UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE ENGENHARIA DE COMPUTAÇÃO

LEONARDO CÍCERO MARCIANO

ARQUITETURA DE TELAS TFT-LCD COM MICROCONTROLADORES E INTERFACES GRÁFICAS EM SISTEMAS EMBARCADOS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2019

LEONARDO CÍCERO MARCIANO

ARQUITETURA DE TELAS TFT-LCD COM MICROCONTROLADORES E INTERFACES GRÁFICAS EM SISTEMAS EMBARCADOS

Trabalho de Conclusão de Curso, apresentado à disciplina de Trabalho de Conclusão de Curso, do Curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná - UTFPR, *Campus* Pato Branco, como requisito parcial para obtenção do título de Engenheiro da Computação.

Orientador: Prof. Dr. Gustavo Weber Denardin

PATO BRANCO

2019

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Pato Branco
Departamento Acadêmico de Informática
Curso de Engenharia de Computação

TERMO DE APROVAÇÃO

Às 8 horas e 20 minutos do dia 12 de julho de 2019, na sala V006, da Universidade Tecnológica Federal do Paraná, Campus Pato Branco, reuniu-se a banca examinadora composta pelos professores Gustavo Weber Denardin (orientador), Diogo Ribeiro Vargas e Giovanni Alfredo Guarneri para avaliar o trabalho de conclusão de curso com o título **Arquitetura de telas TFT-LCD com microcontroladores e interfaces gráficas em sistemas embarcados**, do aluno **Leonardo Cícero Marciano**, matrícula 01494295, do curso de Engenharia de Computação. Após a apresentação o aluno foi arguido pela banca examinadora. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Prof. Gustavo Weber Denardin
Orientador (UTFPR)

_____
Prof. Diogo Ribeiro Vargas
(UTFPR)

_____
Prof. Giovanni Alfredo Guarneri
(UTFPR)

_____
Profa. Beatriz Terezinha Borsoi
Coordenador de TCC

_____
Prof. Pablo Gauterio Cavalcanti
Coordenador do Curso de
Engenharia de Computação

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

O aumento da adoção da utilização da tecnologia de telas de cristal líquido de transistores de filme fino (TFT-LCD) em sistemas embarcados permitiu a existência de aplicações gráficas mais complexas e interativas, porém trouxe vários desafios quanto a utilização de recursos de *hardware* e *software*. Várias bibliotecas gráficas surgiram no mercado com o intuito de facilitar a adoção dessas aplicações gráficas, o que se transformou em um nicho que se encontra em crescimento contínuo. Inicialmente apresentaram alto custo, todavia mais recentemente até mesmo soluções gratuitas podem ser encontradas. Mesmo assim, ainda pode-se avaliar a necessidade de criar uma solução própria ante as disponíveis. Este trabalho propõe explorar essa abordagem, primeiramente ao apresentar a arquitetura utilizada para a interface de telas com microcontroladores que utilizam a tecnologia de TFT-LCD, revisar o estado da arte em relação ao uso atual e realizar comparações com arquiteturas mais simples. Com isso, pode-se implementar de forma básica um *driver* e biblioteca para sistemas embarcados, com objetivo de comparar essa implementação com algumas soluções disponíveis atualmente em termos de desempenho, consumo de recursos computacionais e facilidade de uso.

Palavras-chave: sistemas embarcados, interface gráfica de usuário, telas de cristal líquido, aplicações gráficas.

ABSTRACT

The increase in adoption in thin-film transistor liquid crystal displays (TFT-LCD) in embedded systems allowed more complex and interactive graphical applications to exist, but it has also brought several challenges concerning hardware and software resources. Several graphics libraries have appeared in the market to facilitate the creation of such complex graphic applications, transforming into an area that is continually growing. The first graphic libraries were commercial high-cost solutions. However, free solutions have emerged in the last few years. The need to create a proprietary graphics library despite of such free and commercial libraries can also be evaluated. This work proposes exploring such approach, by firstly presenting the architecture utilized to interface TFT-LCD with microcontrollers, by revising the state of art in relation with the current usage of these displays and making comparisons with simpler architectures. Thus, a basic driver and library for embedded systems can be developed to compare this implementation with some of the currently available solutions, in terms of performance, usage of computational resources and ease of use.

Keywords: embedded systems, graphical user interfaces, liquid crystal displays, graphical applications.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

a-Si	Silício Amorfo
ADC	*Analog-Digital Converter*
AMOLED	*Active-Matrix Organic Light-Emitting Diode*
BPP	*Bits* Por *Pixel*
CCFL	*Cold Cathode Fluorescent Lamp*
CGRAM	*Character Generator Random Access Memory*
CLUT	*Color Look-Up Table*
CRT	*Cathode Ray Tube*
DAC	*Digital to Analog Converter*
DDRAM	*Display Data Random Access Memory*
DE	*Data Enable*
DMA	*Direct Memory Acess*
DMA2D	*Two-Dimensional Direct Memory Access*
FIFO	*First In, First Out*
GDRAM	*Graphic Display Random Access Memory*
GPIO	*General Purpose Input/Output*
GRAM	*Graphic Random Access Memory*
GUI	*Graphical User Interface*
HBP	*Horizontal Back Porch*
HCLK	*High-speed Clock*
HFP	*Horizontal Front Porch*
LCD	*Liquid Crystal Display*
LED	*Light Emitting Diode*
LFSR	*Linear Feedback Shift Register*
LTDC	*LCD-TFT Display Controller*
MIPI	*Mobile Industry Processor Interface*
OLED	*Organic Led Emitting Diode*
p-Si	Silício Policristalino
PCLK	*Peripheral Clock*
PFC	*Pixel Format Conversion*

QVGA	*Quarter Video Graphics Array*
RGB	*Red Green Blue*
RTOS	*Real Time Operating System*
SAW	*Surface Acoustic Wave*
SPI	*Serial Peripheral Interface*
STN	*Super Twisted Nematic*
TFT	*Thin Film Transistor*
VBP	*Vertical Back Porch*
VFP	*Vertical Front Porch*
x-Si	Silício Policristalino
XGA	*Extended Graphics Array*

SUMÁRIO

1 INTRODUÇÃO

Disponível há cerca de meio século, a tecnologia de telas de cristal líquido (*Liquid Crystal Display* - LCD) inicialmente não teve grande aceitação devido a problemas de desempenho: ângulos de visão estreitos, alto tempo de resposta e painéis de tamanho insuficiente. Essa situação teve uma mudança significativa com o advento dos LCDs de matriz ativa, com atenção maior à tecnologia de transistores de filme fino (*Thin Film Transistor* - TFT) de silício amorfo (KUO, 2013).

A primeira tela de TFT-LCD foi comercialmente produzida em 1989. Contudo, a aplicação para essa tecnologia estava restrita a dispositivos que não podiam acomodar uma tela de tubo de raios catódicos (*Cathode Ray Tube* - CRT), como computadores portáteis e câmeras digitais.

