Um dispositivo para transmissão de imagens de um smartphone para um projetor multimídia via rede sem fio

Brucce Neves dos Santos[1]
brucce.neves@gmail.com

[1]Campus de Três Lagoas (CPTL)
Universidade Federal de Mato Grosso do Sul (UFMS)
Caixa Postal 210 – 79620-080 – Três Lagoas – MS – Brasil

Resumo. *Apresentações de slides utilizando projetores multimídia são recursos cada vez mais presentes em ambientes educacionais, em empresas e na vida cotidiana. Este trabalho propõe a implementação de um dispositivo de baixo custo, utilizando um Arduino, que permita um smartphone transmitir uma apresentação de slides convertida em imagens para o projetor multimídia sem a utilização de cabos, de forma a dar mais mobilidade ao usuário e fluidez às apresentações.*

1. Introdução

A apresentação de slides com o auxílio de um projetor multimídia é um recurso útil que nos permite expor ideias, trabalhos, ou qualquer outro conteúdo, de forma organizada, dinâmica e clara. A possibilidade de incluir imagens, diagramas, gráficos e vários outros detalhes nos slides, torna esse recurso ainda mais interessante e indispensável em palestras, reuniões, sala de aula, etc. A utilização de projetores multimídia para a apresentação de slides tem como outra vantagem o melhor aproveitamento do tempo, uma vez que toda a informação visual será projetada e o foco ficará apenas na comunicação verbal [1].

Apesar das grandes vantagens da utilização de um projetor multimídia, seu uso também pode trazer alguns inconvenientes, como incompatibilidade entre o conector de vídeo do projetor multimídia com o conector do computador, problemas no cabo de transmissão, etc. Além da necessidade de usar um computador, e de ter sempre uma pessoa ao lado desse equipamento no momento que precisar fazer as trocas de slides. Por exemplo, dependendo do local onde a apresentação está ocorrendo, pode ser que o usuário tenha que ficar longe do computador e com isso necessite de uma pessoa para passar os slides para ele, o que acaba interferindo na fluidez da apresentação, visto que a pessoa que está trocando os slides não é a mesma que está apresentando.

Neste trabalho propomos a implementação de um dispositivo de baixo custo que permite projetar os slides em um projetor multimídia sem a necessidade do uso de computadores, evitando problemas como defeitos em cabos, incompatibilidade dos conectores de vídeo, assim como permitir ao usuário uma maior mobilidade no local da apresentação.

Este artigo está organizado em seis seções, incluindo essa introdução. Na seção 2 é feita uma revisão dos principais conceitos necessários para o entendimento da proposta, nas seções 3, 4 e 5 são apresentados, reespectivamente, o problema a ser atacado, os objetivos e a metodologia utilizada no desenvolvimento do trabalho. Uma conclusão sobre o trabalho é realizada na seção 6 e, por fim, são apresentadas as referências bibliográficas utilizadas.

2. Conceitos Básicos

Nessa seção apresentamos os principais conceitos necessários para o melhor entendimento da solução proposta nesse trabalho como, por exemplo, os dispositivos escolhidos para o desenvolvimento do projeto e também um pouco sobre como é feita a exibição de uma imagem em um monitor ou projetor multimídia.

2.1. Smartphone

O Smartphone é um telefone celular que tem a capacidade de executar muitas das funções de um computador e normalmente possui uma tela sensível ao toque, capacidade de se conectar à redes Wi-Fi e um sistema operacional capaz de executar aplicativos [2].

O sistema operacional que está presente na maioria dos smartphones atuais é o Android. Esse sistema é baseado no núcleo do Linux e foi desenvolvido pela Android Inc em 2003. No ano de 2005, a Android Inc foi adquirida pela Google, que passou a desenvolver o sistema desde então. Esse sistema operacional é de código-fonte aberto e os aplicativos são feitos utilizando a linguagem de programação Java, com o auxílio de algumas bibliotecas extras para atender as peculiaridades do Android [3, 4, 5, 6, 7].

2.2. Arduino

O Arduino é uma plataforma de prototipagem eletrônica de hardware livre, que possui um microcontrolador e provê suporte à entrada e saída de dados, podendo esses dados serem analógicos ou digitais. O Arduino possui uma linguagem de programação padrão, muito semelhante ao C/C++, que tem como principal objetivo criar dispositivos de baixo custo [8].

Existem varios modelos de Arduino disponiveis, utilizamos nesse trabalho o modelo Due. O Arduino Due, possui um microcontrolador da ATMEL SAM3X8E ARM Cortex-M3, com CPU de 32-bits e 84MHz de *clock*. Tem a característica de ter sido a primeira placa da Arduino baseada em um núcleo de 32-bits ARM [9, 10].

Além do Due, utilizamos também o CC3000, que é um módulo Wi-Fi criado pela *Texas Instruments*, para ser um processador de rede sem fio, independente, que simplifica a implementação de conectividade com a internet, com intuito de minimizar os requisitos do hospedeiro. Esse módulo não foi desenvolvido especificamente para o Arduino mas pode ser utilizado sem perda de desempenho e com certa facilidade nesses dispositivos [11, 12, 13].

2.3. Exibição de imagens por um dispositivo de exibição

Dispositivos de exibição, tais como monitores ou projetores, têm a finalidade de exibir informações através de imagens. Dessa maneira, uma imagem pode ser considerada como a representação visual de uma informação. Na memória de um computador, uma imagem é representada como uma matriz de pixels.

Um pixel, do inglês *picture elements*, é o menor elemento de tela ou imagem ao qual podemos atribuir uma cor. Essa cor provém do modelo RGB, do inglês *Red Green Blue*. Para colorir um pixel são utilizados 8 bits para representar cada umas das cores do modelo RGB, totalizando 24 bits por pixel. Os primeiros 8 bits representam a variação do azul, os próximos 8 bits a variação do verde e os últimos 8 bits representam a variação

do vermelho. Cada uma das cores do modelo podem variar em 256 tons, do mais escuro ao mais claro, formando assim cerca de 16 milhões de possíveis cores para cada pixel [14, 15].

