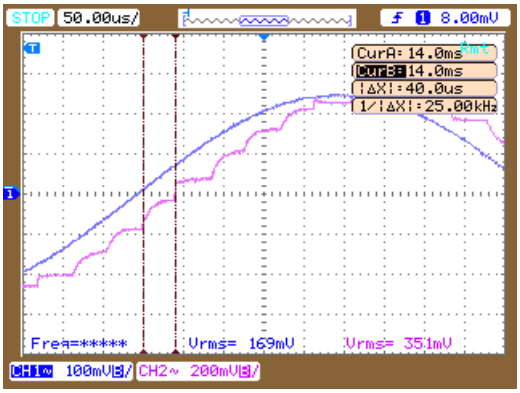
Figura 4.16 – Ruído de entrada (na parte superior em azul) e ruído de saída (na parte inferior em rosa



Fonte: Elaborado pelo próprio autor.

é possível tomar para tal, o valor de 25 kHz medido na saída em cor rosa, que é inferior à desejada (44100 Hz), mas ainda suficiente para o protótipo.

Figura 4.17 – Saída na cor rosa do conversor DA de um sinal senoidal de 1kHz (em azul) amostrado



Fonte: Elaborado pelo próprio autor.

4.5.3 O teste do protótipo completo

Nos testes realizados com o protótipo completo, utilizou-se um pedal *Chromatic Tuner TU-2* da fabricante BOSS para dividir o sinal da guitarra. Esse pedal possui uma entrada, que foi ligada à guitarra, e 2 saídas: uma *output* comum e outra do tipo *bypass* (a primeira possui como função interromper a passagem do sinal quando o afinador é utlizado para oferecer uma afinação silenciosa, enquanto a segunda está sempre ativa). Ambas saídas são de baixa impedância (aproximadamente 10 $k\Omega$) controlada por buffers dentro do pedal. Por esse motivo, o TU-2 foi escolhido para que não houvesse perdas significativas no processo da divisão do sinal.

O sinal *output* do primeiro pedal foi ligado a um outro pedal da mesma fabricante, modelo *OS-2*. Esse pedal é responsável por distorcer o sinal e, para que houvesse uma transição

Figura 4.18 – Montagem dos pedais

Fonte: Elaborado pelo próprio autor.

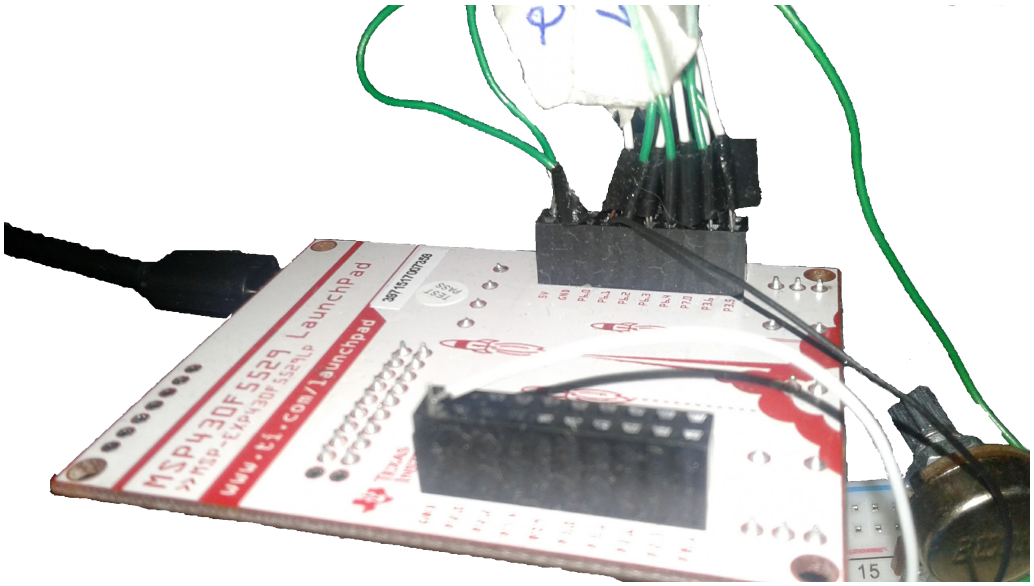
agradável na experimentação, ajustou-se o botão *Level* para que o nível do sinal de saída se assemelhasse ao nível da outra saída (do afinador). Isso para que a diferença obtida entre os sinais fosse apenas de timbre e não também de tensão (o nível de intensidade sonora, ou volume, mensurado pode ser naturalmente diferente devido ao timbre diferente dos 2 sinais). Os outros botões foram ajustados para obter um timbre distorcido comum, bastante utilizado, com um nível moderado de distorção. Essa montagem é ilustrada na Figura 4.18.

Saindo dos pedais, o sinal passa por um circuito formado por um capacitor de desacoplamento e um divisor de tensão, responsável por adicionar um nível de *offset* ao sinal. Tal circuito é detalhado na Figura 4.3, porém para os testes, foi utilizado um potenciômetro linear de 100 kkΩ para fazer o divisor de tensão, e possiblitar melhor ajuste prático.

Os sinais de saída dos pedais, agora com nível DC, são conectados à pinos individuais do MSP430, sendo um para cada sinal e respectivo canal de conversão. Após o processamento dos sinais, a saída em 12 bits é conectada ao conversor digital-analógico. Tais conexões com o *LaunchPad* são ilustradas na Figura 4.19. Um dos modelos de DAC construídos para este protótipo é ilustrado na Figura 4.20, e apesar de terem sido utilizados dois resistores em paralelo para obter a metade do valor de um, apresentou bom funcionamento. Nota-se ainda um capacitor na saída para remover a componente DC do sinal, para que possa seguir a um amplificador de guitarra comum. Nesse caso de teste não foi utilizado o filtro passivo RC passa-baixas, e o ruído apresentado não foi o suficiente para se tornar um problema.

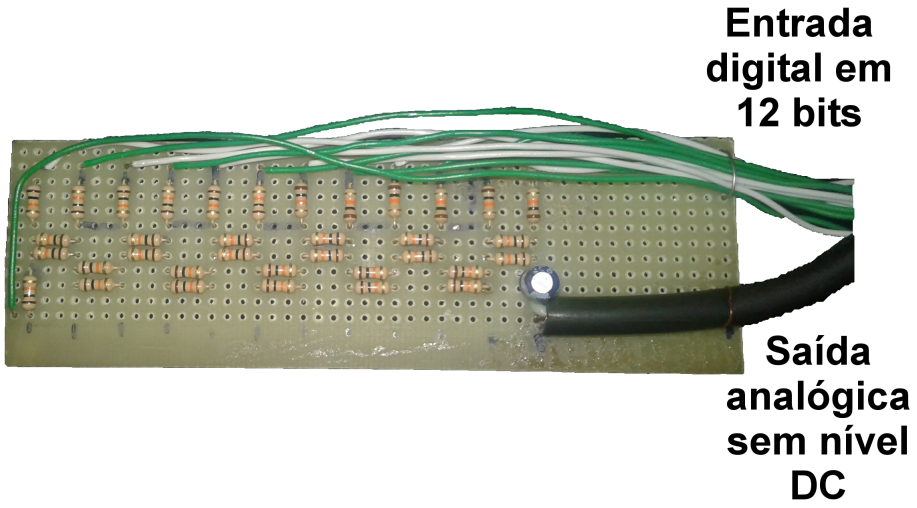
Com o sensor ultrassônico devidamente posicionado na guitarra como na Figura 3.2, pôde-se realizar finalmente o teste do protótipo funcionando conforme a sua utilidade desejada, descrita na elaboração do projeto. E, como esperado, ao se tocar a guitarra em diferentes regiões de seu braço, diversos timbres foram obtidos. Numa situação que é comum para diversas músicas, ao serem executados acordes na região mais próxima do sensor, obteve-se

Figura 4.19 – Conexões ao MSP430 através do *LaunchPad*



Fonte: Elaborado pelo próprio autor.

Figura 4.20 – DAC de rede R-2R com resistores em paralelo



Fonte: Elaborado pelo próprio autor.

um som mais definido e limpo, adequado para tal ocasião. Assim que o executante passou para uma região mais longe do sensor para executar um solo, o timbre ficou mais distorcido, também ideal à ocasião.

Em outra ocasião, repetiu-se esse teste final com um pedal de eco (*delay*) no lugar da distorção. O efeito obtido foi bastante interessante. Da mesma forma que com o pedal de distorção, o eco se tornou mais intenso à medida que a mão do usuário se distanciava do sensor, deixando o som menos definido e com mais profundidade.

Capítulo 5

Conclusão

Este presente trabalho desde o princípio propôs o estudo de tecnologias disponíveis para uma aplicação em áudio. Com o uso de um sensor, seria feito o controle da mistura de dois sinais analógicos de áudio. Para isso, muitas ideias foram planejadas e testadas ao longo do desenvolvimento. Diferentes maneiras se mostraram possíveis, outras nem tanto, e assim as melhores medidas foram adotadas. Dentre todos os estudos, testes e pesquisas feitas, foi possível chegar a algumas conclusões acerca dos problemas enfrentados, como a qualidade do sinal de saída e os acertos, como a funcionalidade e inovação.

5.1 Tecnologia Musical

Na grande área da tecnologia musical, os estudos feitos mostram o quão grande é esse espaço e a importância dos avanços já concluídos e hoje imortalizados. São esses os motivos que cada vez mais incentivam o crescimento dos estudos dessa área. Com o surgimento de novas tecnologias a cada dia, é necessário sempre estudá-las e quando possível aplicá-las às mais diversas utilidades, como fez este trabalho, com as ferramentas disponíveis.

Existe uma certa dificuldade em alguns casos de integrar tecnologia à música. Muitas dessas dificuldades surgem do lado técnico da tecnologia, o que afasta usuários leigos que não se sentem à vontade em tal ambiente. Isso ocorre desde as primeiras inovações tecnológicas e suas entradas no cotidiano musical. Acredita-se que com o melhoramento da interface com o usuário, essa dificuldade diminui. Com este trabalho tem-se um produto que interage com o usuário de maneira simples o suficiente para sua utilização, bem como trazendo um resultado relevante o suficiente para sua utilização.

5.2 Desenvolvimento do Projeto

Com a crescente popularidade e disponibilidade dos *LaunchPads* viu-se a oportunidade de desenvolvimento de uma ideia. Seus módulos independentes acoplados a um CPU possibilitam uma integração entre sinais analógicos e digitais com funções que cumprem o papel necessário para o desenvolvimento do presente projeto.

Mesmo assim, toda aplicação de certa tecnologia a uma função desejada implica em implementações e adaptações. Para isso é necessário, além do estudo, muitos testes como os feitos neste trabalho. Durante todo o decorrer da implementação para chegar ao produto final, muitas decisões foram tomadas a partir de resultados desses testes.

A começar pelo sensoriamento, foi atestado que a escolha da sua tecnologia de acordo com a aplicação é fundamental. O sensor infravermelho foi totalmente descartado de uso quando mostrou-se ineficiente para a situação desejada. Tão importante quanto, é a possibilidade de utilizar a resposta do sensor no objetivo desejado. O MSP430F5529 é provido de todas as ferramentas necessárias para o funcionamento com um sensor ultrassônico, que se mostrou bastante adequado à proposta deste trabalho. Uma parte importante para o prosseguimento deste projeto é o direcionamento do sensor e seu ajuste no braço. É de vital importância que se procure sensores mais compactos e que se construa um mecanismo regulável de posicionamento do componente no braço do instrumento, que ao mesmo tempo seja facilmente acoplado e desacoplado.

Na parte de conversão analógica-digital, o modo sequencial em repetição das conversões seria ideal para um sinal de áudio, já que demanda muitas amostras em pouco período de tempo, porém ao mesmo tempo impedia que a tarefa do sensoreamento fosse realizada. Assim, o protótipo foi engendrado de tal maneira que a conversão fosse controlada de outro jeito, permitindo que rotinas de outras funções fossem cumpridas.

A conversão digital-analógica é uma das partes do projeto que foi realizada externa ao LaunchPad. A ausência de um conversor desse tipo no modelo utilizado implicou na implementação de maneira simples com uma malha de resistores conhecida como rede R-2R. Outra maneira de realizar essa conversão seria com o uso de PWM, porém esse método apresentaria resultado pior do que com a rede de resistores para as frequências dos sinais trabalhados. A malha se mostrou suficientemente boa pois a conversão foi feita, ainda que a presença de ruídos (térmicos e oriundos de interferência) tenha diminuído um pouco sua qualidade.

Não somente na malha R-2R mas em todo o projeto a presença de ruídos também da natureza digital é um fator a diminuir a qualidade do resultado. Eis uma das maiores dificuldades de se trabalhar em áudio principalmente no nível de protótipo, pois a construção, os elementos e todo o projeto em si sofrem interferência na presença desses ruídos, que são sempre indesejáveis. Assim, para futuros trabalhos é interessante que se utilize um conversor AD com maior profundidade de palavra, e um conversor DA em um único encapsulamento (*chip*

pré-fabricado) também favorece a organização e o resultado da conversão em si. Uma outra possibilidade é realizar a substituição da conversão digital por um circuito completamente analógico, utilizando um integrador para o sinal advindo do sensor ultrassônico (ponderação) e realizando a soma de forma também analógica.

Apesar dos problemas apresentados durante todo o trabalho, fica evidente que, ainda que com suas limitações, a ideia pôde ser concluída. Para um protótipo construído para a execução de uma ideia, o projeto desenvolvido teve um desempenho bastante satisfatório com os recursos aplicados.

Referências Bibliográficas

[Academia Musical 2014] Academia Musical (2014). A anatomia da guitarra. Disponível em: <http://www.academiamusical.com.pt/tutoriais/instrumentos/guitarra/anatomia-da-guitarra/>. Acesso em: dezembro de 2016.