Kuo (2013) aponta que a indústria passou a utilizar a tecnologia TFT-LCD para outros produtos comerciais devido ao grande sucesso da introdução dessa tecnologia no segmento de *laptops*. Assim, a gama de aplicações para o LCD aumentou significativamente a partir dos anos 2000, com seu uso expandido para televisores e monitores. O próximo passo foi a aplicação da tecnologia LCD a dispositivos menores, como telefones celulares (OHSHIMA, 2014).

Da mesma forma, a capacidade de processamento disponível para tais dispositivos também aumentou. Assim, ao combinar a evolução da tela de TFT-LCD com a dos processadores, tornou-se possível manipular e mostrar itens mais complexos em telas cada vez menores.

Apesar da adoção em massa das telas TFT-LCD em diversos mercados, seu uso ainda era limitado em sistemas embarcados baseados em microcontroladores. Nesse tipo de sistema é comum a existência de restrições de consumo de energia, de recursos computacionais e de custo. Devido a isso, a visualização de informação é comumente realizada por mostradores numéricos, como os de sete segmentos, ou alfanuméricos, com mais segmentos, limitando o tipo de informação apesentada.

Felizmente, os sistemas embarcados também evoluíram, permitindo, assim, utilizar telas de matriz de pontos (*dot matrix*), mesmo que monocromáticas. Mais recentemente, foi possível empregar telas gráficas de alta resolução que, apesar de ainda possuírem controladores, como no caso da tela de matriz de pontos, apresentam arquitetura e conexão diferenciada, visto a complexidade inerente das imagens a serem manipuladas. Além disso, o uso de telas gráficas e a interação por meio de toque na superfície da tela trazem vários desafios tanto em termos de *hardware* como de *software* (DEVNATH, 2014).

Logo, com o intuito de facilitar a adoção de aplicações gráficas mais complexas em sistemas embarcados, surgiram diversas bibliotecas gráficas no mercado. Rapidamente tais bibliotecas assumiram um nicho de mercado em crescimento, apresentando alto custo. Exemplos disso são a biblioteca *Embedded Wizard* da TARA Systems GmbH (2017), em que a licença de iniciante tem um custo de € 2.990, com limitação comercial de 500 unidades por ano. Outra solução é a *TouchGFX* da Draupner ... (2017), em que uma licença comercial para até 3.000 unidades por ano tem um custo de € 5.000. No entanto, com a disseminação da tecnologia TFT-LCD e o apoio de diversos fabricantes de microcontroladores, houve um crescimento considerável nas opções disponíveis de bibliotecas gráficas, surgindo inclusive soluções gratuitas caso sua utilização se dê por meio de um microcontrolador de um fabricante específico. Um exemplo vem da STMicroelectronics (2014), em parceria com a *Segger Microcontroler GmbH & Co. KG*, permitiu fornecer a biblioteca *STemWin* – baseada na solução *emWin* da *Segger* – para a linha de microcontroladores STM32. Devido a essa diversidade de opções, surge a necessidade de se avaliar questões como facilidade de uso, curva de aprendizado, ocupação de recursos computacionais, entre outros.

Apesar da diversidade disponível no mercado, pode-se também avaliar a necessidade de criação de uma nova biblioteca gráfica. Essa abordagem é a que será utilizada neste trabalho, mostrando não somente de que maneira é concebida esse tipo de biblioteca, mas também como é criado um *driver* que realiza o interfaceamento com as telas de TFT-LCD. Com isso, também pretende-se efetuar uma análise das vantagens da utilização de soluções disponíveis frente ao desenvolvimento de uma solução própria.

1.1 OBJETIVOS

Estão definidos a seguir os objetivos gerais e específicos que se pretendem alcançar com o desenvolvimento deste trabalho.

1.1.1 OBJETIVO GERAL

Apresentar a arquitetura utilizada para interface de telas de TFT-LCD com microcontroladores, assim como conceber *drivers* para interação com essas telas e descrever como esses *drivers* podem ser utilizados para criar uma interface gráfica para sistemas embarcados.

1.1.2 OBJETIVOS ESPECÍFICOS

- Revisar o estado da arte em relação ao atual uso da tecnologia TFT-LCD em microcontroladores;
- Comparar analiticamente as arquiteturas de LCD alfanuméricas de matriz de pontos com as arquiteturas de telas TFT-LCD, mostrando o aumento de complexidade envolvido ao migrar para uma plataforma que faz o uso de tal tecnologia;
- Apresentar a arquitetura de conexão entre um microcontrolador e uma tela de TFT-LCD a partir do *kit* microcontrolado STM32F429I-DISCO;
- Realizar uma implementação básica de um *driver* para telas TFT-LCD e uma interface visual em uma tela TFT-LCD na plataforma STM32F429I-DISCO;
- Comparar algumas das bibliotecas gráficas disponíveis, como *Embedded Wizard* e *LittlevGL*, em termos de desempenho, consumo de recursos computacionais e facilidade de uso;
- Realizar testes de desempenho das bibliotecas a serem comparadas.

1.2 JUSTIFICATIVA

Dada a evolução das aplicações móveis, industriais e comerciais, interfaces gráficas de usuário (*Graphical User Interface* - GUI) têm seu uso cada vez mais requisitado (STMICROELECTRONICS, 2017a). Com isso, se faz necessário entender o funcionamento das bibliotecas que englobam a funcionalidade de interface com o usuário, visando um controle maior sobre a aplicação, seja em termos de como as informações são apresentadas ou até mesmo para introduzir melhoramentos nas ferramentas existentes.

Visto que a maioria das bibliotecas disponíveis são pagas, a análise da arquitetura pode se fazer necessária para facilitar a introdução de uma biblioteca gráfica nova, permitindo que seja orientada a uma utilidade mais específica. Isso também permite que o *hardware* envolvido seja utilizado de forma eficiente.

Como os microcontroladores têm como um dos seus objetivos o baixo consumo de energia, a utilização eficiente dos recursos de *hardware* disponíveis torna-se crucial ao fazer uso de tecnologia TFT-LCD. Shim (2006) aponta que a grande maioria da energia consumida em um sistema móvel embarcado vem da retroiluminação e memórias envolvidas para o funcionamento da tela.

1.3 ESTRUTURA DO TRABALHO

O trabalho se encontra estruturado em cinco capítulos. O primeiro capítulo contextualiza o assunto realizando uma abordagem histórica das telas de TFT-LCD até seu uso em um período mais recente. A partir disso, definem-se os objetivos a serem alcançados e a justificativa para a sua execução.

Em seguida, o Capítulo 2 levanta um referencial teórico acerca do assunto principal do trabalho, iniciando pelos componentes que fazem parte de um sub-sistema gráfico, a integração entre os mesmos e a interface por trás da comunicação com a tela propriamente dita. Então, a arquitetura de conexão desses componentes é apresentada em detalhes, assim como o funcionamento de cada um deles. No final do capítulo, é realizada uma comparação entre um sub-sistema gráfico que consiste de uma tela de matriz de pontos e outro que usa uma tela de TFT-LCD colorida, com o intuito de mostrar o aumento de complexidade envolvido nessa migração.