Uma das maneiras de transmitir uma imagem para um dispositivo de exibição é utilizando o padrão SVGA. O padrão VGA, do inglês *Video Graphics Array*, é um padrão criado pela *International Business Machine* (IBM), que foi implantado nos computadores fornecidos por essa empresa no ano de 1987. O padrão VGA foi desenvolvido com a finalidade de transmitir uma representação gráfica do que acontece no computador para monitores, porém com a popularização do padrão ele foi adotado em aparalhos televisores e projetores multimídia. Esse padrão atinge uma resolução máxima de 640x480 pixels [16].

A fim de alcançar resoluções maiores que a fornecida pelo VGA, foi definido pela *Video Electronics Standards Association* (VESA) o padrão SVGA, ou Super-VGA, que é uma extensão do padrão VGA. O SVGA utiliza o mesmo conector que o VGA e, por esse motivo, o SVGA não conseguiu manter seu nome e é amplamente conhecido apenas como VGA. Nesse trabalho estaremos implementando o padrão SVGA, que é suportado pela maioria dos projetores multimídia, inclusive os mais antigos.

O padrão SVGA é composto por cinco sinais analógicos que são transmitidos por pinos no conector VGA. Os dados de um pixel são transmitidos pelos sinais R, G e B que carregam a informação de cor para o dispositivo de exibição. Esses sinais variam de 0V até 0.7V, indicando a intensidade de cada uma das cores no pixel que será exibido no dispositivo de exibição. Além desses sinais, ainda existe o sinal de sincronismo horizontal (*Hsync*), que tem como função informar ao dispositivo de exibição que acabamos de enviar uma linha e estamos indo para a próxima, e o sinal de sincronismo vertical (*Vsync*), que serve para informar ao dispositivo de exibição que acabamos de preencher a tela e estamos voltando para a primeira linha da tela. Os sinais de sincronismo atuam em duas polaridades, positivo e negativo, ou seja ligado ou desligado.

O preenchimento da tela de um dispositivo de exibição é realizado da esquerda para a direta e de cima para baixo, e é dividido em 4 etapas:

Pulso de sincronismo, representado na Figura 1 por HS e VS, é a etapa onde é emitido um pulso positivo por um determinado tempo e logo após esse pulso é desligado. Durante o pulso de sincronia não podem ser enviadas informações sobre cores para o dispositivo de exibição. Uma observação importante é que a polaridade do pulso varia de acordo com a resolução desejada, dessa maneira, se a polaridade fosse negativa, seria necessário desligar o sinal durante a etapa de sincronismo e nas demais etapas deixar ele ligado.

***Back Porch*,** representado na Figura 1 por HB e VB, é o período de tempo que o dispositivo de exibição utiliza para se preparar para a próxima etapa, durante esse período também não deve ser enviado informações sobre cores para o dispositivo de exibição.

Área Visível é a etapa onde é de fato exibida a imagem no dispositivo de exibição. Vale lembrar que os sinais R, G e B devem sempre ser atualizados ao mesmo tempo, para que o dispositivo de exibição exiba o pixel com a cor desejada. Essa é a única etapa onde são enviadas informações sobre as cores para o dispositivo de exibição.

***Front Porch*,** representado na Figura 1 por HF e VF, é um intervalo utilizado pelo dis-

positivo de exibição para se preparar para a próxima etapa. Durante esse período também não devem ser enviadas informações sobre cores.

Como pode ser notado, o *Back Porch* e o *Front Porch* servem para separar o pulso de sincronismo da área visível, essas etapas surgiram por conta que os monitores CRT (*Cathodic Ray Tube*) precisavam de um tempo para mover o feixe de elétrons do lado direito para o esquerdo e da última linha para o início da tela novamente. Na Figura 1 podemos ver a representação de como funcionam as etapas de preenchimento da tela pelo padrão VGA [17, 18].

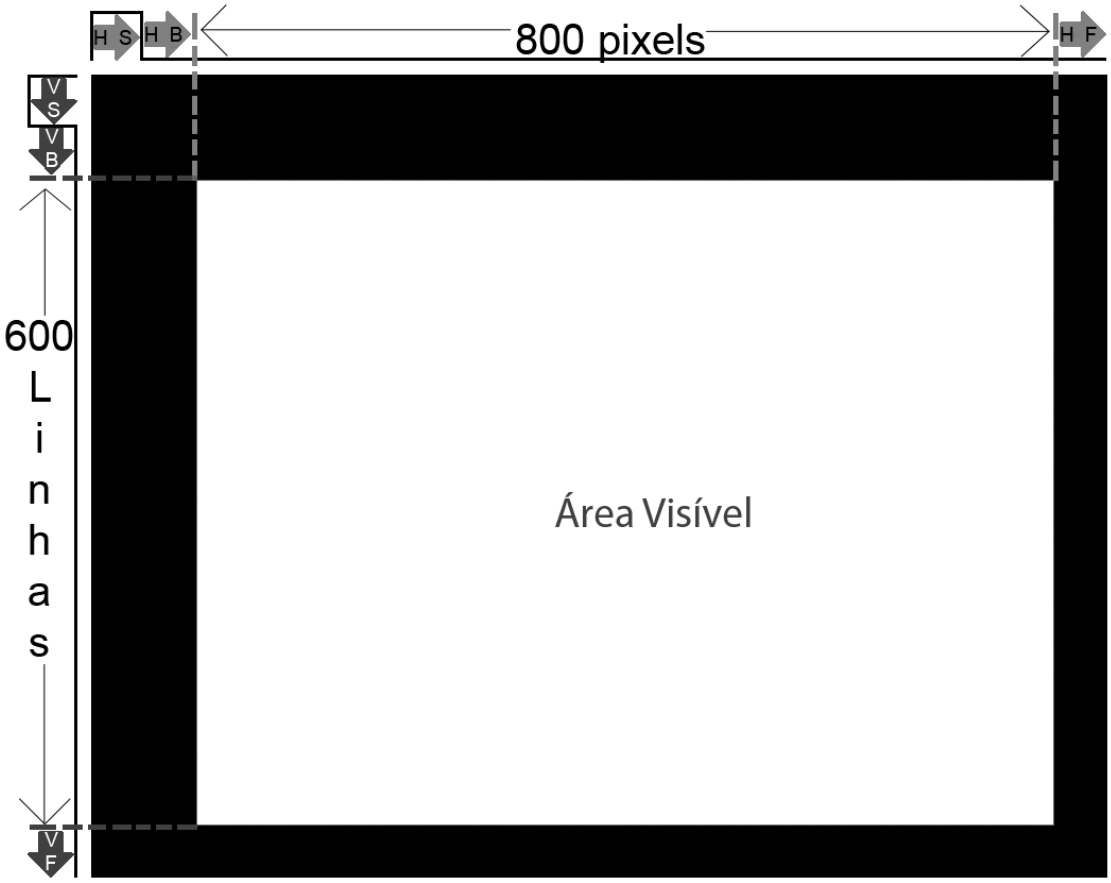

Figura 1. Representação do funcionamento dos sinais de sincronia. (tamanhos ilustrativos)