[Carvalho et al. 2014] Carvalho, L. N., Brandão, M., and Lamar, M. V. (2014). *Implementação em FPGA de um sintetizador de áudio polifônico de baixa complexidade*. São Paulo, Brasil. 12o Congresso de Engenharia de Áudio 18a Convenção Nacional da AES Brasil.

[Davies 2008] Davies, J. H. (2008). *MSP430 Microcontroller Basics*. Newnes, Burlington, Estados Unidos da América.

[Elec Freaks] Elec Freaks. Ultrasonic ranging module hc-sr04. Datasheet disponível em <www.elecfreaks.com>. Acesso em: agosto de 2016.

[Fondazione Renzo Piano] Fondazione Renzo Piano. *Story - IRCAM - Institut de Recherche et Coordination Acoustique/Musique*. Disponível em: <http://www.fondazionerenzopiano.org/project/118/ircam-institut-de-recherche-et-coordination-acoustique-musique/genesis/?l=en>. Acesso em: novembro de 2016.

[Françoise et al. 2013] Françoise, J., Schnell, N., and Bevilacqua, F. (2013). *Gesture-based Control of Physical Modeling Sound Synthesis: a Mapping-by-Demonstration Approach*. Barcelona, Espanha. Proceedings of the 21st ACM international conference on Multimedia (MM'13).

[Leroy et al. 2006] Leroy, N., Fléty, E., and Bevilacqua, F. (2006). *Reflective Optical Pickup for Violin*. Paris, França. 6th International Conference on New Interfaces for Musical Expression (NIME).

[Maxim Integrated Products, Inc.] Maxim Integrated Products, Inc. Understanding sar adcs: Their architecture and comparison with other adcs. Disponível em: <https://www.maximintegrated.com/en/app-notes/index.mvp/id/1080>. Acesso em: dezembro de 2016.

[Mendonça and Zelenovsky 2004] Mendonça, A. and Zelenovsky, R. (2004). Eletrônica digital: Curso prático e exercícios.

[Ünsalan and Gürhan 2014] Ünsalan, C. and Gürhan, H. D. (2014). *Programmable Microcontrollers with Applications: MSP430 LaunchPad with CCS and Grace*. McGraw-Hill Education, Nova Iorque, Estados Unidos da América.

[Rasamimanana et al. 2011] Rasamimanana, N., Bevilacqua, F., Schnell, N., Fléty, E., and Zamborlin, B. (2011). *Modular Musical Objects Towards Embodied Control Of Digital Music*. Funchal, Portugal. 5th International Conference on Tangible, embedded, and embodied interaction.

[Rocha 2010] Rocha, F. (2010). *Questões de Performance em Obras Eletrônicas Mistas*. Florianópolis, Brasil. XX CONGRESSO DA ANPPOM.

[Rockwell Automation and Allen-Bradley 1999] Rockwell Automation and Allen-Bradley (1999). *Fundamentos de Detecção de Presença*. Corporação Internacional Rockwell, São Paulo, Brasil.

[Said-Moorhouse 2015] Said-Moorhouse, L. (2015). Imogen heap's sci-fi gloves make anyone a musician. Disponível em: <http://edition.cnn.com/2015/01/12/technology/imogen-heap-mimu-music-gloves-blk/>. Acesso em: novembro de 2016.

[Texas Instruments 2009] Texas Instruments (2009). *MSP430x5xxx Family User's Guide (SLAU208D)*. Texas Instruments Incorporated, Dallas, Estados Unidos da América.

[Texas Instruments 2013] Texas Instruments (2013). *MSP430 Datasheet (SLAS590L)*. Texas Instruments Incorporated, Dallas, Estados Unidos da América.

[Texas Instruments 2014] Texas Instruments (2014). *MSP430 LaunchPad User's Guide (SLAU533B)*. Texas Instruments Incorporated, Dallas, Estados Unidos da América.

[Texas Instruments 2015] Texas Instruments (2015). *MSP430 User's Guide (SLAU208O)*. Texas Instruments Incorporated, Dallas, Estados Unidos da América.

[The Palatin Project] The Palatin Project. *Life and Work of Elisha Gray*. Disponível em: <http://www.palatin-project.com/palatin-project/elisha-gray.htm>. Acesso em: dezembro de 2016.

[Wanderley 2001] Wanderley, M. M. (2001). *Gestural Control of Music*. Kassel, Alemanha. International Workshop Human Supervision and Control in Engineering and Music.

[Website Imogen Heap 2012] Website Imogen Heap (2012?). The gloves. Disponível em: <http://www.imogenheap.co.uk/thegloves/>. Acesso em: novembro de 2016.

Apêndices

Apêndice A - A Família MSP430

5.2.1 Introdução

Os microcontroladores MSP são fabricados pela Texas Instruments na plataforma MSP430®, que é projetada para aplicações de baixo consumo de energia com sinais mistos (digitais e analógicos - daí a sigla *Mixed Signal Processor*).

São baseados em uma estrutura *RISC* (núcleo com gama restrita de instruções, do inglês *Reduced Instruction Set Computing*) de 16 bits, o que significa que realizam o controle dos dados com uma variedade limitada de instruções. Alguns periféricos externos (digitais e analógicos) podem ser utilizados junto ao controlador, como sensores, *displays* e transmissores de radiofrequência.

O diferencial desses microcontroladores é que são projetados para boa performance com consumo de energia extremamente baixo, o que é altamente necessário quando o usuário se vê obrigado a alimentar o sistema com baterias - como no caso de não haver uma tomada elétrica, transformador de tensão ou para aplicações em locais inóspitos, como florestas e desertos.

A Texas Instruments oferece a possibilidade de trabalhar com os controladores MSP430 em LaunchPads®, que podem apresentar itens como conector micro-USB, alguns cristais osciladores, LEDs, dentre outros dispositivos. Em geral, o LaunchPad se mostra como um ecossistema completo para teste e experimentação com os microcontroladores MSP430.

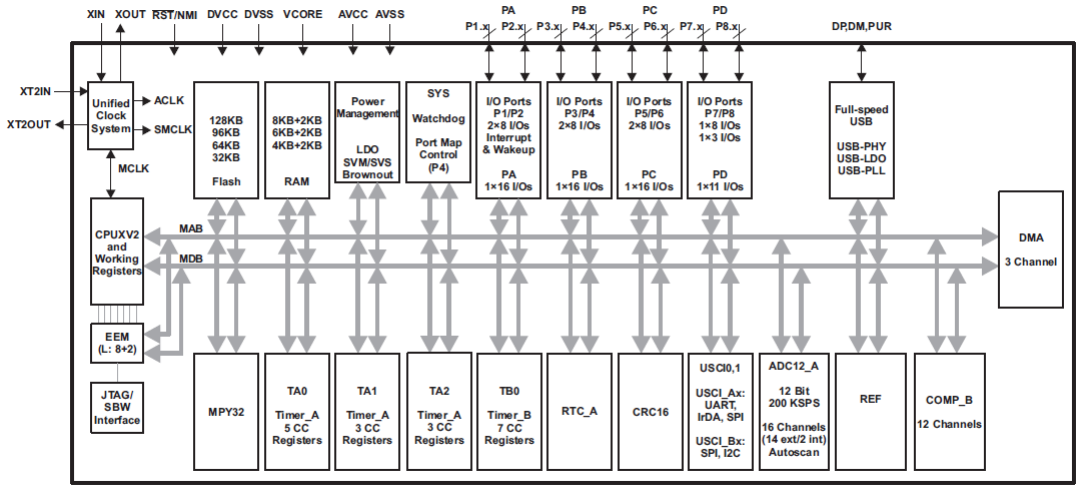
A programação do MSP430 é feita utilizando os registros dos módulos periféricos, como mencionado anteriormente. Esses registros são elementos de memória de 16 bits de profundidade de palavra (no caso da família MSP430), construídos pela associação de *flip-flops*, e que podem ser combinados para formar blocos de memória. Assim, a memória pode ser interpretada como um conjunto de localizações endereçáveis de registros [Ünsalan and Gürhan 2014]. A cada elemento de cada periférico é endereçado ao menos um campo de registro, para que se possa controlá-lo.

5.2.2 Estrutura do MSP430F5529

O modelo MSP430F5529 possui possibilidade de alimentação de 3.6V até o mínimo de 1.8V com baixo consumo, o que lhe confere a capacidade de operar durante muito tempo com a mesma bateria (obviamente esta capacidade varia em função das aplicações e dos módulos utilizados). Essa economia pode ser ainda mais evidente caso seja utilizado um modo de baixo consumo (*Low Power Mode*).

Em modo ativo, com todos os *clocks* em funcionamento, a corrente média que é requisitada da fonte (a 3V e 8 MHz) é de cerca de $290\mu A$ no caso da execução de um programa típico da memória Flash, que é uma corrente relativamente baixa. No caso de funcionamento em modo LPM3, por exemplo, essa corrente pode ser reduzida para $2,1\mu A$ em condições semelhantes (porém nem todos os *clocks* estarão ativos), com tempo rápido de resposta. Aplicações com menos requisitos de processamento podem funcionar em modo *shutdown* com consumo de cerca de $180nA$ na mesma tensão. Em suma, o controlador é realmente econômico em se tratando de energia. O diagrama de blocos com os módulos internos pode ser observado na Figura 5.1.

Figura 5.1 – Módulos internos do controlador MSP430F5529.



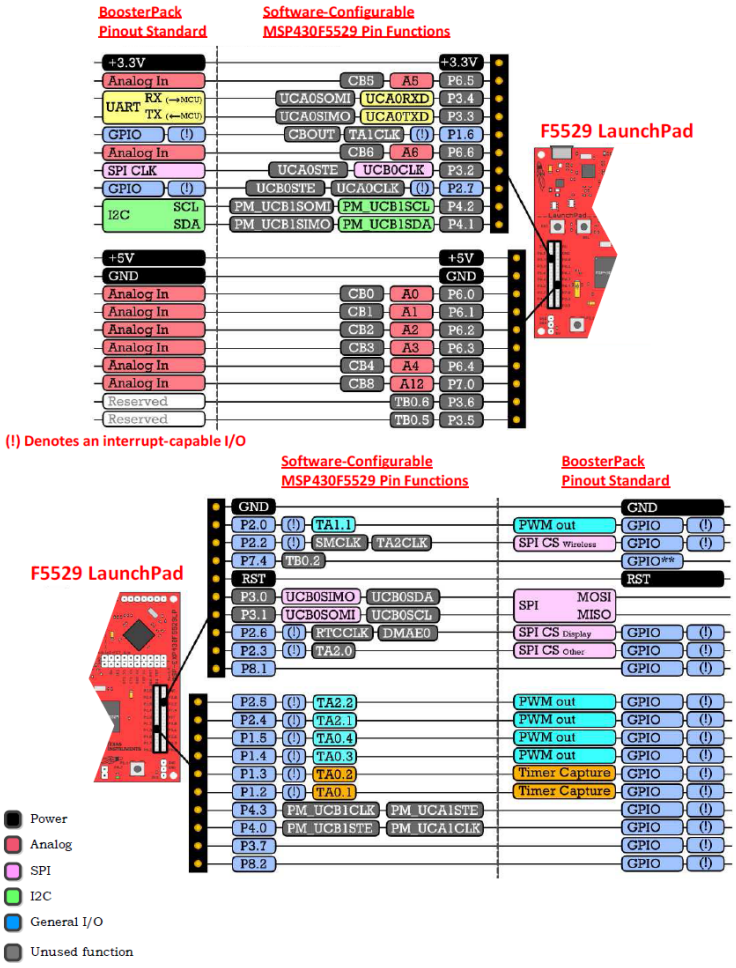
Fonte: [Texas Instruments 2013].

Como se pode ver na Figura 5.1, e lembando o mencionado na seção 2.1, diferentemente de um processador simples, o microcontrolador abarca vários módulos indispensáveis para seu funcionamento. Assim, verifica-se que internamente há um sistema de *clocks*, memórias FLASH e RAM, *timers*, portas de entrada e saída, tensões de referência, conversor analógico-digital, entre outros. O processador em si é designado como *CPU XV2 and Working Registers*, que é composto pelo núcleo de processamento e os registros internos para seu funcionamento.

O LaunchPad MSP430F5529 possibilita o acesso aos pinos de diversas portas de comunicação. A nomenclatura *PX.Y* se refere a um pino Y de uma porta X. Ao se examinar o *datasheet* do controlador percebe-se que existem 8 portas (P1, P2, P3, P4, P5, P6, P7 e P8),

cada uma com 8 pinos cada (menos a porta 8, que tem apenas 3 bits acessíveis), totalizando *a priori* 59 pinos. Alguns deles têm aplicação específica (como os pinos P1.2 e P4.7, que são ligados a LEDs e funcionam somente como saída, acendendo os diodos) ou não são normalmente acessíveis, reduzindo, assim, os pinos disponíveis. Além disso, os pinos das portas são por vezes multiplexados com outros módulos do controlador, fazendo com que esses módulos possam ser acessados externamente. A Figura 5.2 demonstra essa dualidade.

Figura 5.2 – Pinos disponíveis no LaunchPad MSP430F5529 e suas funcionalidades



Fonte: [Texas Instruments 2014].

A unidade de processamento central do MSP430 possui 16 registradores próprios, sendo que os 4 primeiros (R0, R1, R2 e R3) são de uso dedicado do processador e os outros 12 são de uso genérico. A característica especial desses 16 registros é que o processador os acessa muito rapidamente, com apenas um ciclo de *clock* (diferentemente da memória RAM, que leva mais ciclos para ser acessada). Os três primeiros têm funções importantes e devem ser analisados [Texas Instruments 2009]:

1. R0: PC (PROGRAM COUNTER). Esse registro de 20 bits é responsável por armazenar o valor da próxima instrução a ser executada pelo processador. O tamanho das instruções varia entre 2, 4, 6 e 8 bytes, e o PC é incrementado conforme a instrução é

lida. Todas as instruções tem um endereço par, portanto o último bit desse registrador é ligado ao nível lógico 0 por hardware.