O Capítulo 3 discute a metodologia de execução do trabalho e relata como ocorreu o desenvolvimento do *software* explicitado nos objetivos do trabalho, com o detalhamento da criação de cada componente da interface gráfica, o funcionamento da biblioteca desenvolvida e como que se utiliza ela.

O Capítulo 4 engloba os resultados obtidos, iniciando com uma comparação entre duas soluções disponíveis no mercado em termos de preço, disponibilidade para plataformas embarcadas e utilização. Então, entra-se na comparação de desempenho, em uso de memória e processamento, o que inclui a biblioteca desenvolvida no Capítulo 3.

O trabalho se encerra com o Capítulo 5, em que são explicitadas as conclusões oriundas dos resultados obtidos.

2 REFERENCIAL TEÓRICO

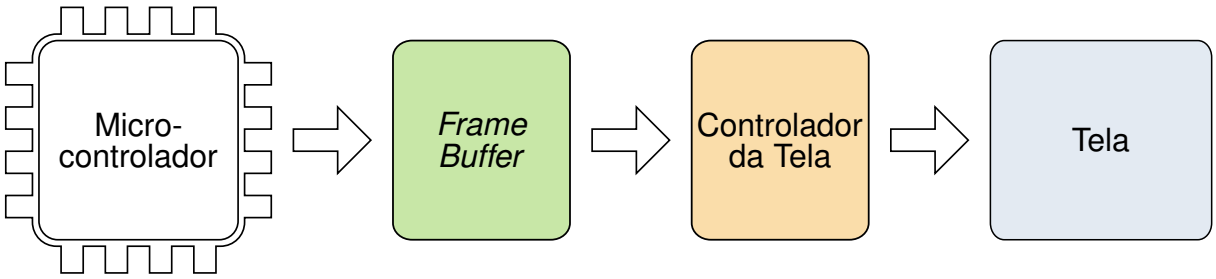
O referencial teórico consiste de três seções principais. Inicialmente, são apresentadas a definição de componentes e os conceitos básicos que fazem parte de um sub-sistema gráfico encontrado em sistemas microcontrolados e embarcados amplamente utilizados. Além disso, são mostradas as diferentes maneiras de integrar esses componentes e a principal interface envolvida na exibição física de uma imagem.

A segunda seção consiste na apresentação da arquitetura de conexão entre os componentes do sub-sistema gráfico. Por fim, é realizada uma comparação que mostra o aumento de complexidade envolvida no processo de migração de telas de matriz de pontos para uma tela de tecnologia TFT-LCD.

2.1 COMPONENTES

O sub-sistema gráfico de um sistema microcontrolado que faz uso de uma tela de TFT-LCD necessita de pelo menos quatro componentes básicos, que são o microcontrolador, o *frame buffer*, o controlador do *display* e a tela propriamente dita (MICROCHIP ..., 2011). A Figura 1 ilustra a integração básica entre essas partes.

Figura 1 – Os quatro componentes básicos de um sub-sistema gráfico embarcado



Fonte: adaptado de *Microchip Technology Inc.* (2011, p. 4).

2.1.1 MICROCONTROLADOR

O microcontrolador é o dispositivo responsável por computar a imagem a ser copiada para o *frame buffer* e realizar a montagem de gráficos primitivos como ícones ou imagens (STMICROELECTRONICS, 2017a). Se faz necessário também que haja capacidade de processamento suficiente para realizar esses cálculos. Além disso, o microcontrolador e o controlador do *display* devem possuir as mesmas configurações em relação à profundidade de cores e a área de memória do *frame buffer* a ser usada (MICROCHIP ..., 2011). Em sua memória também são armazenados todos os gráficos

primitivos que serão utilizados pela aplicação, preferencialmente em memória não-volátil, visto que ela é normalmente encontrada em maior quantidade em um sistema microcontrolado.

2.1.2 *FRAME BUFFER*

O *frame buffer* consiste de uma área de memória, normalmente volátil, que guarda as informações (*pixels*) que são mostradas na tela. Normalmente esta região de memória contém os dados referentes a um quadro que é constantemente atualizado e mandado à tela, podendo estar presente dentro da memória RAM ou em um banco de memória separado. Também pode ser chamada de RAM gráfica (GRAM). O tamanho mínimo necessário para essa área de memória depende da resolução da tela e da profundidade de cores a ser exibida, devendo ser o suficiente para exibir um quadro completo (MICROCHIP ..., 2011).

STMicroelectronics (2017) e *Microchip Technology Inc.* (2011) expõem a Equação (1) para calcular o tamanho mínimo do *frame buffer*, em que F é o tamanho do *frame buffer* em *bits*, P é a quantidade de *pixels* e B é a profundidade de cores em *bits* por *pixel* - BPP.

$$F = P \times B \tag{1}$$

Para obter-se o tamanho em *bytes*, basta dividir (1) por oito. Aplicando essa equação a uma tela QVGA, com 320 *pixels* de largura e 240 de altura, tem-se que o tamanho mínimo do *frame buffer* necessário, utilizando 16 *bits* para definir cada *pixel*, é de 150 kB. Assim, tem-se que:

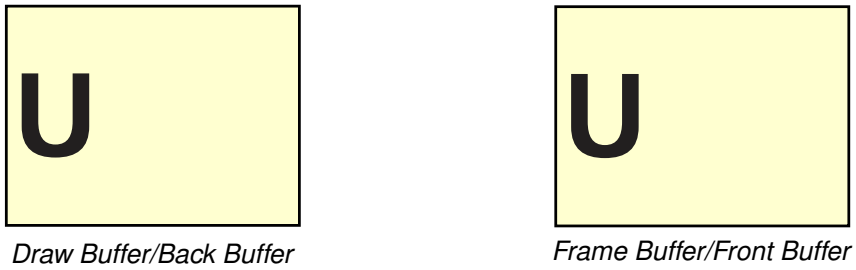
$$F = 320 \times 240 \times 16 = 1.228.800 \textit{ bits}, 153.600 \textit{ bytes} \text{ ou } 150 \text{ kB}$$

Com o conteúdo do *frame buffer* exibido na tela a uma taxa de 60 Hz, é comum que haja uma atualização da tela enquanto escreve-se dados no *frame buffer*, fazendo com que as modificações parciais acabem sendo exibidas ao usuário. Esse comportamento é mais notável em tarefas que são processadas mais lentamente, como a decodificação de uma imagem ou a presença de vários *widgets* – que são elementos da interface de usuário, como botões, janelas e listas – na tela. Para resolver esse problema, um método empregado é a utilização de mais de um *frame buffer* (MICROCHIP ..., 2011). O método de utilização de mais de um *frame buffer* faz uso de um *buffer* frontal, denominado *front buffer*, que exibe a imagem pronta, e um ou mais *buffers* traseiros, para realizar as operações de desenho. Isso permite que a tela