Cada uma das etapas de sincronia possuem um *timming*, ou seja, um tempo de duração, e esse tempo é imprescindível para a comunicação com o dispositivo de exibição. Na Tabela 1 são exibidos os *timmings* de cada uma das etapas, de acordo com a resolução utilizada no dispositivo de exibição.

Vale notar que a duração dos *timmings* de cada etapa do sincronismo vertical é aproximadamente a mesma em dispositivos de exibição com a mesma taxa de atualização, independentemente do tamanho da resolução desejada, mudando a polaridade do pulso de sincronia de ambos os sinais e a frequência do pixel, ou seja o tempo que deve ser dado entre a exibição de um pixel e outro.

	Horizontal		Vertical	
	Pixels	Duração	Linhas	Duração
Pulso de Sincronia	128	3,2 μs	4	105,4 μs
Back Porch	88	2,2 μs	23	607,2 μs
Área Visível	800	20 μs	600	15840 μs
Front Porch	40	1 μs	1	26,4 μs
Total	1056	26,4 μs	628	16579,2 μs

Tabela 1. *Timmings* de cada uma das etapas, em um dispositivo de exibição com taxa de atualização de 60Hz e resolução de 800×600 pixels [19].

Com esses conceitos em mente, fica mais fácil a compreensão da solução proposta para o problema descrito na seção a seguir.

3. Definição do problema e possíveis soluções

Os principais problemas encontrados na utilização de um projetor multimídia, além da necessidade de possuir um computador conectado a ele e de estar próximo ao equipamento toda vez que for realizar uma troca de slides, são as incompatibilidades do conector do projetor multimídia com o conector do computador e possíveis problemas no cabo de transmissão. Nesse contexto o problema é transmitir uma apresentação de slides convertida em imagens de um smartphone através de uma rede sem fio para o projetor multimídia.

Existem dispositivos disponíveis no mercado que solucionariam alguns desses problemas, como conversores e adaptadores de entradas e saídas de vídeo, por exemplo, que resolveriam o problema de incompatibilidade dos conectores de vídeo, mas ainda não estaríamos livres de defeitos nos cabos de transmissão.

Outra solução existente no mercado é o conjunto de transmissor e receptor sem fio para sinais de vídeo, que permite conectar um computador, ou em alguns casos até mesmo um smartphone e transmitir as imagens de um aparelho ao outro sem a necessidade de fios. Porém, alguns desses receptores e transmissores não possuem todos os tipos de conectores de vídeo e são passíveis de interferências nas imagens, dependendo das condições do local de utilização.

Para o inconveniente da falta de mobilidade do usuário, existem dispositivos que permitem realizar a troca de slides utilizando algo como um pequeno controle remoto e também aplicativos de smartphone que funcionam de forma semelhante a esses dispositivos. Mas em todos os casos é necessário possuir um computador ligado ao projetor multimídia.

A melhor solução atualmente no mercado são os projetores multimídia com Wi-Fi embutido, permitindo que computadores, e em alguns casos smartphones, transmitam imagens para o projetor multimídia sem a necessidade de cabos. Para poder realizar essa transmissão, através de uma rede sem fio, é necessário instalar um software fornecido pelo fabricante do projetor. A necessidade da utilização desse software, juntamente com o custo ainda não muito acessível são as principais desvantagens desses projetores. A necessidade do uso do software específico fornecido pelo fabricante, impede as vezes a utilização desses projetores em computadores com sistemas operacionais diferentes da-

queles suportados pelo fabricante do projetor.

4. Objetivos

Levantadas todas as dificuldades da Seção 3, o objetivo deste projeto é desenvolver um dispositivo de baixo custo, utilizando Arduino, que permita ao usuário transmitir uma apresentação em slides de um smartphone para um projetor multimídia, através de uma rede Wi-Fi, de forma que o usuário não perca sua mobilidade e não necessite de outros acessórios além de um smartphone com sistema operacional Android. O desenvolvimento desse dispositivo visa solucionar os problemas destacados na Seção 3 e, além disso, dar uma opção viável para projetores de gerações anteriores, que não possuem dispositivos de rede embutidos.

5. Metodologia

O ponto de partida para o desenvolvimento do projeto foi a criação de um aplicativo para Android, capaz de ler uma apresentação de slides no formato PDF (*Portable Document Format*), permitindo que o usuário navegue pelos slides do arquivo de forma intuitiva. O slide que está sendo exibido na tela do smartphone é convertido para uma imagem e deve ser enviado através de uma rede sem fio para o Arduino. O Arduino, por sua vez deve estar conectado à rede Wi-Fi, para receber os slides e também deve estar conectado ao projetor multimídia através do padrão SVGA para poder projetar o slide recebido pela rede. O padrão SVGA foi implementado do zero, uma vez que o Arduino que utilizamos não possui tal funcionalidade implementada. Na Figura 2 está ilustrado o funcionamento do projeto.

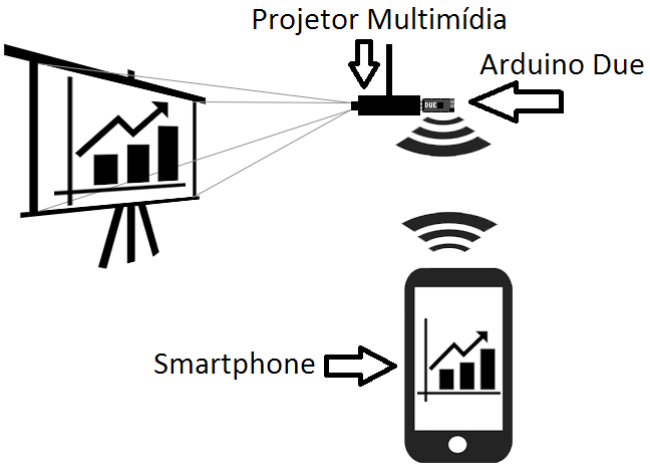

Figura 2. Representação do funcionamento do projeto.