2. R1: SP (STACK POINTER). Quando uma sub-rotina é chamada (como por exemplo, uma interrupção), novas instruções são dadas ao processador e, quando da sua finalização, o processamento deve voltar ao ponto onde parou antes dessa quebra. Há um problema nessa lógica: o valor do PC foi alterado para a execução da sub-rotina e, agora, o processador não tem em vista a próxima instrução. É nesse ponto que atua o *stack pointer*, um registrador de 20 bits que funciona como uma pilha guardando o valor da última instrução do PC antes da execução da sub-rotina. [Davies 2008] faz uma analogia com um mecanismo de servir pratos em uma cafeteria: há um tubo com uma mola ao fundo, sobre a qual pratos são empilhados. O último prato a ser colocado na máquina será obrigatoriamente o primeiro a ser utilizado por um cliente, já que há apenas uma abertura no tubo com a mola. A esse tipo de mecanismo dá-se o nome de LIFO - *last in, first out* (o último que entra é o primeiro que sai) ou de *stack* (pilha). Assim, ao se terminarem as instruções da sub-rotina, o próximo item da pilha é justamente a instrução que poderia estar perdida no problema relatado anteriormente.

3. R2: SR (STATUS REGISTER). Esse registro de 16 bits (sendo 7 deles sem uso) contém uma série de *flags* que controlam o funcionamento do processador (seu estado). Assim, é nesse registrador que ficam armazenadas informações importantes, que podem ser divididas em 3 categorias [Davies 2008]:

   (a) Resultados de operações aritméticas e lógicas: 4 bits são responsáveis por armazenar se a última operação resultou em *carry*, *signed overflow*, negativo ou zero;

   (b) Controle de *Low Power Modes*: 4 bits são responsáveis por armazenar as configurações do modo de operação (ativo ou algum dos LPMs);

   (c) Habilitação de interrupções globais: um bit é dedicado a apontar para o processador se as interrupções vindas dos módulos devem ser mascaradas ou não (ou seja, se devem ou não levantar *flags* e serem resolvidas via ISR).

Apêndice B - Os Módulos Utilizados

As subseções a seguir descrevem alguns dos módulos internos e funções que foram utilizados para o projeto. Para a correta interpretação do que será exposto, deve-se atentar a alguns detalhes do modo como essa seção foi escrita:

- Cada registro tem por definição 16 bits em geral (contados de 0 a 15), sendo que alguns desses bits podem ser reservados (não podem ser utilizados);
- Um ou mais bits (dos 16) formam um *campo* dentro do registro, a depender de sua funcionalidade, que configura um de seus componentes;
- Quando não mencionado, o campo em questão tem valor de início (*default*) igual a 0;
- O verbo *setar* (do inglês *set*) é utilizado para se referir ao processo de colocar um bit em nível lógico alto (1), e o verbo resetar (do inglês *reset*), em nível lógico baixo (0);
- O símbolo b significa que o número que o antecede está na base binária; h significa que o número que o antecede está na base hexadecimal; quando não houver nenhum desses símbolos, o número deve ser lido por definição na base decimal;
- o método utilizado em linguagem C para *set* e *reset* dos pinos está detalhado no Capítulo 4.

5.2.3 *Unified Clock System*

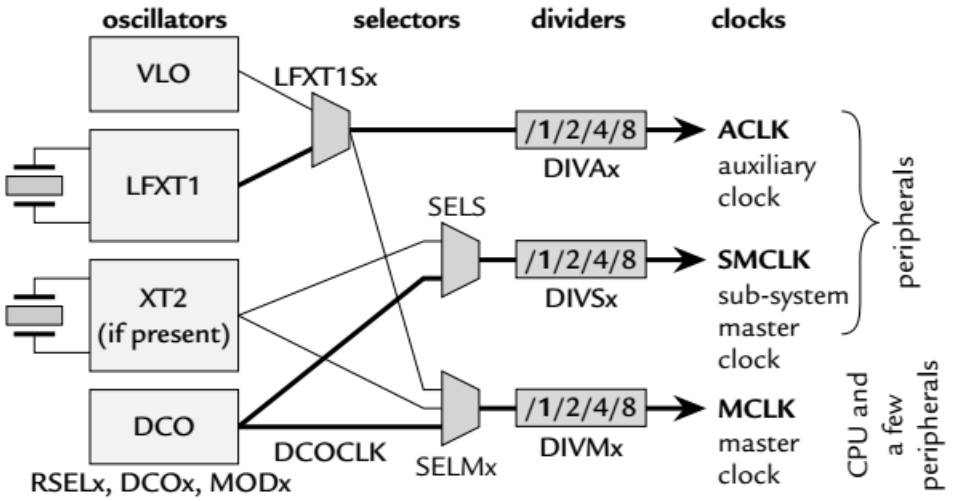
O *Unified Clock System* é um dos módulos periféricos do MSP430F5529LP e possibilita o controle total via *software* dos seus três sinais de saída de *clock*. Os *clocks* são essenciais para qualquer sistema digital e por isso sua qualidade e flexibilidade de controle são de extrema importância. O UCS permite o melhor balanço de performance e baixo consumo de energia. Pode funcionar de maneira totalmente independente, sem nenhum componente externo, ou ainda com o uso de cristais externos.

O UCS possui até 5 fontes internas diferentes de *clock*: XT1CLK, XT2XLK, VLOCLK, REFOCLK e DCOCLK. De acordo com suas características, uns são mais adequados para algumas aplicações do que outros e algumas funções específicas do *LaunchPad* só se conectam a certa fonte de *clock* por definição de fábrica. Os osciladores XT1CLK e XT2CLK são parecidos e funcionam com cristais. No modelo de *LaunchPad* usado, esses cristais já vêm fixados na placa, sendo um de alta frequência em 4 MHz para ambos osciladores e um em baixa frequência apenas para o XT1 em 32768 Hz. A fonte do VLOCLK é totalmente interna, de baixíssimo consumo de energia e baixa frequência. Sua implementação é puramente digital e tem variação considerável de ± 4 kHz na frequência de base de 10 kHz, não sendo, portanto, indicado para aplicações que necessitem de precisão. O REFOCLK também é interno, possui frequência típica de 32768 Hz e assim como o VLOCLK não é uma

fonte de *clock* muito precisa. Por último, o DCOCLK é um *clock* controlado digitalmente que pode ser estabilizado pelo FLL, que será abordado mais a frente.

Como pode ser observado no diagrama de blocos simplificado do UCS, na Figura 5.3, essas fontes de *clock* citadas são selecionadas para saírem do módulo em três canais diferentes: ACLK, MCLK e SMCLK. Existem opções de divisores de frequência para que se chegue ao valor mais próximo possível do desejado. Tais divisões são controladas no registro UCSCTL5, que além de configurar os divisores das fontes das três saídas ajusta também o divisor da fonte do ACLK.

Figura 5.3 – Diagrama de blocos do UCS



Fonte: [Davies 2008].

Dentro dessa vasta gama de opções e variedades de *clock*, serão mais bem detalhado os que de alguma forma fazem parte do escopo deste trabalho, assim como os registros responsáveis por suas respectivas configurações. A escolha dos *clocks* foi feita por conveniência, facilidade de uso e necessidade exigida pela função que os utiliza. Não obstante, é importante salientar que isso não significa que são as únicas opções disponíveis no LaunchPad.

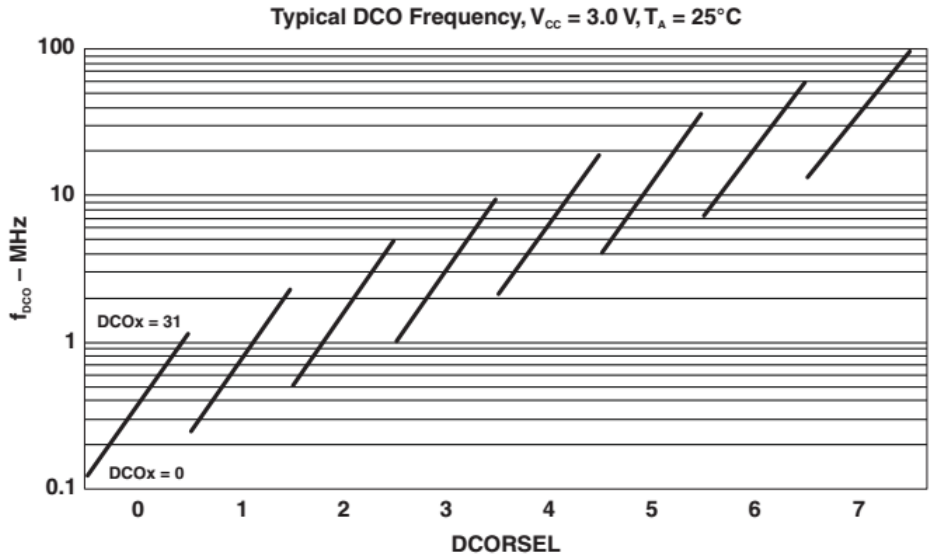
5.2.3.1 *Digitally-controlled oscillator (DCO)*

O DCO é controlado digitalmente a partir de uma frequência de referência. É o *clock* dedicado principalmente para alimentar o processador mas pode ser referência para alguns periféricos também. Como é o *clock* que segue para o processador, é necessário que cumpra alguns requisitos, por isso é capaz de retornar rapidamente de um *low power mode* (LPM) na velocidade correta, sem que demore a se estabilizar. Esse tempo de retorno gira em torno de 1 a 2 μs, segundo [Davies 2008].É importante também que tenha estabilidade, e o desvio (da precisão) gira em torno de 1% a 2%, segundo o mesmo autor.

A frequência e todo o funcionamento do DCOCLK é ajustado por *software* nos quatro primeiros registros do UCS, do UCSCTL0 ao UCSCTL3. As faixas de frequências que ele

pode atuar é escolhida de acordo com o gráfico da Figura 5.4 nos bits DCORSEL dentro do registro UCSCTL1.

Figura 5.4 – Faixas de frequências escolhidas pelos bits DCORSEL



Fonte: [Texas Instruments 2013].

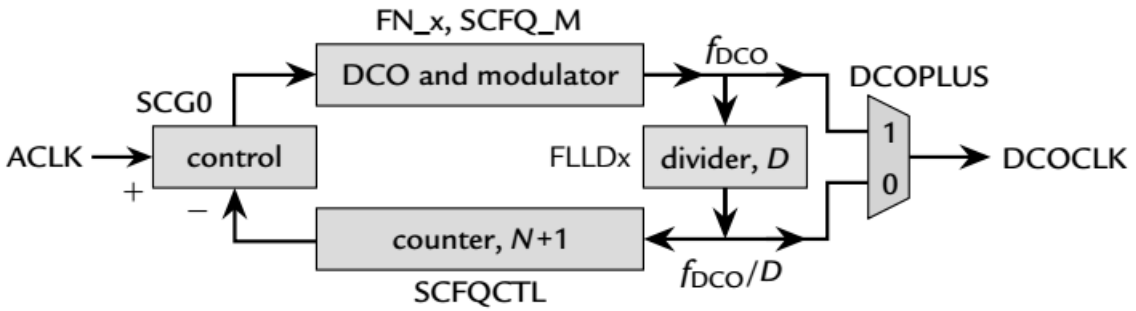
No primeiro registro, o UCSCTL0, é possível escolher valores para os bits DCO e MOD. Ambos são também responsáveis por ajustar a frequência do DCOCLK. Os bits DCO dividem a faixa de frequência escolhida em 32 frações menores, ou *taps* de frequência, como são chamadas. Selecionando-se esses bits, escolhe-se em qual fração o oscilador irá trabalhar e é possível chegar mais perto ainda da frequência desejada. No campo MOD, é configurado o tipo da modulação a ser usada. Essa modulação funciona de forma a criar uma frequência ligeiramente maior em um *tap* do DCO, ou seja (DCO + 1). Assim os bits MOD determinam quantos ciclos da frequência maior criada (DCO + 1) serão inseridos no DCOCLK a cada 32 ciclos. Exemplificando, se o MOD é configurado como igual a 8, tem-se 8 ciclos na frequência aumentada, além dos 24 ciclos restantes na frequência anterior ajustada nos bits DCO. Essa modulação é muito útil se o objetivo é obter uma frequência média bem precisa, que pode ser calculada pela expressão 5.1 (sendo a frequência o inverso do período [T]). Porém, se o caso necessita mais de um período constante do que de uma média precisa, a modulação não é útil e deve ser desligada no registro seguinte (UCSCTL1), no campo DISMOD. É importante lembrar que com a modulação desligada, a frequência pode oscilar mais com variações de temperatura e tensão.

$$T_{medio} = \frac{MOD \cdot T_{DCO+1} + (32 - MOD) \cdot T_{DCO}}{32} \tag{5.1}$$

O DCOCLK possui também uma ferramenta bastante útil chamada FLL (*Frequency Locked Loop*). O FLL é um *loop* da saída do DCOCLK passando em dois divisores de frequência e retornando a um comparador, que vai elevar a frequência da saída do DCOCLK até que sua realimentação, que é o sinal original dividido por 2 fatores diferentes, seja de igual valor

à referência adotada. Ou seja, se uma referência de 32768 kHz for adotada e os divisores forem ajustados em 2 e 2, temos a frequência de retorno igual a 8192 kHz. Para igualar o valor dessa entrada de retorno com a referência adotada, a saída do DCOCLK é multiplicada por 4, para que quando seja divida por 2 e 2 novamente no *loop*, a frequência de retorno seja igual à referência. Em outras palavras, os divisores funcionam na verdade como multiplicadores da frequência, e o FLL é uma ótima ferramenta para estabilizar o *clock*. Seu esquema de funcionamento pode ser visualizado na Figura 5.5.

Figura 5.5 – Diagrama de blocos do FLL



Fonte: [Davies 2008].