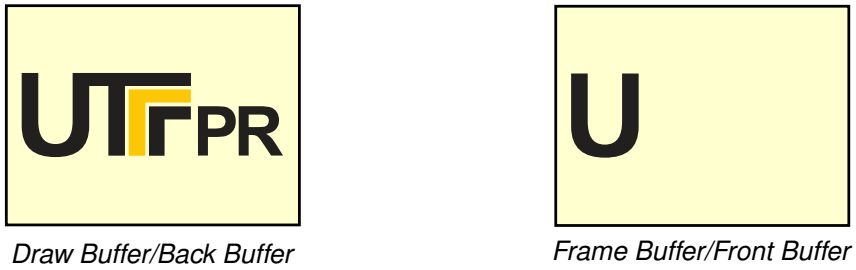
exiba quadros completos, mesmo que operações de desenho estejam em andamento (SEGGER ..., 2015).

Na Figura 2 observa-se em detalhe um dos casos de utilização de múltiplos *frame buffers*, fazendo uso de um *buffer* frontal e um traseiro, denominado *double buffering*. Neste exemplo, o *front buffer* e *back buffer* são denominados *frame buffer* e *draw buffer*, respectivamente.

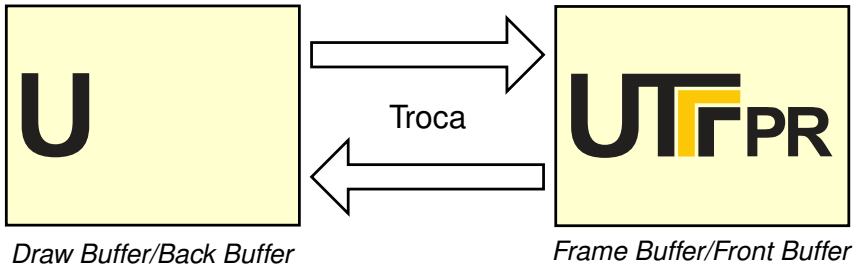
Figura 2 – Detalhamento do processo de *double buffering*



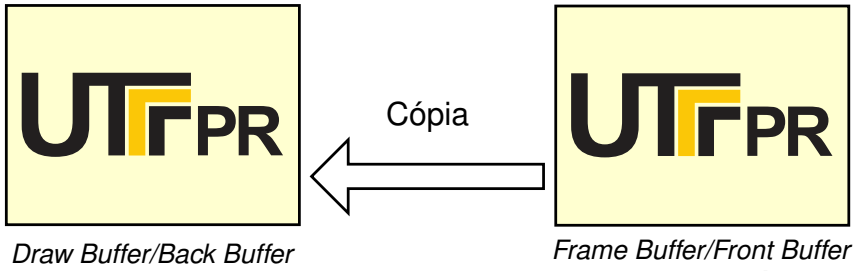
(a) Primeira etapa, em que o *frame buffer* e o *draw buffer* possuem o mesmo conteúdo



(b) Segunda etapa, em que o *draw buffer* é atualizado com as informações do próximo quadro



(c) Terceira etapa, em que os *frame buffers* são trocados e a nova imagem é exibida ao usuário



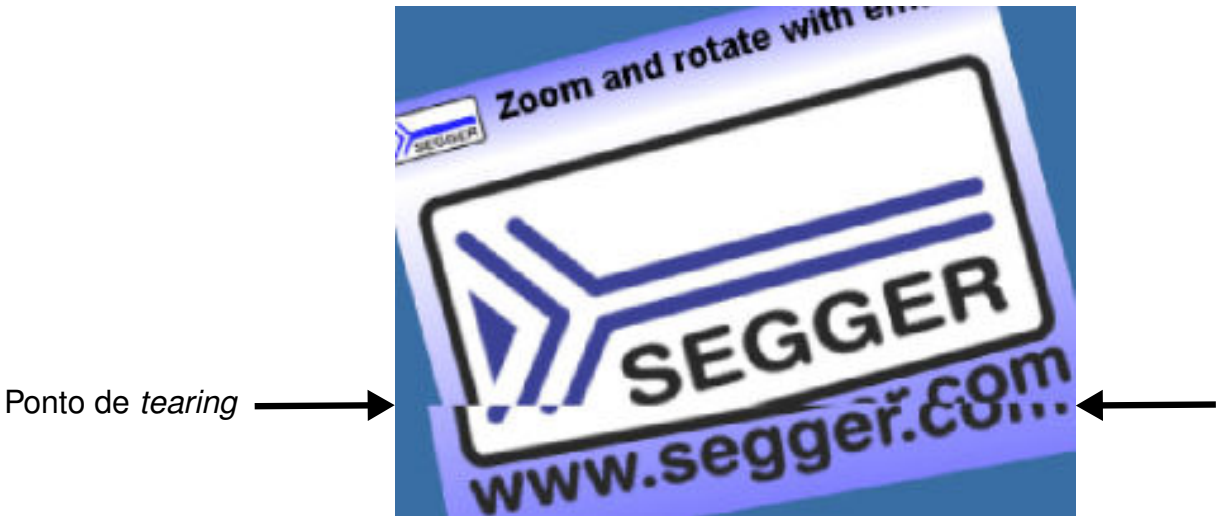
(d) Quarta etapa, em que o *draw buffer* recebe o conteúdo do *frame buffer* e o processo recomeça

Fonte: adaptado de *Microchip Technology Inc.* (2011, p. 30).

Outra variante é o *triple buffering*, em que três *frame buffers* são utilizados ao invés de dois, de modo que um deles é frontal e dois são traseiros. Sua operação ocorre pelo desenho do primeiro *frame buffer* de trás, o qual se torna o frontal após a operação, tornando-se visível na tela. Caso exista a necessidade de realizar mais operações de desenho antes do primeiro *frame buffer* traseiro se tornar visível, essas operações são realizadas no segundo *frame buffer* traseiro, deixando o primeiro pendente de ser exibido. A vantagem sobre o *double buffering* é a possibilidade de continuar realizando operações de desenho mesmo após o término do quadro que será o novo *front buffer*, pois com dois *frame buffers* ou a troca é feita imediatamente, o que causa o efeito indesejado de *tearing* – problema apontado por *STMicroelectronics* (2017) devido a falta de sincronização com a tela – ou esperar o próximo sinal de sincronização vertical sem poder realizar operações de desenho no outro *frame buffer*, diminuindo o desempenho do sistema (SEGGER ..., 2015).

O efeito *tearing* é descrito por *Silicon Laboratories Inc.* (2016), que ocorre quando há uma troca repentina de quadro enquanto o controlador está enviando os dados do quadro atual para a tela. Visualmente falando, a parte superior da tela mostra o quadro antigo e a parte inferior da tela exibe o quadro novo. Isso causa um efeito de corte na imagem, aparentando ao usuário como se houvessem linhas que passam através do quadro exibido na tela.