Nas subseções seguintes descrevemos com um pouco mais de detalhes o desenvolvimento do projeto. Inclusive dedicamos a subseção 5.1 para ressaltar algumas limitações que tivemos e quais as ações tomadas para contornar tais limitações.

5.1. Limitações

Essa subseção descreve as principais limitações encontradas durante o desenvolvimento do projeto, tanto com relação às limitações de hardware do Arduino como às limitações de software do Android.

A maioria das limitações encontradas durante o trabalho foram com relação ao hardware, uma vez que o Due possui uma quantidade limitada de memória SRAM (96 kB [9, 10]). Com essa quantidade de memória não conseguimos armazenar uma imagem de 800×600 pixels que era o objetivo inicial, mesmo que utilizando um único byte para representar a cor de um pixel. Com isso optamos por armazenar uma imagem de 320×240 pixels e um byte para representar a cor do pixel. Por causa dessa limitação de apenas um byte para uma cor, decidimos trabalhar com escala de cinza, em vez de trabalharmos com uma quantidade limitada de cores uma vez que, em um byte, teríamos 3 bits para cor vermelha, 3 bits para cor verde e 2 bits para a cor azul. Em outras palavras, implementamos o padrão SVGA de 800×600 pixels porém enviando uma imagem de 320×240 pixels em escala de cinza.

Outra limitação de hardware que esbarramos foi com o módulo CC3000. Por ser um módulo que visa o baixo consumo de energia, ele possui um *buffer* muito pequeno, não permitindo receber pacotes de dados acima de 95 bytes ou muitos pacotes que somados ultrapassem esse valor. Caso isso ocorra, o CC3000 trava de tal forma a ser necessário reiniciar o Due para voltar ao estado normal de operação. O módulo também não possui a função de criar uma rede, sendo necessário deixar gravado as informações da rede a qual ele deve se conectar dentro do código ou utilizar o recurso *SmartConfig* [12, 13], o qual não será abordado nesse trabalho.

As limitações do Android são bem menores e podem variar de acordo com o dispositivo que está sendo usado, podendo até ocorrer o caso de não encontrar nenhum problema. Durante a execução do projeto esbarramos no problema da falta de memória RAM, e na perda de pacotes enviados pela rede. Essas limitações podem ser causadas, respectivamente, pelo excesso de aplicativos abertos no smartphone e pela má qualidade da rede em que os dispositivos estão conectados.

Colocadas essas dificuldades, as seções a seguir descrevem com mais detalhes as implementações realizadas tanto para o aplicativo Android quanto para o dispositivo Aduino.

5.2. Android

O aplicativo para Android foi desenvolvido em duas etapas. A primeira etapa consiste em ler o arquivo em formato PDF e exibir os slides na tela do smartphone. Para isso, foi preciso implementar um pequeno gerenciador de arquivos dentro do aplicativo que permite ao usuário navegar pelas pastas do dispositivo até encontrar o PDF desejado. Tendo encontrado o arquivo PDF, o aplicativo irá ler e exibir o primeiro slide do PDF, gerando duas cópias desse slide em resoluções diferentes. A primeira possuirá a dimensão da tela do smartphone e é utilizada para permitir que o usuário visualize o slide atual na tela do aparelho. A segunda cópia é renderizada com 320×240 pixels para que seja transmitida para o Arduino. Apesar da versão 5.0 do Android possuir uma biblioteca nativa para ler arquivos no formato PDF, a quantidade de aparelhos que suportam essa versão do Android é pequena. Segundo a *IDE Android Studio* somente 35,4% dos aparelhos atualmente possuem a versão do Android igual ou superior a 5.0. Dessa maneira utilizamos a biblioteca *Android PDF Viewer* [20], que foi desenvolvida para ter como requisito mínimo a versão 1.5 do Android. Assim o aplicativo poderá ser utilizado por uma maior quantidade de aparelhos, desde que possuam pelo menos a versão 1.5 do Android.

A segunda etapa consiste em transformar a cópia de 320×240 pixels do slide atual para escala de cinza e depois enviá-la pela rede Wi-Fi para o Arduino. Para se determinar como enviar as imagens pela rede, primeiro foram realizados testes com o objetivo de definir qual protocolo seria melhor aplicado para essa conexão, como os testes foram realizados antes de tomarmos conhecimendo das limitações descritas na Seção 5.1 utilizamos imagens com a resolução de 800×600 pixels. Esse teste consistia em enviar a mesma imagem, repetidas vezes de smartphone para um computador rodando *software* feito em Java que simulava o Arduino recebendo os pixels da imagem e os desenhavam na tela do computador. Foram testados três formas diferentes para o envio das imagens utilizando os protocolos TCP e UDP, para analisar qual se adequaria melhor a nossa necessidade. O primeiro teste consistia em enviar a imagem pixel a pixel. Nesse teste o protocolo TCP obteve sucesso em enviar a imagem completa por implementar a garantia de entrega de pacotes, porém levou um pouco mais tempo que o protocolo UDP para enviar a mesma quantidade de imagens. O problema encontrado no UDP, foi que ele nem sempre conseguia completar a imagem, pois como não tem garantia de entrega, se um pacote (pixel) se perdia, a imagem começava e ficar distorcida. No segundo teste a imagem também foi enviada pixel a pixel, porém cada pixel levava junto a posição x e y a qual ele pertencia na imagem original. No TCP não houve mudanças significativas, porém no UDP como o pixel possuia as coordenadas corretas de onde devia se encaixar, a imagem não ficava distorcida como no teste anterior, porém se um pixel se perdia a imagem ficava com esse pedaço faltando até que esse pixel fosse reenviado. No pior caso, o mesmo pixel poderia se perder em todos os envios, nesse sentido possuía uma chance da imagem ficar incompleta. O último teste consistiu em enviar a imagem linha a linha, e cada linha carregava consigo a sua posição y na imagem original. Nesse teste houve uma redução do tempo de envio das imagens tanto no TCP quanto no UDP, e o UDP conseguiu completar a imagem em um tempo razoável, tendo que reenviar a imagem poucas vezes para completá-la.