A fonte do *clock* de referência do FLL é selecionada no registro UCSCTL3 no campo SELREF, e um divisor dessa referência é ajustado no mesmo registro, no campo FLLREFDIV. No registro anterior (UCSCTL2) são ajustados os divisores dentro do FLL, que são o FLLD e o FLLN, sendo que este pode receber qualquer valor numério não nulo. Um ajuste com valor 0 resulta em 0+1, ou seja, 1, e assim por diante com qualquer valor dentro do seu limite de bits. Assim, temos o valor final da frequência do DCOCLK com FLL calculado pela expressão 5.2:

$$DCOCLK = FLLD \cdot (FLLN) \cdot \frac{FLLREFCLK}{FLLREFDIV}, \tag{5.2}$$

5.2.3.2 *Internal Trimmed Low-Frequency Reference Oscillator (REFO)*

O REFO é uma referência estável fixa em 32768 Hz e pode ser usado como FLLREFFCLK. Pode também ser utilizado como uma referência direta para uma saída de *clock* como, por exemplo, o ACLK, configurando os bits SELA_2 do registro UCSCTL4. Tal registro é responsável por conectar as fontes de *clock* às três saídas do UCS: ACLK, SMCLK e MCLK.

5.2.3.3 *XT1 Oscillator*

O oscilador XT1 permite funcionamento tanto em baixa como alta frequência. É bem flexível em suas configurações sendo possível ao usuário controlar via software uma capacitância de carga interna, o *drive strength* do sinal, além de poder operar com cristais externos.

O seu controle é feito no registro UCSCTL6, que também controla parâmetros do oscilador XT2 e permite desligar, além dos cristais, a saída SMCLK.

5.2.4 *Timers*

O *timer* é um temporizador digital cuja função é contar ciclos do *clock*. Várias são suas aplicações, sendo muito utilizados para sinalizar períodos pré-definidos de tempo (funcionando como um *output*, por exemplo) ou para sinalizar a duração de sinais (funcionando como um *input*, por exemplo). Existem 4 módulos módulos temporizadores presentes no MSP430F5529: 3 são referentes a diferentes instâncias do *Timer A* e 1 é referente ao *Timer B*. Suas configurações são levemente diferentes, e o *Timer B* não foi utilizado. Os 3 módulos do *Timer A* são destinados, respectivamente, ao *Timer A0*, *Timer A1* e *Timer A2*, que têm configurações semelhantes. Esses são formados por 2 blocos principais: o contador em si e os registros de comparação e captura (CCRn). No *TA0* existem 5 desses registros CCR (variando entre CCR1 até CCR5) para o mesmo contador, enquanto no *TA1* e no *TA2* existem 3 desses registros respectivos para cada. O diagrama de blocos do *Timer A* é mostrado na Figura 5.6.

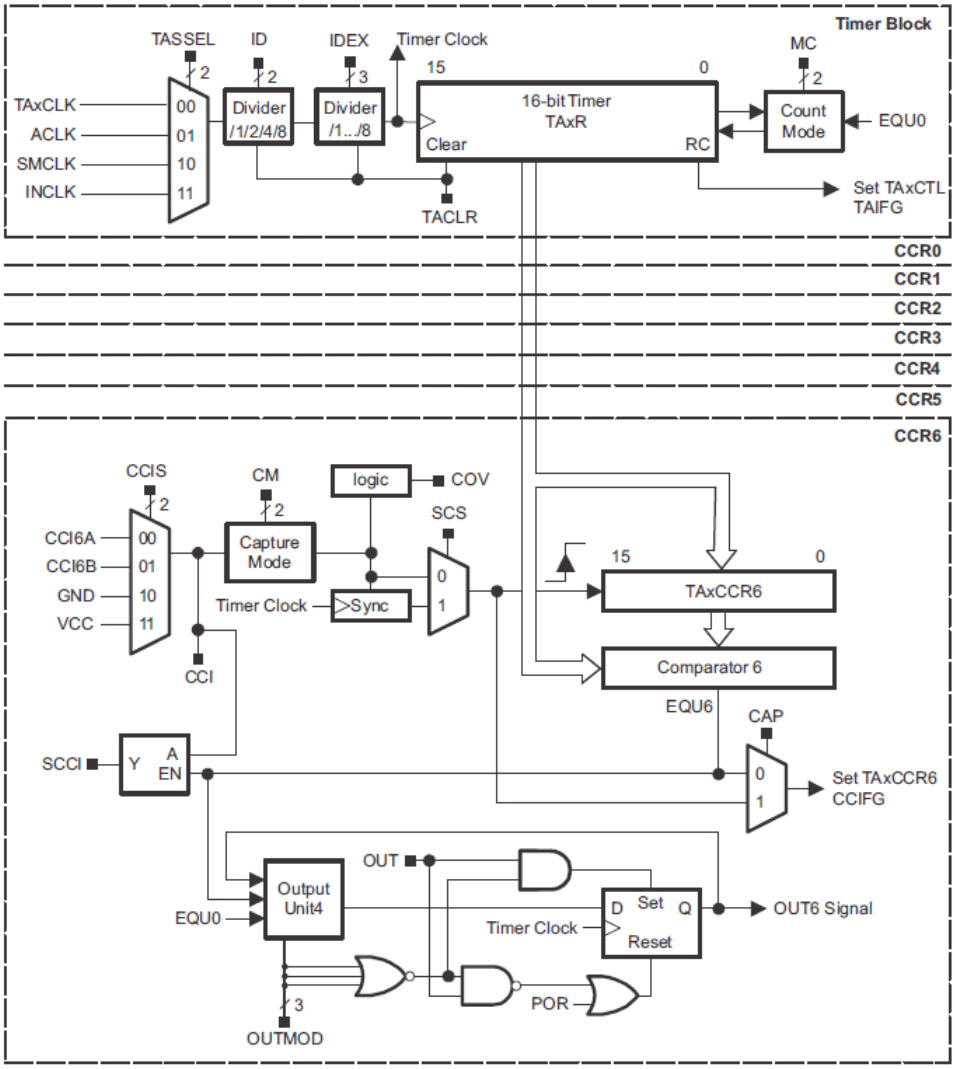
Os CCR são utilizados para sinalizar quando o valor do *timer* atinge certo valor (comparação) ou quanto tempo dura certo sinal (captura). Nesse último caso, pode-se exemplificar com o retorno do sinal do sensor ultrassônico: quanto maior é a largura do pulso recebido pelo controlador, maior é a distância que foi medida. Assim, o temporizador pode capturar os momentos em que o sinal começa e termina, retornando assim a quantidade de ciclos de c*clock* que esse sinal durou. Então, com breves cálculos aritméticos (proporções) envolvendo esse número de ciclos pode-se saber a distância exata constatada pelo sensor.

A contagem pode ser configurada para ser realizada de 3 modos diferentes. O primeiro modo é o de subida (*up mode*): o temporizador conta de 0 até o valor definido no registro TAxCCR0 em *loop* (CCR0 está sendo usado para comparação), ou seja, ao atingir o valor CCR0 o contador volta ao zero e o processo se dá novamente. Existe também o modo contínuo (*continuous mode*), cuja única diferença para o modo de subida é o valor final (comparado): nesse caso não é o valor de CCR0, e sim 0FFFFh, o valor máximo que o contador pode assumir. Por último, existe o modo de subida/descida (*up/down mode*): o temporizador conta de 0 até o valor definido TAxCCR0 e, então, de CCR0 até 0, decrescendo (no caso o contador fica em CCR0 por um único ciclo, então o correto é que o ciclo de descida se dá de CCR0-1 até 0). O ciclo de subida-descida também é feito em laço.

Existem basicamente duas *flags* de interrupção do *Timer A*:

TAIFG: Esta *flag* é acionada quando o contador chega no último valor antes do 0. Ou seja, significa que completou um ciclo. No modo de subida, ela é ativada entre CCR0 e 0; no modo contínuo, entre 0FFFFh e 0; no modo de descida/subida, é ativada entre 1 e 0, na parte de descida (já que a contagem é decrescente nessa parte, após o 1 vem o 0).

Figura 5.6 – Diagrama de blocos do módulo *Timer A*. A quantidade de registros CCR pode variar conforme a instância.



Fonte: [Texas Instruments 2015].

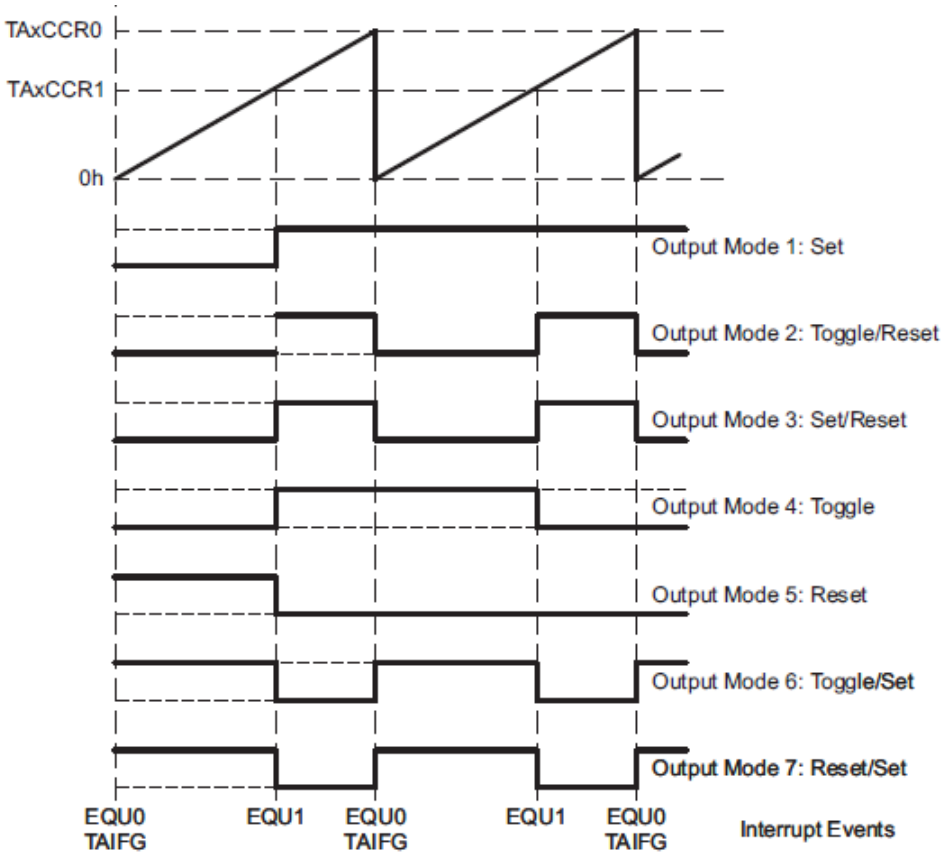
Essa interrupção tem uma única origem.

CCIFG: Esta *flag* indica que algum registro CCR atingiu se valor de comparação ou de captura, conforme esteja configurado. Portanto, nos modos de subida e de subida/descida sempre haverá uma interrupção ao se chegar valor máximo, ou seja, logo após CCR0 ser atingido. Como qualquer bloco CCR pode gerar essa interrupção, existe um vetor responsável por indicar qual bloco a gerou.

Um sinal de saída automático do *timer*, dependente das *flags*, pode ser configurado para aplicações desejadas. Esse sinal, OUTn, pode ser utilizado, por exemplo, para o controle de ondas de largura de pulso modulada (ou PWM, do inglês *Pulse Width Modulation*). Os tipos de sinal de saída disponíveis estão esquematizados na Figura 5.7.

Assim sendo, para o funcionamento correto do *timer* é necessário que se configure o módulo UCS e os registros CCR, além dos registros de controle geral e as interrupções. Os

Figura 5.7 – Possíveis configurações do sinal de saída OUTn do *Timer A* em contagem do tipo *up mode*.



Fonte: [Texas Instruments 2015].

registros mais importantes para o projeto são relatados a seguir.

5.2.4.1 *TAxCTL*

Esse registro controla cada *Timer Ax* (por exemplo, TA**0**CTL controla o *Timer A**0***). Os diferentes campos selecionam qual *clock* será utilizado como base de contagem (e uma possível divisão de sua frequência), qual será o modo de contagem e a ativação das *flags* de interrupção.

5.2.4.2 *TAxR*

Esse registro representa o valor atual do *Timer Ax*. Os bits de 0 a 15 guardam o valor binário (o bit mais significativo é o 15) atual do contador.

5.2.4.3 *TAxCCTLn*

Esse registro controla cada bloco n de comparação e captura (CCR) do *Timer Ax*. Por exemplo, o registro TA1CCTL0 controla o bloco CCR0 do *Timer A1*. Os campos selecionam a função de comparação ou a de captura e suas respectivas configurações, como a fonte de uma captura e se ela ocorrerá em um flanco de subida ou de descida. É por meio desse registro que é controlado também o sinal de saída OUTn do bloco CCR

5.2.4.4 *TAxCCRn*

Esse registro é a memória de cada bloco n de comparação e captura (CCR) do *Timer Ax*. Por exemplo, o registro TA1CCR0 armazena o valor de CCR0 do *Timer A1*. Em modo de captura, o valor de TAxR é copiado para esse registrador quando uma captura for concluída; em modo de comparação, o valor desse vetor é comparado ao valor de TAxR.

5.2.4.5 *TAxIV*

Esse registro é associado às *flags* de interrupção e é responsável por armazenar sua fonte. Assim, o sistema pode identificar a fonte de certa interrupção, permitindo que o programador configure uma rotina para seu tratamento.