Figura 3 – Ilustração do efeito de *tearing*



Fonte: adaptado de *SEGGER Microcontroller GmbH & Co. KG* (2015, p. 1018).

Uma das consequências imediatas desse tipo de abordagem, além de ser uma das desvantagens, é a necessidade do dobro de memória, no caso do *double buffering*, ou o triplo para o *triple buffering*, o que não pode estar sempre disponível dependendo do sistema a ser utilizado (MICROCHIP ..., 2011).

2.1.3 CONTROLADOR DA TELA

A função principal do controlador do *display* é atualizar a tela constantemente, enviando o conteúdo do *frame buffer* repetidamente. Antes do envio propriamente dito, pode ser necessária a decodificação dos dados para o formato de cor apropriado. Além dos dados, o controlador necessita também enviar os sinais de controle adequado para a tela, que tem seus requisitos de temporização a serem respeitados (MICROCHIP ..., 2011).

O controle da tela pode ser feito tanto por um circuito integrado dedicado a função, como o controlador ILI9341, que é embutido ao módulo da tela, quanto a um que se encontra dentro do microcontrolador. Um exemplo deste último é denominado *LCD-TFT Display Controller* (LTDC), presente em microcontroladores da *STMicroelectronics*.

Esses controladores possuem o intuito de retirar a carga de processamento do microcontrolador, diminuindo a interação necessária com a tela diretamente por parte da CPU (STMICROELECTRONICS, 2017a). Também é possível utilizar ambos os controladores em conjunto com este intuito.

2.1.4 TELA

A tela é o dispositivo que mostra uma sequência de cores em uma matriz de *pixels* e converte uma representação digital de cor para cores reais, visíveis pelo olho humano (MICROCHIP ..., 2011). A tecnologia de TFT-LCD utiliza uma tecnologia de matriz ativa composta por uma grelha de transistores de filme fino. O padrão da indústria para esse tipo de LCD produzido em massa são os TFTs de silício amorfo (a-Si). Outras tecnologias disponíveis são a de silício monocristalino (x-Si) e silício policristalino (p-Si) (FUJITSU ..., 2006).

Uma tela pode ser caracterizada pelas seguintes propriedades (STMICROELECTRONICS, 2017a; MICROCHIP ..., 2011; FREESCALE ..., 2015):

- **Resolução:** é o número de *pixels* que a tela exibe, representado pela quantidade horizontal multiplicada pela vertical;
- **Taxa de Quadros:** é a taxa de atualização do conteúdo visto no *display* em hertz. Deve acontecer em pelo menos 60 Hz, pois taxas menores podem ocasionar efeitos visuais desagradáveis;
- **Profundidade de Cor:** é a quantidade de *bits* usado para representar a cor de um único *pixel* em uma memória. É mensurada em *bits* por *pixel* (BPP).

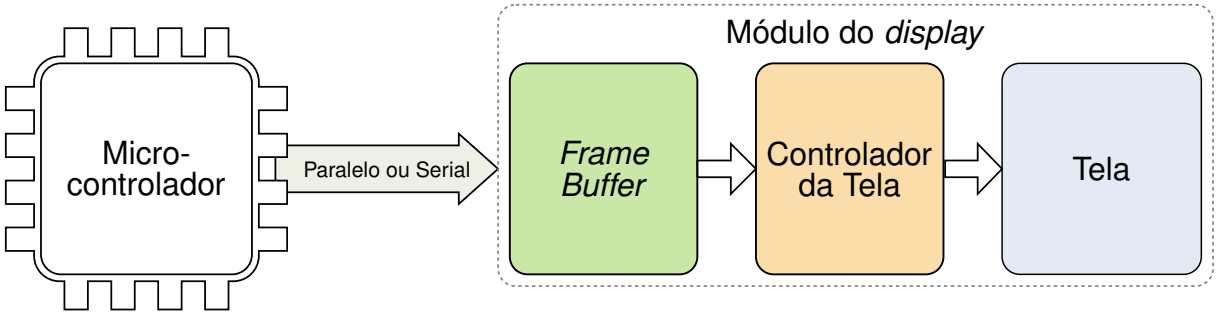
- **Retroiluminação:** no caso de telas de tecnologia TFT ou Nemática Super Trançada (*Super Twisted Nematic* - STN), é necessário uma iluminação adicional atrás do painel para poder visualizá-lo. Pode ser feita utilizando LED ou Lâmpada Fluorescente de Cátodo Frio (*Cold Cathode Fluorescent Lamp* - CCFL);
- **Contraste:** a proporção de intensidade entre as cores branca e preta na tela. Quanto maior o contraste, melhor a qualidade do *display*;
- **Ângulo de Visão:** ângulo vertical ou horizontal no qual a tela pode ser devidamente vista, em graus.

Um exemplo é o módulo LCD SF-TC240T-9370A-T, que possui uma tela de resolução QVGA. Embutido nela, há o controlador ILI9341 e um painel de toque, para prover interações por meio de toque na tela. Possui luz de retroiluminação de LED da cor branca e suporte para até 262.144 cores, ou 18 *bits* de profundidade de cor. Seu ângulo de visualização é de 45 graus na horizontal em ambos os lados, 35 graus na vertical no sentido positivo e 15 no sentido negativo (SAEF ..., 2012).

2.2 INTEGRAÇÃO DOS COMPONENTES

Como visto na Seção 2.1, um sub-sistema gráfico consiste de quatro componentes principais. Além disso, pode-se escolher como realizar a integração dos mesmos, dependendo de fatores como desempenho, custo e espaço. A escolha depende da aplicação em questão, existindo quatro combinações distintas para a realização desse passo. É comum em todas as combinações que mais de um componente se encontre em mais de um circuito integrado, o que diminui a quantidade de dispositivos presentes na integração. A primeira combinação é vista na Figura 4.

Figura 4 – Integração entre os componentes de um sub-sistema gráfico com uso de um microcontrolador e um módulo do *display*

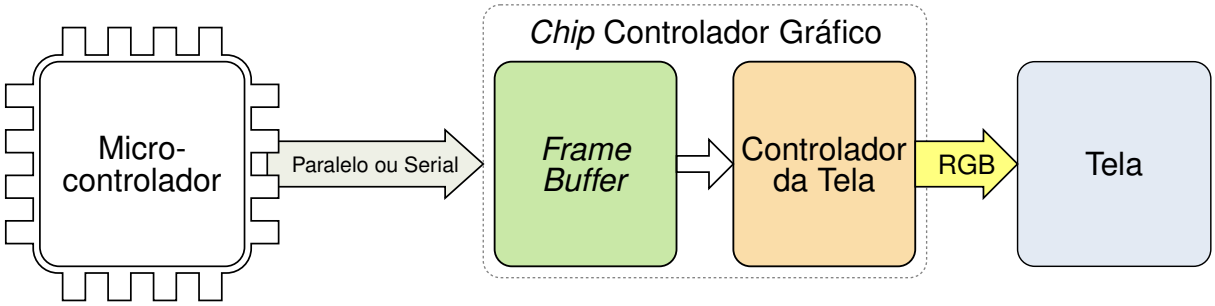


Fonte: adaptado de *Microchip Technology Inc.* (2011, p. 7).