Feitos os testes, foi decidido que as imagens seriam enviadas com linhas identificadas, através do protocolo UDP, pois o mesmo inclui a vantagem de não ser orientado a conexões, o que permite enviar os pacotes em Broadcast pela rede, sem a necessidade de saber qual o IP que foi atribuido ao módulo CC3000 na rede. Porém como foi ressaltado na Seção 5.1, o CC3000 possui uma limitação de 95 bytes no *buffer* de leitura, o que nos impede de enviar uma linha inteira. Dessa maneira optamos por fragmentar uma linha em 10 partes, onde cada parte possui 34 bytes, sendo 32 bytes de pixels, um byte para identificar a linha e outro byte para identificar qual a ordem que esse fragmento deve aparecer na linha. Além disso, também colocamos um *delay* entre o envio de cada um desses fragmentos, no intuito de tentar impedir que o Android envie muitos pacotes e trave o CC3000.

5.3. Arduino

No Arduino, foi necessário implementar o padrão SVGA, em outras palavras fazer com que o Arduino se comunique com o dispositivo de exibição através dos pinos do conector VGA seguindo as etapas de sincronismo definidas por esse padrão, para isso foi utilizado a estrutura *Timer/Counter* que vem embutido dentro do microcontrolador. Para o Arduino poder se conectar à rede Wi-Fi incluimos o módulo CC3000.

O *Timer/Counter* (TC) é uma estrutura que permite ao programador configurar um evento síncrono que gera uma interrupção na CPU (*Central Processing Unit*), ou seja a

cada período de tempo o TC faz com que a unidade central de processamento pare o que está fazendo, execute uma determinada função e depois retome de onde parou. A grande vantagem de usar essa estrutura é que esse contador é incrementado de acordo com a oscilação do relógio do microcontrolador, dessa maneira não ocupa a CPU para efetuar essas tarefas. Utilizamos o TC para simular os sinais de sincronia.

O microcontrolador usado no Arduino Due possui três instâncias de TC idênticas, cada instância TC possui três canais, onde cada canal pode ser configurado de forma independente para executar uma ampla gama de funções. Dentre essas funções utilizamos a PWM (*Pulse Width Modulation*), para emitir um pulso durante um determinado tempo, em outras palavras essa função irá ligar um determinado pino do Arduino durante o período especificado e após esse tempo ele desligará automaticamente esse pino. Na Figura 3 é ilustrada a configuração do TC utilizado nesse trabalho.

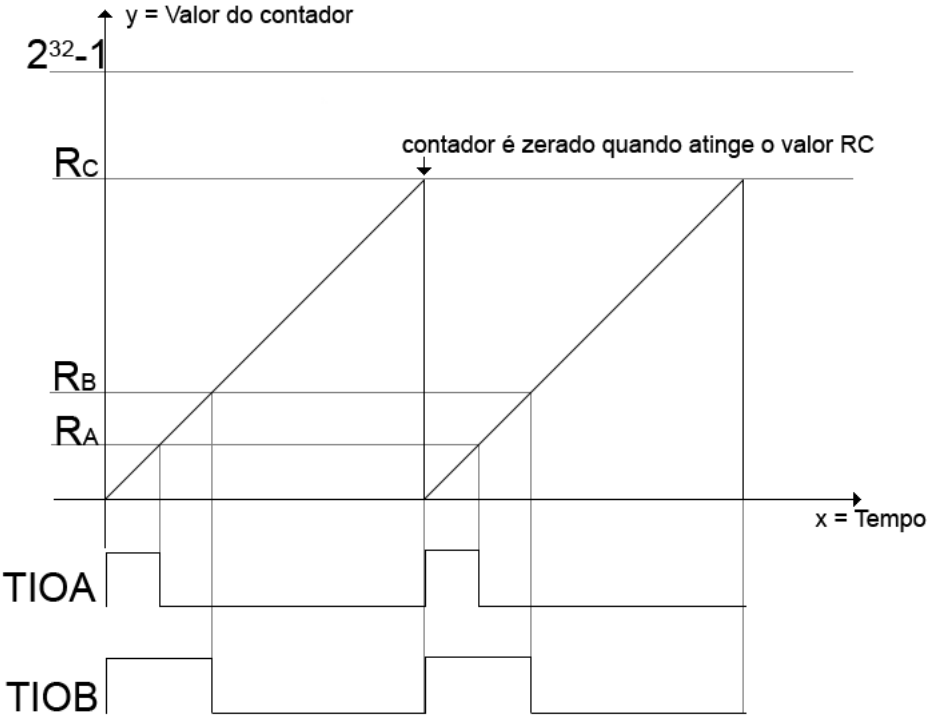

Figura 3. Representação do funcionamento do TC configurado para gerar interrupção por comparação de RC e utilizando o PWM [21, 22] (tamanhos ilustrativos).

Cada canal do TC possui um contador de 32 bits, que pode ser incrementado de 5 formas diferentes, dentre elas optamos pela $\frac{MasterClock}{2}$, onde o contador é incrementado a cada duas oscilações do relógio. Na solução proposta, o contador será incrementado até o valor de RC (*RC compare Interrupt*), que, quando alcançado gera uma interrupção e o ciclo é reiniciado, em outras palavras o valor de RC define o tamanho do ciclo. Quando uma interrupção acontece é chamada uma função denominada *TCx_Handler* (onde o x é um número que diferencia os nove canais), que é onde o programador pode colocar o trecho de código que ele quer que seja executado naquele momento, desde que o tempo de executar esse trecho de código seja menor que a duração do ciclo. A primeira coisa a ser feita dentro dessa função é ler o valor do TC *Status Register* (TC_SR), caso não seja

lido será gerado um erro no próximo ciclo.