5.2.5 GPIO

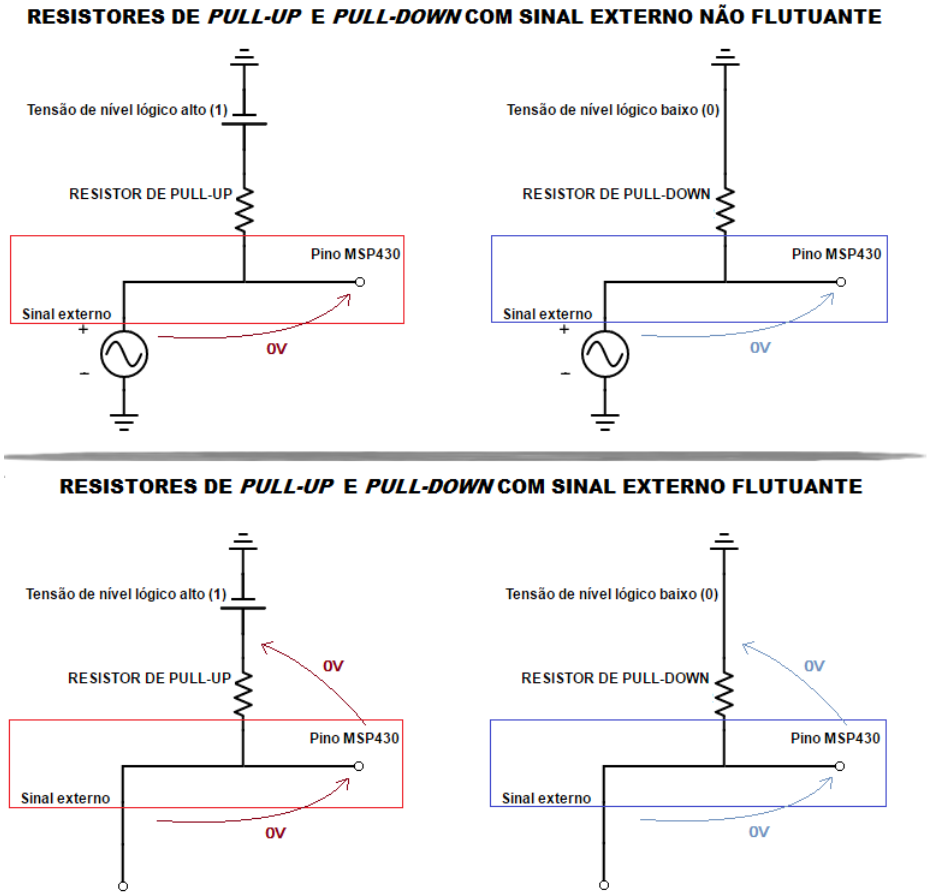
As portas do controlador são seu meio de comunicação com o mundo externo e com seus próprios periféricos. Podem ser configuradas para enviar ou receber sinais digitais (modo de *input*/*output*) ou para atuarem como acessos aos módulos internos do controlador, como por exemplo a saída OUTn do *timer*. Como cada porta possui no máximo 8 pinos, seus registradores têm só 8 bits, e estão descritos mais abaixo. GPIO é o acrônimo para *General Port Input/Output* (do ingês, portas gerais de entrada/saída).

No caso do MSP430F5529 as portas P1 e P2 podem ser programadas para gerarem interrupções no sistema. Isso pode acontecer numa transição de nível lógico baixo para nível lógico alto (bordo de subida) ou ao contrário, de nível lógico alto para nível lógico baixo (bordo de descida), e os registradores permite uma configuração individual para cada pino dessas duas portas. Uma simples aplicação dessa funcionalidade pode ser exemplificada da seguinte forma: um protocolo de comunicação I2C, que por *default* segura o sinal em 1, em determinado momento, para sinalizar começar a enviar os dados, muda esse sinal para 0V para sinalizar que deseja começar a enviar um pacote de dados (a operação completa é um pouco mais complexa que isso, já que envolve 2 sinais distintos, sendo um do *clock*, chamado SCL, e outro de dados, chamado SDA). Assim, deseja-se identificar quando há um bordo de descida na porta que está recebendo este sinal, para que os dados possam ser processados:

isso é feito com uma interrupção.

Além disso, as portas possuem outras funcionalidades interessantes, como resistores de *pull up*/*pull down*. Quando configurados como *input*, o pino fica à mercê do sinal a ele enviado. Portanto, se o sinal vale 0, o pino será lido como 0 e, se o sinal vale 1, o pino será lido como 1. Porém, caso o sinal esteja flutuando (ou seja, alta impedância: não está conectado em nenhuma tensão), a leitura poderia ser comprometida caso não fosse usado esse método. Os resistores mantém um nível lógico definido nesses casos, o que não permite que leituras erradas ocorram em inesperados de flutuação. Como pode ser visto na Figura 5.8, a entrada do pino está no mesmo nó que o sinal enviado (os nós estão demarcados nos retângulos coloridos), portanto assume a mesma tensão, independentemente da tensão acoplada ao resistor. Porém, caso o sinal flutue, a tensão de alimentação do resistor será repassada ao pino, que também está flutuando (ou seja, nenhuma corrente deve fluir, a menos das perdas). Portanto, o sistema não terá a leitura da porta comprometida.

Figura 5.8 – Esquema dos resistores de *pull-up* e *pull-down*.



Fonte: Elaborado pelo autor.

5.2.5.1 PxIE

Esse registro existe somente para as duas primeiras portas (ou seja, P1IE e P2IE). Ele habilita as interrupções dessas portas para cada pino. Portanto, caso se deseje habilitar o funcionamento das interrupções do pino **2.0**, seta-se o primeiro bit (bit **0** dos 7 da porta) do registrador P**2**IE.

5.2.5.2 PxIES

Esse registro existe somente para as duas primeiras portas (ou seja, P1IES e P2IE). Ele seleciona o que vai acionar a interrupção (funciona em conjunto com PxIE), ou seja, se a *flag* será gerada em um bordo de subida (mantém-se em 0 o valor desse registro para esse caso) ou em um bordo de descida (seta-se esse registrador para esse caso). Portanto, caso se deseje que o pino **1.7** gere uma interrupção se houver nele um bordo de descida, é necessário ativar o bit **7** do registrador P**1**IES.

5.2.5.3 PxIV

Esse registro é somente de leitura e existe somente para as duas primeiras portas (ou seja, P1IV e P2IV). Ele é o vetor de interrupção: armazena a origem da *flag* e possibilita ao usuário saber de onde partiu a interrupção. Com ele é possível, então tratá-la.

5.2.5.4 PxIFG

Esse registro é responsável por armazenar as *flags* de interrupção (é utilizado em conjunto com os registros anteriores). Ou seja, ao haver uma interrupção na porta x (x podendo ser 1 ou 2), o *software* irá procurar uma ISR. Com auxílio do registrador PxIV pode-se saber a origem e, após tratamento, zerar essa flag no registrador PxIV. Assim, caso a porta **2.3** gere uma interrupção, o bit **3** do registro P**2**IFG estará ativado enquanto essa interrupção não for tratada.

5.2.5.5 PxSEL

Seleciona o modo de operação do bit, sendo 0 para *input*/*output*, por padrão, e 1 para utilização como função de algum módulo periférico interno. Se o bit **6** do registrador P**5**SEL estiver ativado, por exemplo, o pino **P5.6** terá seu funcionamento dedicado ao *Timer B0*. Ao final do *datasheet* do controlador, no capítulo de *Input/Output Schematics*, estão detalhados todas essas configurações de funcionamento dos pinos para os módulos periféricos [Texas Instruments 2013].

5.2.5.6 PxDIR

Caso o pino esteja configurado como *I/O* (portanto PxSEL está em 0 para esse pino), pode-se selecionar se ele vai operar como entrada (*Input*: PxDIR deve valer 0 para esse pino) ou como saída (*Output*: PxDIR deve valer 1 para esse pino).

5.2.5.7 PxREN

Caso o pino esteja configurado como **entrada** (PxSEL e PxDIR estão em 0 para esse pino), pode-se ativar ou desativar os resistores de *pull up*/*pull down*. O funcionamento e conexão dos resistores é ilustrado na Figura 5.8.

5.2.5.8 PxIN

É um registrador somente de leitura. Possibilita saber o nível lógico em que se encontra o um pino da porta x.

5.2.5.9 PxOUT

Um dos registradores de GPIO mais importantes: ativa a saída dos pinos. Caso o pino esteja configurado como saída, colocar o bit referente a ele nesta porta em nível alto fará com que o pino assuma o valor 1 (e 0 para o caso de nível lógico baixo, respectivamente). Ou seja, no LaunchPad do MSP430F5529LP, sendo que o pino P1.2 é ligado diretamente a um LED de fábrica, ativar o bit 2 do registrador P1OUT fará com que o LED seja aceso.

Caso o pino esteja configurado como *input*, esse registro passa funcionar como extensão do registrador PxREN. Assim, caso ps resistores de *pull-up*/*pull-down* estejam ativados no pino (PxREN=1), PxOUT irá selecionar qual deles será utilizado. Se for setado, o *pull-up* é selecionado; se for resetado, o *pull-down* é selecionado.

5.2.5.10 PxDS

Esse registrador é pouco utilizado, mas não menos importante. DS vem do inglês *Drive Strength*, ou seja, caso um pino tenha de ser acoplado a sistemas com alta impedância, talvez seja necessário reforçar a corrente que ele oferece. Isso é feito setando o seu bit do registro PxDS.

5.2.6 ADC_12A

O módulo ADC_12A, cujo diagrama de blocos pode ser visualizado na Figura 5.9, realiza a conversão de sinais analógicos em sinais digitais (em inglês, *Analog-Digital Conversion*)

e seu núcleo é de tecnologia SAR (*Successive Approximation Register*, sigla em inglês para Registrador de Aproximações Sucessivas), o que o torna muito rápido e uniforme. Esse tipo de conversão é feito da seguinte forma:

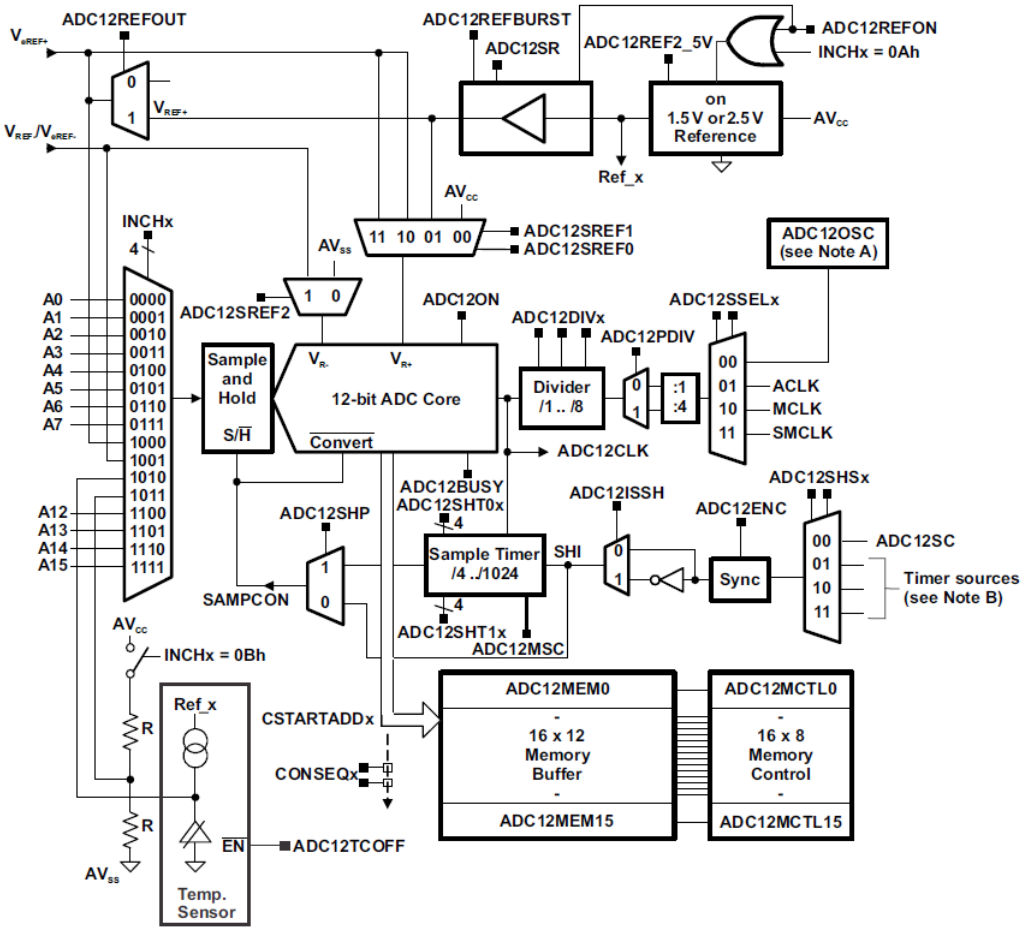
1. Efetua-se o processo de amostragem (*sample-and-hold*) do sinal de entrada;
2. O conversor seta o bit 11 do sinal digital (coloca o bit mais significativo, MSB, em nível alto), enquanto os bits seguintes são mantidos em nível lógico baixo;
3. É realizada uma comparação entre o sinal de entrada e o sinal de saída modificado no passo anterior: se o sinal de saída, com o MSB ativo, apresentar valor (amplitude de pico) maior do que o sinal de entrada, o bit é então resetado. Caso contrário, o bit é mantido ativo e passa-se para o próximo estágio;
4. O conversor seta, então, o segundo bit mais significativo (bit 10), fazendo com que a tensão de referência seja metade da tensão anterior;
5. A comparação descrita no passo 3 é repetida, agora para o sinal com o bit 10 ativado (o bit 11 permanece igual ao final do processo anterior);
6. De modo similar, o mesmo processo é feito para os bits restantes (9, 8, 7, 6, 5, 4, 3, 2, 1, 0), individualmente, até que ao final tenha-se uma boa aproximação digital do sinal, limitado superiormente pelo valor de entrada. Essa sucessão pode ser visualizada na Figura 5.10 em um sistema simplificado de 4 bits.