Nesse arranjo, o *frame buffer*, o controlador da tela e a tela se encontram dentro de um mesmo dispositivo, normalmente chamado de módulo do *display*. Assim, tem-se dois dispositivos, sendo que esses podem se comunicar por meio de interfaces paralelas ou seriais (MICROCHIP ..., 2011).

Os benefícios dessa associação envolvem a economia de espaço por conta do uso de menos dispositivos. Além disso, não é necessário um circuito integrado separado para a realização de funcionalidade gráfica. Porém, esse tipo de opção tem um custo mais elevado. Caso seja necessário utilizar uma solução diferente, pode ser que uma mudança no *driver* de *software* também ocorra. Por fim, como os componentes são integrados em um único módulo, é possível que falte memória para recursos mais sofisticados (MICROCHIP ..., 2011). Na Figura 5 observa-se a próxima associação.

Figura 5 – Integração entre componentes de um sub-sistema gráfico utilizando um microcontrolador, um *chip* controlador gráfico e a tela



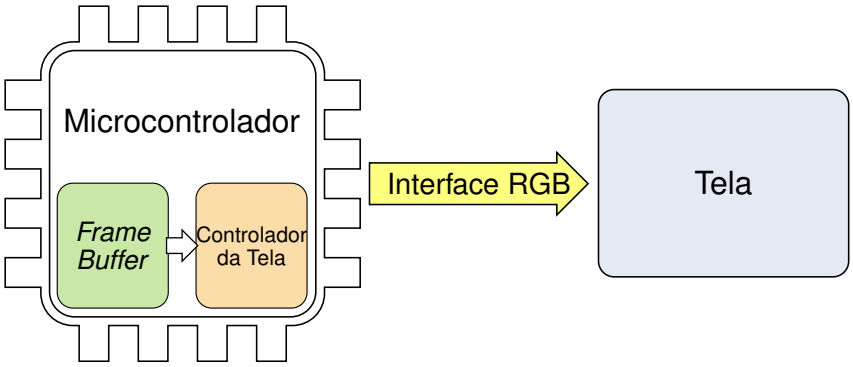
Fonte: adaptado de *Microchip Technology Inc.* (2011, p. 7).

Com três dispositivos, essa combinação separa o *frame buffer* e o controlador da tela em um único CI, chamado de *chip* controlador gráfico. O microcontrolador se comunica com esse *chip* utilizando interface paralela ou serial, tal qual na configuração anterior, deixando a comunicação com a tela para esse controlador, por meio do uso da interface RGB. A interface RGB é paralela e provê os dados de um *pixel* e os sinais de temporização necessários (MICROCHIP ..., 2011).

Devido ao fato da tela não conter qualquer circuito de controle, é possível trocá-la separadamente sem realizar ajustes no código do *driver* de *software*. Ademais, essa opção costuma possuir um custo menor do que a configuração com controlador e *frame buffer* embutidos. Porém, o aumento do número de dispositivos pode acarretar em um incremento na área de placa de circuito impresso a ser utilizada. Por fim, como o tamanho de memória para o *frame buffer* é fixo nesse CI, o maior tamanho

da tela que pode ser usado acaba sendo limitado pelo *frame buffer* (MICROCHIP ..., 2011). A terceira combinação é explicitada na Figura 6.

Figura 6 – Integração entre componentes de um sub-sistema gráfico utilizando um microcontrolador e a tela



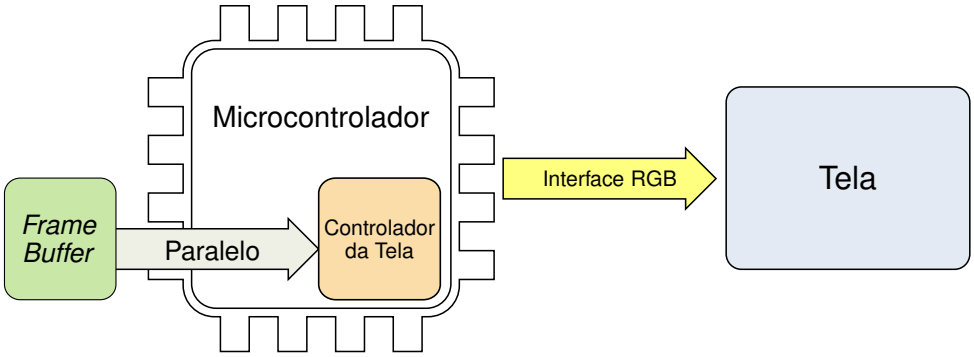
Fonte: adaptado de *Microchip Technology Inc.* (2011, p. 7).

Nessa configuração, o microcontrolador é responsável pelo controle da tela e pelo *frame buffer*. Normalmente, essa é a opção mais compacta e barata que existe, visto que apenas o CI do microcontrolador é necessário para o funcionamento da tela, com a comunicação diretamente ocorrendo por meio da interface RGB. Visto que a área de memória para o *frame buffer* se encontra junto com a memória de trabalho do dispositivo, a composição de imagens pode se tornar mais rápida. Além disso, se for necessária a troca da tela, não será obrigatória a troca do código do *software* de *driver*, assim como na associação da Figura 5 (MICROCHIP ..., 2011).

Porém, também como naquela associação, pode haver uma limitação ao tamanho da tela devido a memória destinada ao *frame buffer*, que pode ser escassa (MICROCHIP ..., 2011). Além disso, atribui-se uma carga de processamento consideravelmente alta ao microcontrolador por fazê-lo controlar a tela, adicionado ao fato de necessitar também gerar a imagem que será enviada. Por último, tem-se o arranjo da Figura 7.

Similar ao da Figura 6, a única diferença envolvida é que a memória do *frame buffer* é colocada em um dispositivo separado. O microcontrolador continua utilizando a interface RGB para realizar o controle da tela e a comunicação com a memória do *frame buffer* pode ser feita utilizando outro protocolo e até mesmo o recurso de acesso direto à memória (*Direct Memory Acess* - DMA). Isso permite uma maior liberdade a escolha da tela a ser usada, visto que pode-se escolher uma memória apropriada para ela. Todavia, isso representa um CI a mais a ser utilizado, o que aumenta o custo e espaço físico necessário para a solução (MICROCHIP ..., 2011).

Figura 7 – Integração entre componentes de um sub-sistema gráfico utilizando um microcontrolador, a tela e um *frame buffer* externo



Fonte: adaptado de *Microchip Technology Inc.* (2011, p. 7).