Os canais também possuem três variáveis denominadas RA, RB e RC, que podem receber valores e serem configuradas para gerarem interrupções quando o contador atingir o valor especificado nelas. Porém RA e RB não fazem o ciclo recomeçar de forma automática e para serem utilizadas no próximo ciclo precisam ter seus valores atribuídos novamente. As variáveis RA e RB possuem um pino associado a cada uma delas, que são denominados, respectivamente, TIOA e TIOB (os pinos referenciados por TIOA e TIOB variam para cada canal), que podem ser ativados de forma independente caso queira usar o PWM [21, 22].

Para se configurar o TC precisamos, para cada etapa de sincronia, determinar o valor equivalente do contador, em outras palavras foi necessário descobrir o tempo que leva cada incremento do contador para então definir o valor do contador equivalente a cada etapa, sendo assim foi preciso calcular o tempo de exibição de um pixel e converter esse tempo para a frequência de atualização do contador do TC [17].

O primeiro passo para esse cálculo, consiste em converter a taxa de atualização do dispositivo de exibição de hertz (Hz) para microsegundos (μs), encontrando assim a duração de um *Frame Rate* em μs, ou seja, o tempo de exibir uma tela:

$$FrameRate = \frac{1}{TaxaDeAtualizacao} \times 10^6 \tag{1}$$

Encontrado o *Frame Rate*, foi necessário descobrir o tempo exibição de uma linha. Para isso basta dividir o *Frame Rate* pelo total de linhas de uma tela:

$$linha = \frac{\textit{FrameRate}}{VF + VS + VB + AreaVisivelVertical} \tag{2}$$

Após encontrado o tempo de exibição de uma linha, precisamos encontrar o tempo de exibição de um pixel. Para isso foi divido o tempo de uma linha pelo total de pixels em uma linha:

$$pixel = \frac{linha}{HF + HS + HB + AreaVisivelHorizontal} \tag{3}$$

Depois foi definida a frequência que será atualizado o contador com base no ciclo de *clock* do microcontrolador. Utilizamos nesse trabalho:

$$Freq.AtualizacaoContador = \frac{MasterClock}{2} \tag{4}$$

Definida a frequência de atualização do contador devemos convertê-la para microsegundos, encontrando assim o *Counter Rate*, ou seja, a frequência de atualização do contador em microsegundos:

$$CounterRate = \frac{1}{Freq.AtualizacaoContador} \times 10^6 \tag{5}$$

Feito isso podemos encontrar qual valor do contador é equivalente ao tempo de exibição de um pixel, sendo necessário somente dividir o tempo de exibição de um pixel,

encontrado anteriormente, pelo *Counter Rate*:

$$PixelCounter = \frac{pixel}{\textit{CounterRate}} \tag{6}$$

Tendo encontrado o *PixelCounter*, devemos encontrar o valor do contador para cada etapa do *Hsync*, sendo necessário apenas multiplicar esse resultado pela quantidade de pixels em cada etapa e subtrair 1 do resultado, subtraimos 1 pois o zero conta.

$$PixelCounter \times QtdPixel - 1$$

Feito isso para as 4 etapas e somando os resultados, teremos o *LineCounter*, ou seja, o valor total do contador para uma linha, e assim podemos encontrar os valores para cada etapa do *Vsync*.

Nesse trabalho utilizamos dois canais do TC, um para o *Vsync* outro para o *Hsync*, ambos configurados para funcionar de forma semelhante, como pode ser visto na Figura 3. Para uma melhor compreensão dos cálculos, utilizaremos a notação "_H" e "_V" após as variáveis dos TC para indicar se estamos configurando, respectivamente, o TC horizontal ou TC vertical. Por exemplo, RA_H, signifca que estamos nos referindo ao TC utilizado para configurar as etapas de sincronismo horizontal. Como foi dito, para definir o valor do contador para o pulso de sincronismo horizontal (HS) usamos:

$$HS\_Counter = pixelCounter \times HS - 1 \tag{7}$$

Atribuímos o valor do *HS_Counter*, obtido na fórmula anterior, ao RA_H, que está configurado para gerar o PWM. Também habilitamos o uso do pino TIOA e o conectamos ao pino do conector VGA do dispostivo de exibição. Como pode ser observado na Figura 3, o RA irá gerar um sinal positivo durante o tempo especificado, irá parar quando o contador alcançar o valor de RA e só voltará a ser ligado quando o ciclo reiniciar. De forma semelhante, definimos o valor do *Back Porch* horizontal (HB) usando.

$$HB\_Counter = pixelCounter \times HB - 1 \tag{8}$$

Atribuímos para o RB_H o resultado da soma do *HS_Counter* com *HB_Counter*, pois o *Back Porch* deve ser contado após o final do sinal de sincronismo. Utilizamos o RB_H para gerar uma interrupção após o *Back Porch*, que marca o início da etapa de área visível. Dessa maneira, o *Hsync* executará sua função *TCx_Handler* que irá iniciar a transmissão da imagem, todavia a imagem só será transmitida caso o *Vsync* também esteja na etapa de área visível. É importante enfatizar que executando o código desse jeito irá gerar um espaço entre o canto esquerdo do dispositivo de exibição e os pixels da primeira coluna da imagem transmitida, isso acontece pois não está sendo considerado o tempo que demora o processo do TC interromper a CPU até chegar na linha que irá iniciar a transferência da imagem. Essas informações não foram incluídas no cálculo pois demandam um conhecimento muito aprofundado sobre o microcontrolador usado.

De forma semelhante, podemos definir o valor do *Front Porch* (HF) e da área visível horizontal com um único cálculo:

$$HF\_V\_Counter = pixelCounter \times (HF + AreaVisivelHorizontal) - 2 \tag{9}$$

Para atribuir o valor de RC_H foi necessário definir o valor do contador referente ao tempo total de exibição de uma linha, para isso somamos todos os valores obtidos anteriormente:

$$LineCounter = HB\_Counter + HS\_Counter + HF\_V\_Counter \tag{10}$$

O RC_H será utilizado para criar o ciclo de uma linha, como pode ser observado na Figura 3. O RC é resetado toda vez que o contador alcança o valor dele.