O ADC pode trabalhar com cerca de 200ksps (*kilo samples per second* - mil amostras por segundo), a depender do *clock* utilizado, possibilitando ao programador também controlar o período de amostragem e a tensão de referência do sinal de saída. Existem 12 canais de entrada externa para o módulo, ou seja, até 12 sinais podem ser digitalizados simultaneamente, com a possibilidade de utilizar interrupções de cada um desses canais. Uma outra especificação interessante é que há, internamente, um *buffer* de 16 palavras que guarda o resultado das conversões sem necessidade de utilizar o núcleo de processamento central ou outras memórias. A Figura 5.9 representa a estrutura do módulo ADC_12A.

Todo o processo DE amostragem e conversão é controlado pelo sinal SHI e pode funcionar de duas formas:

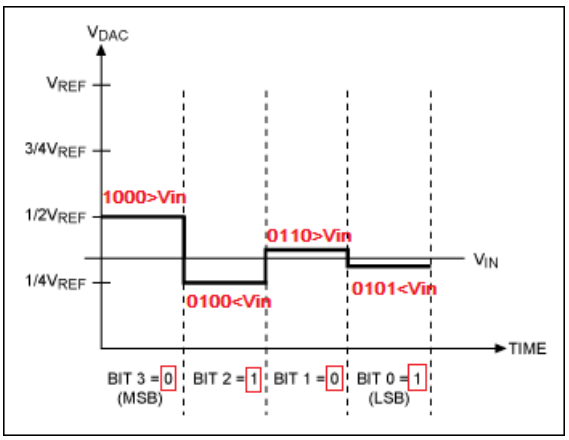
1. *Extended Sample Mode*: é selecionado quando o bit ADCSHP tem valor zero. Para esse caso, o sinal SHI controla diretamente o sinal SAMPCON e assim define o tempo de amostragem t_{sample}. A conversão do sinal amostrado começa depois do flanco de descida do SAMPCON, após um intervalo necessário à sincronização com o ADC12CLK (t_{sync});
2. *Pulse Sample Mode*: não utiliza o sinal SHI para controlar a duração do sinal SAMPCON e o tempo t_{sample} de amostragem. Neste caso, o SHI é apenas um sinal de

Figura 5.9 – Diagrama de blocos estrutural do conversor ADC_12A do MSP430F5529.



Fonte: [Texas Instruments 2015].

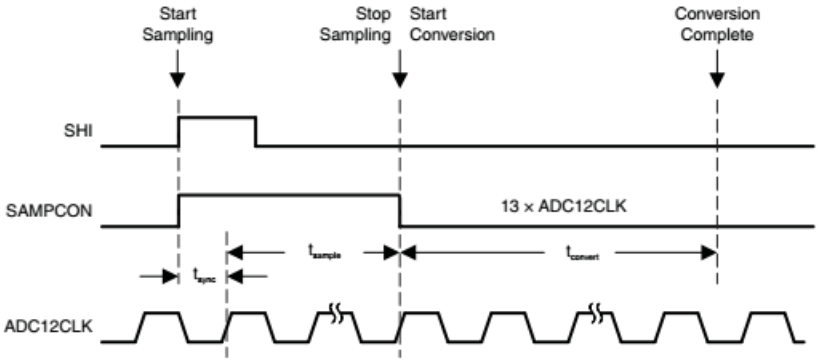
Figura 5.10 – Conversão ADC pelo método de aproximações sucessivas



Fonte: Adaptado de [Maxim Integrated Products, Inc.].

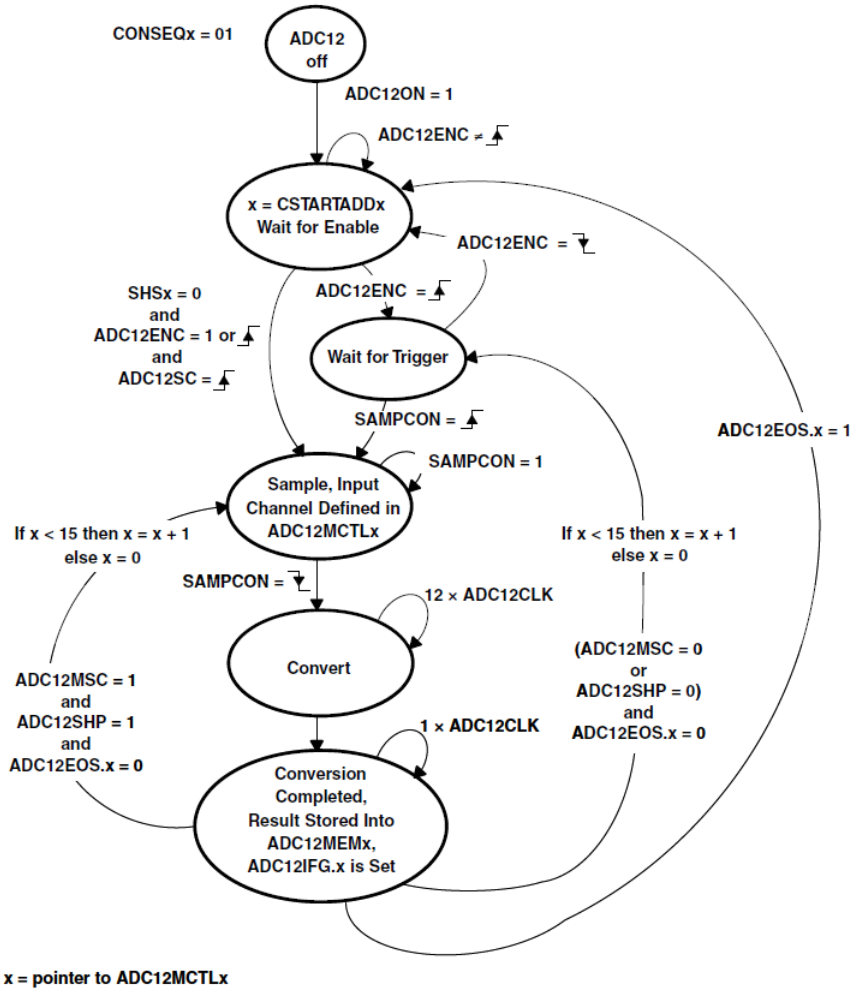
trigger para o *sampling timer*, que controla diretamente aduração do SAMPCON e assim o t_{sample}, definindo um número de ciclos determinado nos bits ADC12SHT0x e ADC12SHT1x, como pode ser visto na Figura 5.11.

Figura 5.11 – *Pulse Sample Mode*



Fonte: [Texas Instruments 2015].

Figura 5.12 – *Sequence-Of-Channels-Mode*



Fonte: [Texas Instruments 2015].

Existem quatro modos selecionáveis padronizados de conversão (podem ser selecionados por meio do campo ADC12CONSEQx), que se diferenciam no modo como os canais de entrada são tratados. São eles:

1. *Single-Channel Single-Conversion* ;

2. *Sequence-of-Channels (Autoscan Mode)*;

3. *Repeat-Single-Channel*;

4. *Repeat-Sequence-of-Channels (Repeated Autoscan Mode)*.

O modo *Sequence-of-Channels* indica a amostragem e conversão sequencial dos canais do ADC12 sem repetição (ou seja, não há um laço: há uma única conversão dos canais). Os resultados do ADC são armazenados nos registros de memória ADC12MEMx começando pelo mesmo definido no bit CSTARTADDx. A sequência é interrompida no canal onde o bit ADC12EOS (*End Of Sequence*) é ativado e a sequência só recomeça no próximo sinal de *trigger* do SHI que é disparado pelo bit ADC12SC. O ciclo de funcionamento desse modo pode ser visualizado na Figura 5.12.

O ADC pode operar com 8 bits, 10 bits ou 12 bits de saída, sendo que profundidades de palavra maiores demandam maiores tempos de conversão. Portanto, o modo de 8 bits gasta apenas 9 ciclos de processamento (mais especificamente, ciclos do ADC12CLK), o modo de 10 bits gasta 11 ciclos, e o modo de 12 bits, 13 ciclos - é o viés de se ter uma maior qualidade do sinal digital. Esse custo deve ser levado em conta ao se processar sinais com frequência muito elevada, já que pode ser que o ADC não seja rápido o suficiente com profundidades de palavra altas para não haver erro de processamento.

5.2.7 REF

Este módulo é responsável por fornecer a tensão de referência necessária para o funcionamento de módulos analógicos. Neste projeto o REF é necessário e configurado para o uso do ADC, já que o modelo utilizado não possui referência interna no próprio módulo de conversão analógico-digital.

5.2.8 32-Bit *Hardware Multiplier* (MPY32)

Este módulo é responsável por realizar multiplicações em *hardware* dedicado. Como o processador possui arquitetura RISC, uma multiplicação seria realizada em somas sucessivas, ocupando muitos ciclos de processamento. Assim, o MPY32 desocupa o processador dessa função para que tal operação aconteça de forma mais rápida e eficiente sem prejudicar aplicações que demandem rapidez e sincronia. Em seu funcionamento básico apenas recebe os números a serem multiplicados nos registros MPY e OP2 enquanto a resposta pode ser lida no registro RESLO. Tais registros recebem números de até 16 bits para serem multiplicados.

5.2.9 Modos de baixo consumo

O MSP430 é um microcontrolador projetado para consumo reduzido de energia. Como visto anteriormente, é possível com ele conceber sistemas embarcados que, alimentados por pilhas elétricas, operam durante um tempo considerável (dependendo da capacidade da bateria e do tipo de utilização, esse período pode ser superior a semanas).

Para que essa necessidade de energia seja ainda menor, é possível ao programador trabalhar com os *Low Power Modes* (cuja tradução poderia ser algo como Modos de Baixo Consumo), chamados também pelo seu acrônimo LPM. Esses modos proporcionam algumas mudanças na qualidade do processamento realizado, seja pela mudança de *clocks*, seja pela ativação e desativação de módulos periféricos não essenciais, ao colocar o processador em estado de dormência.

Os LPM são por vezes utilizados no código como laços (de modo similar a um *while* ou *for*) em situações diversas. Por exemplo, em *softwares* cuja rotina é baseada em interrupções, ao final do código principal pode-se utilizar um modo de baixo consumo. Dessa maneira, o processador entra em dormência, o *hardware* dos módulos continua em funcionamento (a depender do modo utilizado), e o consumo de energia é bastante reduzido. Assim, ao ocorrer uma interrupção, o processador é acordado novamente, realiza seu tratamento e volta ao estado de dormência.

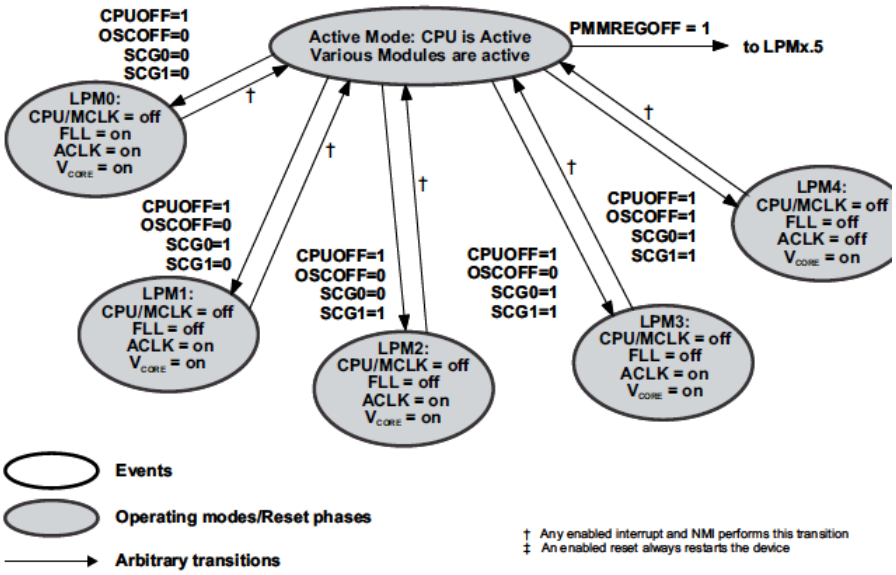
A seguir distinguem-se 5 modos diferentes de baixo consumo, tais quais LPM0, LPM1, LPM2, LPM3 e LPM4. Ainda há mais duas alterações possíveis nos modos 3 e 4, resultando em outros dois modos: LPM3.5 e LPM4.5. A entrada no *low power mode* é sempre controlada pelo programador, de modo que tem de ser ativada via *software*. Já a saída de um LPM para o modo ativo se dá por interrupções (pode ser também programada para ocorrer depois de determinado período de tempo). O diferencial de cada LPM é descrito a seguir:

- LPM0: Esse modo desliga o núcleo de processamento central por meio da desativação do *master clock*, que é sua fonte de *clock*.
- LPM1: Além de desligar a CPU, como o modo anterior, o LPM1 também desliga o *frequency locked loop* (FLL). O SMCLK pode ou não estar ativo, à escolha do programador (isso pode ser alterado no registro SMCLKOFF).
- LPM2: Esse modo, ao acréscimo das funcionalidades do LPM1, desliga o SMCLK, independentemente do registro SMCLKOFF.
- LPM3: O único *clock* que pode continuar funcionando nesse modo é o ACLK. Além disso, todas as alterações provocadas pelo LPM2 também existem no LPM3.
- LPM4: O LPM4, em suma, desliga todos os *clocks*. Apenas o V_{core} permanece ativo. A única maneira de sair desse modo de baixo consumo é provocando uma interrupção ou resetando o sistema, reiniciando-o.

- LPM3.5: Para além das funcionalidades do LPM3, o regulador também é desativado, fazendo com que não haja retenção de memória. O *real time clock* pode permanecer ativo, se configurado corretamente para isso.
- LPM4.5: Para além das funcionalidades do LPM4, o regulador também é desativado, fazendo com que não haja retenção de memória. Nenhuma fonte de *clock* é acessível nesse modo.

A Figura 5.13 demonstra como os campos do R2 (*Status Register*) são alterados para a seleção de cada LPM. Além disso, demonstra como o sistema alterna entre os modos de baixo consumo e o modo ativo.