2.3 PADRÕES DE INTERFACE PARA A TELA

A padronização das interfaces para telas é atualmente desenvolvida e mantida pela aliança MIPI (*Mobile Industry Processor Interface*), uma organização colaborativa global que objetiva definir e promover especificações de interface para dispositivos móveis (STMICROELECTRONICS, 2017a). Nesta seção será discorrido sobre três interfaces definidas pela MIPI.

A primeira é a MIPI-DBI, acrônimo para *MIPI Display Bus Interface*. Foi um dos primeiros padrões a serem publicados pela aliança. Esse padrão é usado para a interface com uma tela que possui GRAM integrada, de modo que os dados de pixel são atualizados na memória interna a tela. Existem três tipos de interface definidos nesse padrão (STMICROELECTRONICS, 2017a):

- Tipo A: baseado no barramento do Motorola 6800;
- Tipo B: baseado no barramento do Intel 8080;
- Tipo C: baseado no protocolo SPI.

A seguir tem-se a *MIPI Display Parallel Interface*, denotada MIPI-DPI, padroniza a interface que envolve o uso de um controlador TFT, como na utilização de sinais RGB de 16 ou 24 *bits* em conjunto com os sinais de sincronização (Sincronização Vertical e Horizontal, *clock* e habilitação de dados). O DPI é utilizado para a interface com telas sem uso de *frame buffer*, sendo que os dados dos *pixels* devem ser transmitidos em tempo real. Isso faz com que o desempenho de tempo real seja bom, porém requer uma alta largura de banda por parte do microcontrolador (STMICROELECTRONICS, 2017a).

Por último, existe a MIPI-DSI, *MIPI Display Serial Interface*. Esse padrão tem a intenção de diminuir a quantidade de linhas para a interface de uma tela, por meio do uso de um *link* diferencial multi-linhas de alta largura de banda. Essa ligação física também utiliza um padrão da MIPI, o MIPI D-PHY. Essa interface funciona encapsulando sinais DBI ou DPI, transmitindo-os pelo D-PHY por um protocolo denonimado PPI (STMICROELECTRONICS, 2017a).

Nota-se que as interfaces usada para controle de telas TFT apresentadas aqui possuem um barramento de sinais denotado RGB, utilizado para transmissão de dados de *pixel*. Estas interfaces são utilizadas em conjunto com o barramento paralelo RGB para prover funções diferentes, em que normalmente o barramento RGB é utilizado para transferência de dados de imagens e o serial, para o controle do módulo da tela.

Um exemplo prático disto é o *kit* de desenvolvimento mencionado neste trabalho, o STM32F429I-DISCO que utiliza a interface serial periférica (*Serial Peripheral Interface* - SPI) para o controle do módulo da tela e o barramento RGB para a transferência dos dados de imagem. Outro exemplo é o *kit* de desenvolvimento STM32F769I-DISCO, que utiliza um conector com o padrão MIPI-DSI para controle.

Devido a isso, os sinais de controle e de dados dessa interface serão explanados em detalhes na próxima seção. Porém, é necessário ressaltar que essa não é a única interface existente, visto que há interfaces como a LVDS (*Low-Voltage Differential Signaling*) que também são utilizadas para telas de LCD.

2.4 INTERFACE RGB

A interface RGB é uma interface de comunicação paralela que envia dados de maneira síncrona para a tela. Com o intuito de diminuir custos e oferecer flexibilidade, os módulos LCD que dispõem desse tipo de interface vêm sem o *frame buffer*. Os sinais que compõem essa interface são os dados RGB, sincronização vertical e horizontal, habilitação de dados e *clock* do *pixel* (GU, 2016; FAIRCHILD ..., 2007; ILI ..., 2011).

2.4.1 DADOS RGB

O acrônimo RGB se refere as cores vermelha (*red*), verde (*green*) e azul (*blue*). O modelo de cores RGB é aditivo, no qual essas cores são adicionadas em várias porcentagens para reproduzir um grande conjunto de novas cores, pela sobreposição de luz dessas três cores. Como os dados são enviados de forma paralela, vários

bits são destinados a cada componente de cor, e cada *pixel* na imagem armazenada tem a soma dos *bits* atribuídos aos *sub-pixels* vermelho, verde e azul (FREESCALE ..., 2015).

O número de *bits* dos dados RGB são divididos entre as três cores primárias, representada em uma sigla do tipo RGB<$N_R N_G N_B$>, em que N_R é a quantidade de *bits* atribuída a cor vermelha, N_G a verde e N_B a azul. Logo, em uma profundidade de cor de 16 BPP, se o barramento for dividido em cinco *bits* para representar a intensidade da cor vermelha, seis de verde e cinco de azul, a sigla resultante será RGB565 (MICROCHIP ..., 2011).

Para a tela mencionada na Seção 2.1.4, com 18 BPP, a divisão resulta no padrão RGB666. A Figura 8 mostra como os *bits* das cores são colocadas no barramento de dados, em que D_n é o n-ésimo *bit* do barramento de dados, R_n o n-ésimo *bit* do subconjunto da cor vermelha, G_n da cor verde e B_n da azul.

Figura 8 – Distribuição de um *pixel* RGB666 no barramento de dados

Escrita na memória de *frame* de 18BPP	D_{17}	D_{16}	D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
	R_5	R_4	R_3	R_2	R_1	R_0	G_5	G_4	G_3	G_2	G_1	G_0	B_5	B_4	B_3	B_2	B_1	B_0

Fonte: adaptado de *ILI Technology Corp.* (2011, p. 45).