Tendo encontrado o *LineCounter*, ou seja, o valor do contador para uma linha, podemos definir os valores para cada etapa do *Vsync* utilizando o seguinte cálculo:

$$LineCounter \times QtdLinhas$$

Dessa maneira, valor para o pulso de sincronismo vertical (VS) é definido por:

$$VS\_Counter = LineCounter \times VS \tag{11}$$

Atribuímos o valor do *VS_Counter* obtido na fórmula anterior ao RA_V, que como o RA_H está configurado para gerar o PWM. Também está habilitado o uso do pino TIOA e o conectamos no pino do conector VGA do dispostivo de exibição. De forma semelhante para definir o valor para a etapa de *Back Porch* vertical (VB) usamos:

$$VB\_Counter = LineCounter \times VB \tag{12}$$

Atribuímos para o RB_V o resultado da soma do *VS_Counter* com *VB_Counter*, pois o *Back Porch* deve ser contado após o final do sinal de sincronismo. Utilizamos o RB_V para gerar uma interrupção após o *Back Porch*, que marca o início da etapa de área visível. Dessa maneira, o *Vsync* executa sua função *TCx_Handler* informando ao *Hsync* que a etapa de área visível vertical foi iniciada.

De forma semelhante, definimos o valor do *Front Porch* (VF) e da área visível vertical:

$$VF\_V\_Counter = pixelCounter \times (VF + AreaVisivelVertical) \tag{13}$$

O RC_V também é utilizado para criar o ciclo de um *Frame*, de forma semelhante ao RC_H. Para atribuir o valor de RC_V foi necessário definir o valor do contador referente ao tempo total de um *Frame*, para isso somamos todos os valores obtidos anteriormente.

$$FrameCounter = VB\_Counter + VS\_Counter + VF\_V\_Counter \tag{14}$$

Na Figura 4 é mostrado como deve ser a conexão do Arduino com o conector do dispositivo de exibição para que funcione corretamente.

Uma vez definido como seriam controlados os sinais de sincronismo foi preciso definir um mecanismo para realizar a transferência dos dados dos pixels da imagem que está armazenada na memória RAM do Arduino para o dispositivo de exibição. Esse processo não poderia ser controlado apenas pela CPU do Arduino, uma vez que a quantidade

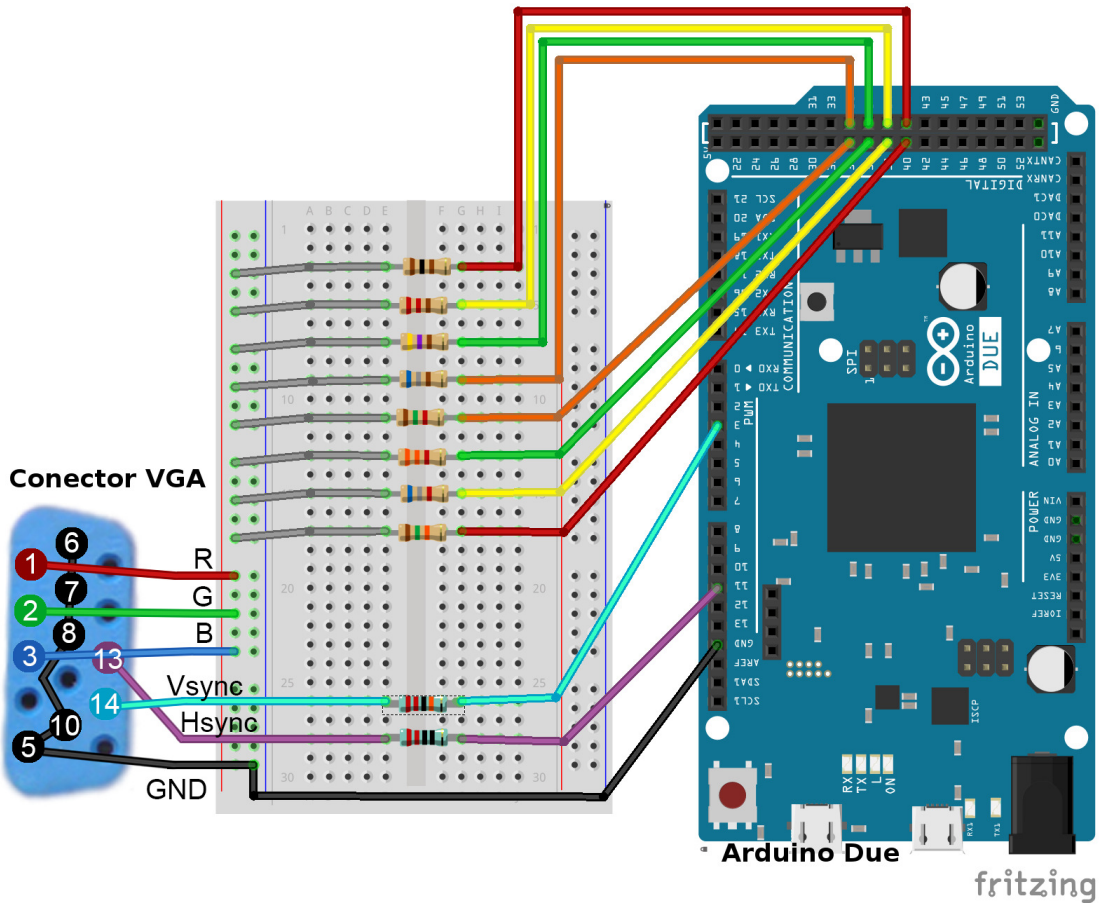

Figura 4. Representação de como se conectar o Arduino Due no conector fêmea VGA (tamanhos ilustrativos).

de dados a ser processados para mostrar uma imagem na tela em tempo real é muito grande, e com isso não conseguiríamos executar mais nenhuma tarefa na CPU. Optamos então pela utilização do *Direct Memory Access Controller* (DMAC), que é uma estrutura embutida no microcontrolador do Arduino Due. O DMAC tem como finalidade transferir dados de um periférico para outro sem a intervenção da CPU do Arduino. Para seu funcionamento precisamos informar qual é a origem dos dados e qual o destino, também precisamos informar a quantidade de dados a serem transferidos e mais algumas configurações referentes a como devem ser transferidos esses dados de um lado para outro [21, 23, 24].