Figura 5.13 – Diagrama de blocos dos *Low Power Modes*.



Fonte: [Texas Instruments 2015].

5.2.10 O *Watchdog*

O *Watchdog* (WDT: *WatchDog Timer*) é um temporizador cujo funcionamento é capaz de interromper o sistema, resetando-o. Ele funciona da seguinte maneira:

1. Durante um certo período de tempo, fica ocioso, apenas realizando sua contagem;
2. Em determinado momento, programado, ele sinaliza que necessita de um *feedback*.
3. Caso o sistema retorne uma determinada informação (*feedback*) para o *watchdog*, volta-se ao passo 1. Caso contrário, o WDT provoca o *reset* do sistema.

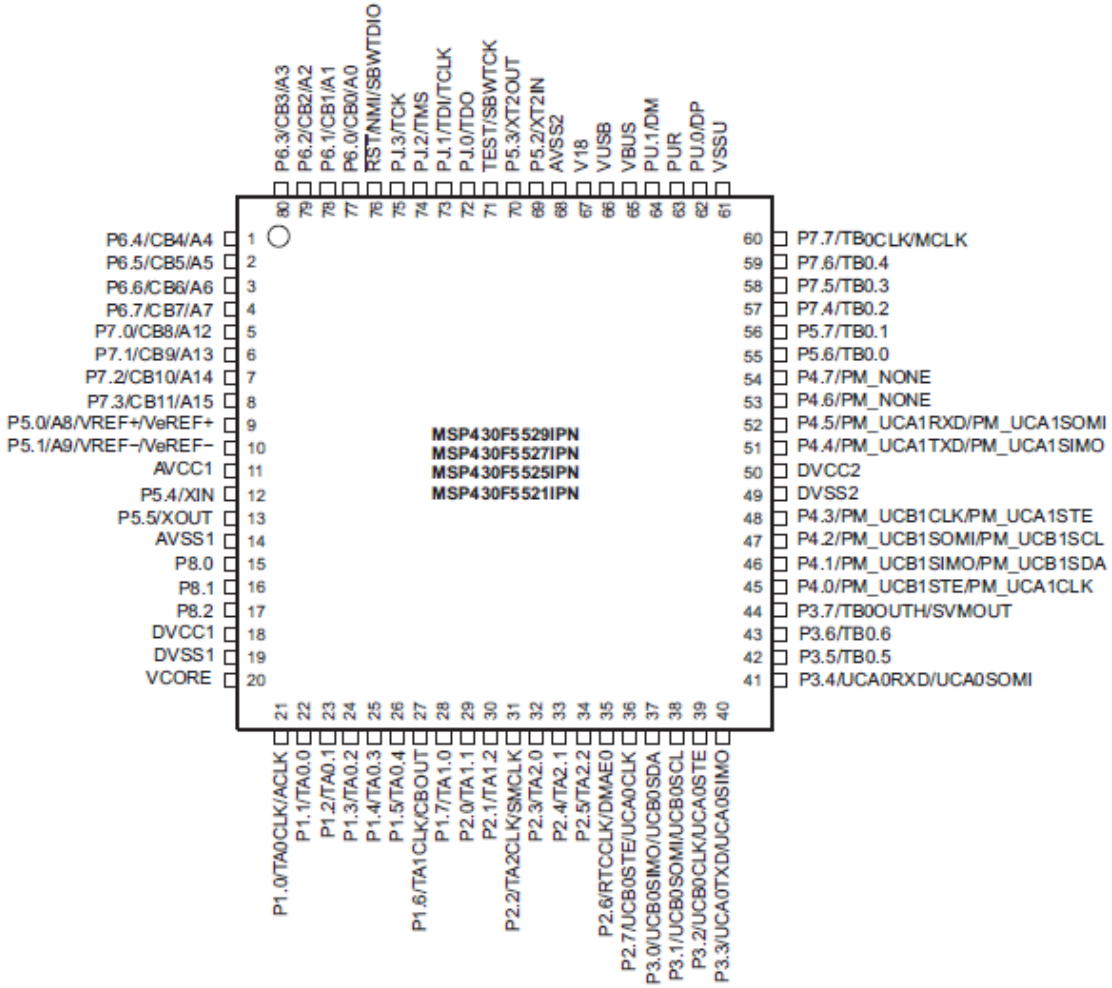
Ele é utilizado para que se garanta que o sistema não está em falha. Ou seja, caso um certo módulo apresente defeito e interrompa seu funcionamento, o WDT não receberá resposta e

resetará o controlador. Obviamente, o *watchdog* deve ser bem programado, assim como a resposta do sistema para seu bom funcionamento. De modo geral, na maior parte das aplicações o WDT é desativado para que não cause nenhum *crash* de um sistema, seja porque se saiba que um perifério não apresentará defeitos ou simplesmente porque não se enseje tanta segurança de operação.

Uma outra funcionalidade do WDT é que pode-se mudar seu resultado de *reset* para interrupção, utilizando-o como um temporizador de intervalo comum (ele realiza a contagem e, ao final, em caso de não haver *feedback*, causa uma interrupção).

Apêndice C - Esquemático da pinagem do MSP430F5529

Figura 5.14 – Pinagem do controlador MSP430F5529.



Fonte: [Texas Instruments 2013].

Apêndice D - Códigos e Datasheet do HC-SR04

```
1 #ifndef SETUPUS_H_
2 #define SETUPUS_H_
3
4 void setupUS();
5
6 #endif /* SETUPUS_H_ */
7
```

```
1 #ifndef SETUPUCS_H_
2 #define SETUPUCS_H_
3
4 void setupUCS();
5
6 #endif /* SETUPUCS_H_ */
7
```

```
1 #ifndef SETUPADC_H_
2 #define SETUPADC_H_
3
4 void setupADC();
5
6 #endif /* SETUPADC_H_ */
7
```

```c
#include <msp430.h>

void setupUCS() {


    UCSCTL0  = 0x0000;          // Set lowest DCOx, MODxw
    UCSCTL1  = DCORSEL_5;       // Select DCO range to 16MHz (2.5MHz < Range5 < 54.1MHz)
    UCSCTL2 |= 244 | FLLD__2;   // Multiplier N: (N+1) * FLLD' * 32.768kHz = 16.056320 MHz, FLLD'=2
    UCSCTL3  = FLLREFDIV_0   |  // FLL divider 1;
               SELREF__XT1CLK;  //;DCO FLL source = XT1
    UCSCTL4  = SELA_0 |         // ACLK source = XT1;
               SELS_3 |         //;SMCLK source = DCOCLK;
               SELM_3;          //;MCLK source = DCOCLK
    UCSCTL6 &= ~XT1OFF;         // Ensure XT1 is on

    __delay_cycles(250000);     // Accomodation cycles

}                               // Close setupUCS()

```

```
1 ////////////////////////////////////////////////////////////////
2 //                   UNIVERSIDADE DE BRASILIA                 //
3 //            DEPARTAMENTO DE ENGENHARIA ELETRICA             //
4 //              TRABALHO DE CONCLUSAO DE CURSO 2              //
5 //                                                            //
6 // ORIENTADOR: PROF. DANIEL CHAVES CAFE                       //
7 // DESENVOLVIDO POR:                                          //
8 // DANIEL CAVALLARE PIRES   - MATRICULA: 09/0044444           //
9 // PEDRO MAURICIO DE SOUSA  - MATRICULA: 11/0073975           //
10 //                                                           //
11 // ESTE SOFTWARE E RESPONSAVEL PELO CONTROLE DO MODULO ADC12 COM  //
12 // O MICROCONTROLADOR MSP430, PELA CAPTURA E A DIGITALIZACAO DE 2 //
13 // SINAIS DE AUDIO.                                          //
14 //                                                           //
15 // BRASILIA, 1 DE DEZEMBRO DE 2016.                          //
16 //                                                           //
17 ////////////////////////////////////////////////////////////////
18
19 ////////////////////////////////////////////////////////////////
20 //                        HARDWARE PINNAGE                    //
21 //                            MSP-OUT                         //
22 //                                                            //
23 //   MSP430F5529LP          <--->             FUNCTION        //
24 //       P6.0 ------------------------------- ADC IN A0       //
25 //       P6.1 ------------------------------- ADC IN A1       //
26 //       P1.6 ------------------------------- ADC OUT 00      //
27 //       P6.6 ------------------------------- ADC OUT 01      //
28 //       P3.2 ------------------------------- ADC OUT 02      //
29 //       P2.7 ------------------------------- ADC OUT 03      //
30 //       P4.2 ------------------------------- ADC OUT 04      //
31 //       P4.1 ------------------------------- ADC OUT 05      //
32 //       P3.5 ------------------------------- ADC OUT 06      //
33 //       P3.6 ------------------------------- ADC OUT 07      //
34 //       P7.0 ------------------------------- ADC OUT 08      //
35 //       P6.4 ------------------------------- ADC OUT 09      //
36 //       P6.3 ------------------------------- ADC OUT 10      //
37 //       P6.2 ------------------------------- ADC OUT 11      //
38 //                                                            //
39 ////////////////////////////////////////////////////////////////
40
41
42
43 #include <msp430.h>
44
45 void setupADC() {
46
47
48     // GPIO configure
49      // ADC INPUT CHANNELS
50      P6SEL    |= BIT0 |     // P6.0 peripheral - ADC A0;
51                  BIT1;      //;P6.1 peripheral - ADC A1
52      P6REN    |= BIT0 |     // Resistors on for A0
53                  BIT1;      // Resistors on for A1
54      P6OUT    &=~BIT0 |     // Pull-down resistor for A0
55                  BIT1;      // Pull-down resistor for A1
56
57      // ADC OUTPUT CHANNELS
58      P1DIR    |= BIT6;      // P1.6 is input (ADC BIT 00)
59      P2DIR    |= BIT7;      // P2.7 is input (ADC BIT 03)
60      P3DIR    |= BIT2 |     // P3.2 is input (ADC BIT 02)
61                  BIT5 |     // P3.5 is input (ADC BIT 06)
62                  BIT6;      // P3.6 is input (ADC BIT 07)
```

```
63      P4DIR    |= BIT1 |     // P4.1 is input (ADC BIT 05)
64                  BIT2;      // P4.2 is input (ADC BIT 04)
65      P6DIR    |= BIT2 |     // P6.2 is input (ADC BIT 11)
66                  BIT3 |     // P6.3 is input (ADC BIT 10)
67                  BIT4 |     // P6.4 is input (ADC BIT 09)
68                  BIT6;      // P6.6 is input (ADC BIT 01)
69      P7DIR    |= BIT0;      // P7.0 is input (ADC BIT 08)
70      P1DS     |= BIT6;      // P1.6 high drive strength (ADC BIT 00)
71      P2DS     |= BIT7;      // P2.7 high drive strength (ADC BIT 03)
72      P3DS     |= BIT2 |     // P3.2 high drive strength (ADC BIT 02)
73                  BIT5 |     // P3.5 high drive strength (ADC BIT 06)
74                  BIT6;      // P3.6 high drive strength (ADC BIT 07)
75      P4DS     |= BIT1 |     // P4.1 high drive strength (ADC BIT 05)
76                  BIT2;      // P4.2 high drive strength (ADC BIT 04)
77      P6DS     |= BIT2 |     // P6.2 high drive strength (ADC BIT 11)
78                  BIT3 |     // P6.3 high drive strength (ADC BIT 10)
79                  BIT4 |     // P6.4 high drive strength (ADC BIT 09)
80                  BIT6;      // P6.6 high drive strength (ADC BIT 01)
81      P7DS     |= BIT0;      // P7.0 high drive strength (ADC BIT 08)
82
83    // Voltage Reference configure
84
85    REFCTL0  = REFMSTR | REFON; // 1,5V voltage reference
86
87    // ADC configure
88
89    ADC12CTL0 &= ~ADC12ENC;     // Ensure enable conversion is off
90
91    ADC12CTL0 |= ADC12SHT0_2  | // A0 sample time = 16 ADC12CLK cycles;
92                  ADC12SHT1_2  | //;A1 sample time = 16 ADC12CLK cycles;
93                  ADC12ON      | //;ADC12 on;
94                  ADC12OVIE    | //;ADC12 overflow interrupt enable;
95                  ADC12TOVIE;    //;ADC12 conversion time interrupt en.
96    ADC12CTL0 &=~ADC12MSC;      // Further S&C are not automatic;
97    ADC12CTL1 |= ADC12SHS_0   | // SC source = Timer_A2
98                  ADC12SHP     | // S&H pulse sourced from timer
99                  ADC12DIV_0   | //;Clock divider = 1;
100                 ADC12SSEL_2  | //;Clock source = MCLK = 16MHz;
101                 ADC12CONSEQ_1; //;Repeated channels conversion
102   ADC12CTL2 |= ADC12TCOFF    | // Temperature sensor off;
103                 ADC12RES_2 ;   //;12 bit resolution;
104   ADC12MCTL0|= ADC12SREF_0  | // Reference = AVcc-AVss;
105                 ADC12INCH_0;   //;MEM0 input channel = A0 (P6.0)
106   ADC12MCTL1|= ADC12EOS     | // End of sequence (last channel);
107                 ADC12SREF_0  | //;Reference = AVcc-AVss;
108                 ADC12INCH_1;   //;MEM1 input channel = A1 (P6.1)
109   ADC12IE    = 0x03;          // Enable interrupt for A0 and A1
110
111   // Timer Configure
112
113   TA2CTL     = TASSEL_2 |     // SMCLK = 16MHz;
114                 MC_1     |     //;UP mode;
115                 TACLR    |     //;Clear counter;
116                 TAIE;          //;Interrupt enable.
117   TA2CCR0    = 180;           // At 16Mhz 180 cycles = [1/(2(44100Hz))]s
118
119 }                              // Close setupADC()
120
```

```
1 ////////////////////////////////////////////////////////////////
2 //                   UNIVERSIDADE DE BRASILIA                 //
3 //            DEPARTAMENTO DE ENGENHARIA ELETRICA             //
4 //              TRABALHO DE CONCLUSAO DE CURSO 2              //
5 //                                                            //
6 // ORIENTADOR: PROF. DR. DANIEL CHAVES CAFE                   //
7 // DESENVOLVIDO POR:                                          //
8 // DANIEL CAVALLARE PIRES   - MATRICULA: 09/0044444           //
9 // PEDRO MAURICIO DE SOUSA  - MATRICULA: 11/0073975           //
10 //                                                           //
11 // ESTE SOFTWARE E RESPONSAVEL PELO CONTROLE DOS MODULOS     //
12 // ULTRASSONICO HC-SR04 COM O MICROCONTROLADOR MSP430 E      //
13 // ANALOGIC-DIGITAL CONVERTER DO MESMO MICROCONTROLADOR, PELA //
14 // AQUISICAO DOS DADOS E PELA MISTURA PONDERADA DE DOIS SINAIS //
15 // ANALÓGICOS.                                               //
16 //                                                           //
17 // BRASILIA, 27 DE NOVEMBRO DE 2016.                         //
18 //                                                           //
19 ////////////////////////////////////////////////////////////////
20
21 ////////////////////////////////////////////////////////////////
22 //                        HARDWARE PINNAGE                    //
23 //                             MSP-OUT                        //
24 //                                                            //
25 //        MSP430F5529LP          <--->          FUNCTION      //
26 //         5V ------------------------------- Vcc             //
27 //        GND ------------------------------- GND             //
28 //       P6.0 ------------------------------- ADC IN A0       //
29 //       P6.1 ------------------------------- ADC IN A1       //
30 //       P6.2 ------------------------------- ADC OUT 11      //
31 //       P6.3 ------------------------------- ADC OUT 10      //
32 //       P6.4 ------------------------------- ADC OUT 09      //
33 //       P7.0 ------------------------------- ADC OUT 08      //
34 //       P3.6 ------------------------------- ADC OUT 07      //
35 //       P3.5 ------------------------------- ADC OUT 06      //
36 //       P4.1 ------------------------------- ADC OUT 05      //
   //
37 //       P4.2 ------------------------------- ADC OUT 04      //
38 //       P2.7 ------------------------------- ADC OUT 03      //
39 //       P3.2 ------------------------------- ADC OUT 02      //
40 //       P6.6 ------------------------------- ADC OUT 01      //
41 //       P1.6 ------------------------------- ADC OUT 00      //
42 //       P1.2 ------------------------------- Trigger         //
43 //       P1.3 ------------------------------- Echo            //
44 //        3V3 -------------------------------    -            //
45 //       P6.5 -------------------------------    -            //
46 //       P3.4 -------------------------------    -            //
47 //       P3.3 -------------------------------    -            //
48 //                                                            //
49 ////////////////////////////////////////////////////////////////
50
51 #include <msp430.h>
52 #include "setupADC.h"
53 #include "setupUS.h"
54 #include "setupUCS.h"
55
56     // Global variables
57      //US variables
58      volatile unsigned int
59      pulseWidth=1,
60      fallEdge=1100,
61      riseEdge=1000;
```

```
 62
 63      //ADC variables
 64      volatile unsigned int
 65      clSignal,                   // Stores clean signal after ADC;
 66      fxSignal,                   //;stores FX signal after ADC;
 67      outSignal,                  //;stores definitive digital signal after processing
 68      multFactor=1,               //;multiplication operand for result
 69      cl=1,                       //;clSignal bit-shifted
 70      fx=1,                       //;fxSignal bit-shifted
 71      clPond=1,                   //;first signal balanced
 72      fxPond=1;                   //;second signal balanced.
 73
 74
 75
 76
 77 void main(){
 78
 79
 80     WDTCTL = WDTPW + WDTHOLD;    // Stop WDT
 81
 82     setupUCS();                  // Configure clock system
 83     setupUS();                   // Configure ultrassonic system
 84     setupADC();                  // Configure ADC system
 85
 86     ADC12CTL0 |= ADC12ENC      | // Enable ADC conversion
 87                  ADC12SC;       // Start ADC S&C
 88
 89     __enable_interrupt();        // Enable GIE
 90     _low_power_mode_0();         // Enters LPM0 (CPU, MCLK OFF; FLL, ACLK, SMCLK ON)
 91
 92 }                                // Close main()
 93
 94
 95 ////////////////////////////////////////////////////////////////////////////////////
 96 ////////////////////////////////////////////////////////////////////////////////////
 97
 98     // Interrupt Service Routine (sensing process interrupts)
 99 #pragma vector=TIMER0_A0_VECTOR // Interrupt Service Routine for TA0CCR0
100  __interrupt void TA0_CCR0_ISR(void){
101 }                                // Close TA0_CCR0_ISR
102
103 #pragma vector=TIMER0_A1_VECTOR // Interrupt Service Routine for TA0 non-CCIFG0
104  __interrupt void TA0_ISR(void){
105     switch(TA0IV) {
106     case 2:                      // Interrupt source TA0CCR1
107         P1OUT &= ~BIT2;          // Set P1.2 off (STOP TRIGGER)
108         break;
109     case 4:                      // Interrupt source TA0CCR2 (RISING AND FALLING EDGES)
110         if (TA0CCTL2 & CCI)      // CCR2 interrupt (RISING edge)
111          riseEdge = TA0CCR2;     // Store rising edge time value
112         else {                   // CCR2 interrupt (FALLING edge)
113          fallEdge = TA0CCR2;     // Store falling edge time value
114          pulseWidth =            // Number of timer cycles taken
115          (fallEdge-riseEdge);
116         }                        // Close if-else
117         break;
118     case 0xE:                    // Interrupt source TAIFG (CCR0)
119         P1OUT  |=  BIT2;         // Set P1.2 on (START TRIGGER)
120         break;
121     default: break;
122     }                            // Close Switch-Case
123 }                                // Close TA0_ISR
```

```
124
125
126 #pragma vector=TIMER1_A0_VECTOR // Interrupt Service Routine for TA1CCR0
127  __interrupt void TA1_CCR0_ISR(void){
128 }                               // Close TA1_CCR0_ISR
129
130
131 #pragma vector=TIMER1_A1_VECTOR // Interrupt Service Routine for TA1 non CCIFG0
132 __interrupt void TA1_ISR(void){
133     switch(TA1IV) {
134     case 2:                     // Interrupt source TA1CCR1 (pulse ending)
135         P1OUT &= ~BIT0;         // Reset P1.0 (RedLed OFF)
136         break;
137     case 0xE:                   // Interrupt source TA1CCR0 (PWM Overflow, pulse beginning)
138         P1OUT |=  BIT0;         // Set P1.0 (RedLed ON)
139         break;
140     default: break;
141     }                           // Close switch-case
142
143     TA1CCR1 = pulseWidth;       // TA1CCR1 is related to last DutyCycle
144 }                               // Close TA1_ISR
145
146
147     // Interrupt Service Routines (ADC process interrupts)
148
149 #pragma vector=TIMER2_A0_VECTOR // Interrupt Service Routine for TA2CCR0
150 __interrupt void TA2_CCR0_ISR(void){
151 }                               // Close TA2_CCR0_ISR
152
153 #pragma vector=TIMER2_A1_VECTOR
154 __interrupt void TA2_ISR(void){
155     switch(__even_in_range(TA2IV,14)){
156     case 0: break;
157     case 2: break;
158     case 4: break;
159     case 6: break;
160     case 8: break;
161     case 10: break;
162     case 12: break;
163     case 14:
164         ADC12CTL0 |= ADC12SC;
165         ADC12CTL0 &=~ADC12SC;
166         break;
167     default: break;
168 }
169
170 }
171 #pragma vector = ADC12_VECTOR
172 __interrupt void ADC12_ISR(void){
173     switch(__even_in_range(ADC12IV,36)){
174     case  0: break;                           // No interrupt pending
175     case  2: break;                           // ADC overflow
176     case  4: break;                           // ADC timing overflow
177     case  6:                                  // ADC12IFG0
178         ADC12IFG &=~ ADC12IFG0;
179         break;
180     case  8:                                  // ADC12IFG1
181         clSignal = (unsigned int)             // clSignal assumes A0
182                    ADC12MEM0;
183         fxSignal = (unsigned int)             // fxSignal assumes A1
184                    ADC12MEM1;
185
```

```
186         if (pulseWidth>256)
187             pulseWidth=128;                 // (1/32768)*128*(1/2)=DIST/[VEL. AR
    (340.29m/s)] =~66,5cm
188         multFactor= pulseWidth>>1;
189         cl        = (clSignal)>>2;          // deslocamento de bit do sinal cl
190         fx        = (fxSignal)>>2;          // deslocamento de bit do sinal fx
191         MPY       = multFactor;             // multiplicador 1: pulseWidth
192         OP2       = cl;                     // multiplicador 2: sinal cl deslocado
193         clPond    = (RESLO);                // resultado 1: sinal cl na proporcao da
    distancia
194         MPY       = 64 - multFactor;        // multiplicador 1: complemento de pulseWidth
195         OP2       = fx;                     // multiplicador 2: sinal fx deslocado
196         fxPond    = (RESLO);                // resultado 2: sinal fx na proporção inversa
    de cl
197         outSignal = ((clPond+fxPond)>>3);   // soma balanceada dos resultados com ganho 2
198
199
200         //Habilitação das saídas
201         if (outSignal&BITB)
202             P6OUT |= BIT2;
203         else P6OUT &= ~BIT2;
204         if (outSignal&BITA)
205             P6OUT |= BIT3;
206         else P6OUT &= ~BIT3;
207         if (outSignal&BIT9)
208             P6OUT |= BIT4;
209         else P6OUT &= ~BIT4;
210         if (outSignal&BIT8)
211             P7OUT |= BIT0;
212         else P7OUT &= ~BIT0;
213         if (outSignal&BIT7)
214             P3OUT |= BIT6;
215         else P3OUT &= ~BIT6;
216         if (outSignal&BIT6)
217             P3OUT |= BIT5;
218         else P3OUT &= ~BIT5;
219         if (outSignal&BIT5)
220             P4OUT |= BIT1;
221         else P4OUT &= ~BIT1;
222         if (outSignal&BIT4)
223             P4OUT |= BIT2;
224         else P4OUT &= ~BIT2;
225         if (outSignal&BIT3)
226             P2OUT |= BIT7;
227         else P2OUT &= ~BIT7;
228         if (outSignal&BIT2)
229             P3OUT |= BIT2;
230         else P3OUT &= ~BIT2;
231         if (outSignal&BIT1)
232             P6OUT |= BIT6;
233         else P6OUT &= ~BIT6;
234         if (outSignal&BIT0)
235             P1OUT |= BIT6;
236         else P1OUT &= ~BIT6;
237        break;
238
239     case 10: break;                         // ADC12IFG2
240     case 12: break;                         // ADC12IFG3
241     case 14: break;                         // ADC12IFG4
242     case 16: break;                         // ADC12IFG5
243     case 18: break;                         // ADC12IFG6
244     case 20: break;                         // ADC12IFG7
```

```
245     case 22: break;                         // ADC12IFG8
246     case 24: break;                         // ADC12IFG9
247     case 26: break;                         // ADC12IFG10
248     case 28: break;                         // ADC12IFG11
249     case 30: break;                         // ADC12IFG12
250     case 32: break;                         // ADC12IFG13
251     case 34: break;                         // ADC12IFG14
252     case 36: break;                         // ADC12IFG15
253     default: break;
254     }                                       // Close switch-case
255 }
256
```

Ultrasonic Ranging Module HC - SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:
(1) Using IO trigger for at least 10us high level signal,
(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.
(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.
Test distance = (high level time×velocity of sound (340M/S) / 2,

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

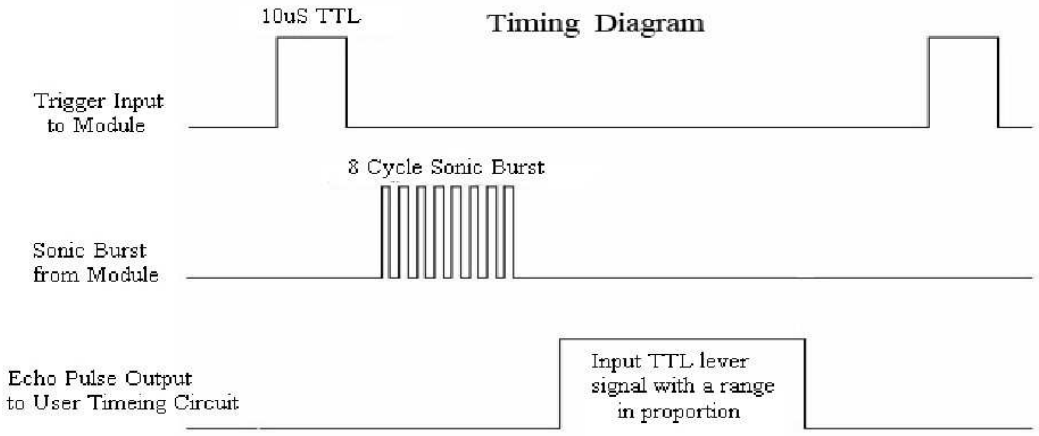
Electric Parameter

Working Voltage	**DC 5 V**
Working Current	**15mA**
Working Frequency	**40Hz**
Max Range	**4m**
Min Range	**2cm**
MeasuringAngle	**15 degree**
Trigger Input Signal	**10uS TTL pulse**
Echo Output Signal	**Input TTL lever signal and the range in proportion**
Dimension	**45*20*15mm**

Vcc Trig Echo GND

Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



Attention:

- The module is not suggested to connect directly to electric, if connected electric, the GND terminal should be connected the module first, otherwise, it will affect the normal work of the module.
- When tested objects, the range of area is not less than 0.5 square meters and the plane requests as smooth as possible, otherwise ,it will affect the results of measuring.