Além do DMAC, foi necessário o uso de uma segunda estrutura para coordenar essa transferência, pois não podemos simplesmente transferir os dados em qualquer velocidade, já que os dados que estão sendo transferidos são pixels para o dispositivo de exibição e é necessário haver uma pequena pausa entre o envio de um pixel e outro. Para isso utilizamos a SMC (*Static Memory Controller*) que também vem embutido no microcontrolador, essa estruturua foi projetada para assegurar o êxito da transfêrencia de dados entre o microcontrolador e os dispositivos externos. Ela gera sinais que controlam o acesso a periféricos ou dispositivos de memória externa, em outras palavras ela permite ao programador gerenciar a velocidade que serão enviados os dados para o destino com base na oscilação do relógio. A principal configuração dessa estrutura é a quantidade de oscilações que ela deve aguardar entre a transferência de um dado e outro [21].

Neste trabalho utilizamos um vetor de bytes com 77040 posições para armazenar os pixels, o tamanho desse vetor vem da dimensão da imagem que é 320×240 mais 240 bytes. Importante ressaltar que incluimos um byte a mais em cada linha para que o DMAC, após enviar a linha toda, desligue automaticamente o envio de informações de cores, deixando os sinais de cores preparados para o *Front Porch*. Note que esse byte que foi incluido não será alterado, e sempre possuirá valor zero.

O módulo CC3000 foi programado para aguardar até que a rede pré-determinada, "ArduinoVGA", esteja na área de alcance, dessa maneira o CC3000 se conecta nela e fica aguardando os pacotes. Ao receber um pacote ele verifica qual linha e o número do fragmento, e o armazena no vetor de bytes na posição correta.

6. Conclusão

Nesse trabalho foi mostrado que é possivel implementar um dispositivo de baixo custo que permite transferir imagens de um dispositivo móvel Android para o Arduino através de uma rede Wi-Fi, sendo o Arduino utilizado para efetuar a comunicação através do padrão SVGA com o projetor multimídia, aproveitando principalmente projetores mais antigos. Porém, o objetivo inicial de enviar uma imagem de 800×600 pixels não foi atingindo completamente por conta de limitações de hardware. Contudo, as limitações foram contornadas adotando uma imagem de tamanho menor e em escala de cinza. Para se atingir o objetivo proposto inicalmente precisaríamos expandir a memória do Arduino, pois ela se mostrou insuficiente para guardar uma imagem com 800×600 pixels.

Referências

[1] J. C. Antonio, "Uso pedagógico de apresentações de slides digitais," *SBO*, 2010. Acesso em: 30/06/2015.

[2] O. U. Press, "Oxford dictionaries." Disponível em: http://www.oxforddictionaries.com/us/definition/american_english/smartphone. Acesso em: 10/07/2015.

[3] P. Deitel, H. Deitel, A. Deitel, and M. Morgano, *Android para programadores: Uma Abordagem Baseada em Aplicativos, 1st Edition*, ch. 1, p. 4. bookman, 2012.

[4] Android, "Android." Disponível em: http://www.android.com/history/. Acesso em: 15/06/2015.

[5] Android, "Android source." Disponível em: http://source.android.com/source/index.html. Acesso em: 15/06/2015.

[6] Android, "Android developer." Disponível em: http://developer.android.com/about/index.html. Acesso em: 15/06/2015.

[7] Gartner, "Gartner says worldwide smartphone sales recorded slowest growth rate since 2013." Disponível em: http://www.gartner.com/newsroom/id/3115517. Acesso em: 10/09/2015.

[8] Arduino, "Arduino." Disponível em: http://www.arduino.cc/en/Guide/Introduction. Acesso: 15/06/2015.

[9] Arduino, "Arduino due." Disponível em: https://www.arduino.cc/en/Main/ArduinoBoardDue. Acesso em: 07/02/2016.

[10] Atmel, "Microcontroller atsam3x cortex-m3 mcu." Disponível em: http://www.atmel.com/devices/ATsam3x8e.aspx. Acesso em: 10/02/2016.

[11] "Texas instruments - cc3000." Disponível em: http://www.ti.com/product/cc3000. Acesso em: 15/06/2016.

[12] "Texas instruments - cc3000." Disponível em: http://processors.wiki.ti.com/index.php/CC3000. Acesso em: 15/06/2016.

[13] "Texas instruments - cc3000." Disponível em: http://www.ti.com/lit/ds/symlink/cc3000.pdf. Acesso em: 15/06/2016.

[14] L. Ammeraal and K. Zhang, *Computação Gráfica para Programadores Java, 2ª Edição*, ch. 1, p. 3. LTC, 2008.

[15] J. da Fonseca, *Tipografia & Design gráfico: Design e Produção de Impressos e Livros*, ch. 11, pp. 148–149. bookman, 2008.

[16] W. Berry, "Vga controller card," Sept. 22 1992. US Patent 5,150,109.

[17] N. Gammon, "Arduino uno output to vga monitor." Disponível em: http://www.gammon.com.au/forum/?id=11608. Acesso em: 10/02/2016.

[18] I. Skliarova, "Desenvolvimento de circuitos reconfiguráveis que interagem com um monitor vga." Disponível em: http://revistas.ua.pt/index.php/revdeti/article/view/2071/1943. Acesso em: 10/02/2016.

[19] TinyVGA, "Svga signal 800 x 600 @ 60 hz timing." Disponível em: http://tinyvga.com/vga-timing/800x600@60Hz. Acesso em: 02/02/2016.

[20] F. Hechler, "Android pdf viewer library." Disponível em: http://andpdf.sourceforge.net/. Acesso em: 07/10/2015.

[21] Atmel, "Datasheet microcontroller sam3x." Disponível em: http://www.atmel.com/Images/atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf. Acesso em: 10/02/2016.

[22] Atmel, "Pulse width modulation generation using the at91 timer/counter." Disponível em: http://www.atmel.com/Images/doc2682.pdf. Acesso em: 10/02/2016.

[23] Atmel, "Dma controller (dmac) driver." Disponível em: http://www.atmel.com/Images/Atmel-42291-SAM3A-3U-3X-4E-DMA-Controller-DMAC_ApplicationNote_AT07892.pdf. Acesso em: 10/02/2016.

[24] R. Livre, "Dma acesso direto a memória." Disponível em: http://robolivre.org/conteudo/dma-acesso-direto-a-memoria. Acesso em: 10/02/2016.