2.4.2 TEMPOS DE SINCRONIZAÇÃO

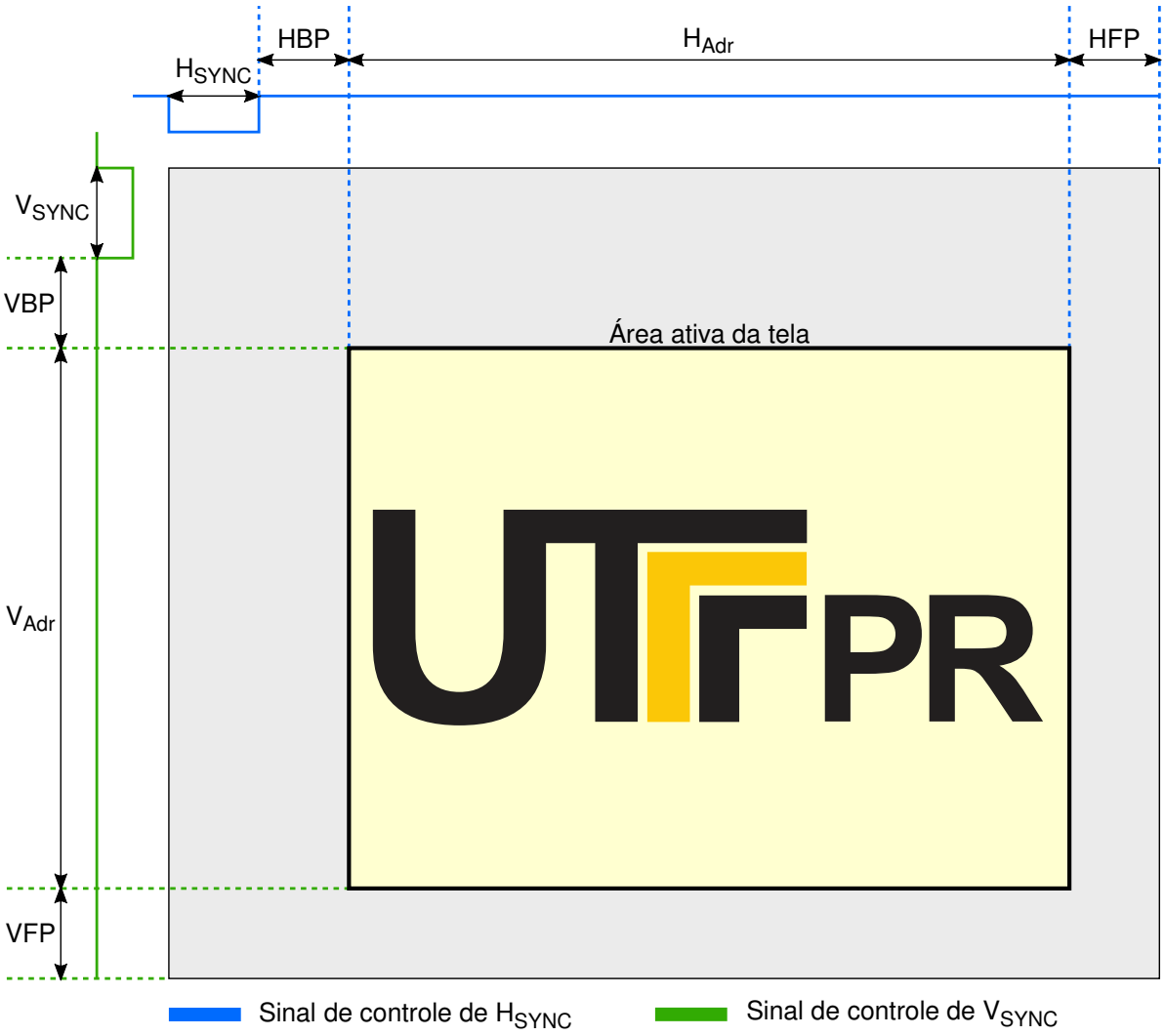
Para realizar a atualização de um quadro na tela, existem vários parâmetros de tempo necessários, devido a necessidade de tempo entre a mudança de conteúdo em linhas e quadros.

Em alguns desses intervalos não há *pixels* que são alterados. Esses tempos são denominados *Horizontal Back Porch* (HBP), *Horizontal Front Porch* (HFP), *Horizontal Address* (H_{Adr}), *Horizontal Synchronization* (H_{SYNC}), *Vertical Back Porch* (VBP), *Vertical Front Porch* (VFP), *Vertical Address* (V_{Adr}) e *Vertical Synchronization* (V_{SYNC}). A Figura 9 coloca de forma visual esses intervalos, considerando que a ordem de atualização começa no canto superior esquerdo, que a tela é varrida linha a linha até chegar no canto inferior direito e que a polarização dos sinais de sincronização ocorrem em nível baixo (ILI ..., 2011; SILICON ..., 2016).

Os sinais de sincronização horizontal (H_{SYNC}) e vertical (V_{SYNC}) têm o intuito de gerenciar o escaneamento das linhas da tela e do quadro inteiro, respectivamente. Quando uma linha inteira de *pixels* é recebida com sucesso, ocorre o pulso de V_{SYNC} antes do recebimento de novos dados para a próxima linha da tela. De maneira aná-

loga, quando a tela terminar de atualizar o quadro todo, ocorre o pulso de V_{SYNC} antes de proceder com os dados do próximo quadro (STMICROELECTRONICS, 2017a; ILI ..., 2011).

Figura 9 – Tempos de sincronização horizontal e vertical de atualização de um quadro



Fonte: adaptado de *STMicroelectronics* (2017, p. 24) e *ILI Technology Corp.* (2011, p. 46).

Os *back porches*, tanto horizontal como vertical, se referem aos tempos entre seus respectivos pulsos de sincronização e o começo de novos dados válidos. Já os *front porches* são os tempos entre o fim da entrada de dados válidos até o começo dos seus pulsos de sincronização. Assim, têm-se que HBP e HFP são intervalos que ocorrem antes e depois de cada linha, respectivamente, no passo que VBP e VFP se aplicam da mesma forma, porém para cada quadro (SILICON ..., 2016; SHARP CORPORATION, 2007).

Segundo *SHARP Corporation* (2007), esses termos se originam das definições de sinais de televisão analógica. A partir disso, vê-se em Gupta (2005) que o tempo de *front porch* tinha a função de assegurar que o pulso de sincronização horizontal começasse de um nível fixo, que o intervalo de *back porch* era para permitir a absorção de oscilações indesejadas e que os pulsos de sincronização eram usados para iniciar os circuitos de escaneamento e garantir a sincronia entre esses circuitos no receptor e no transmissor.

Por último, existe os intervalos de endereçamento horizontais e verticais, períodos em que dados RGB válidos são transferidos para a tela (ILI ..., 2011). A área composta por esses endereçamentos é denominada como a área ativa da tela (STMICROELECTRONICS, 2017a).

Essas informações são fornecidas pelo fabricante da tela ou do controlador a ser utilizado. A Tabela 1 mostra essas informações para a tela utilizada neste trabalho e a Seção 2.4.4 como estas informações são utilizadas para o cálculo do *clock* do *pixel*, denominado DOTCLK.

Tabela 1 – Períodos de sincronização para uma tela QVGA

Parâmetro	Símbolo	Mín.	Típico	Máx.	Unidade
Sincronização Horizontal	H_{SYNC}	2	10	16	DOTCLK
Back Porch Horizontal	HBP	2	20	24	DOTCLK
Endereçamento Horizontal	H_{Adr}	-	240	-	DOTCLK
Front Porch Horizontal	HFP	2	10	16	DOTCLK
Sincronização Vertical	V_{SYNC}	1	2	4	Linha
Back Porch Vertical	VBP	1	2	-	Linha
Endereçamento Vertical	V_{Adr}	-	320	-	Linha
Front Porch Vertical	VFP	3	4	-	Linha

Fonte: adaptado de *ILI Technology Corp.* (2011, p. 46).

2.4.3 HABILITAÇÃO DE DADOS

O sinal de habilitação de dados (*Data Enable* - DE) tem a função de indicar que os dados presentes no barramento de dados RGB são válidos e devem ser transferidos à tela (STMICROELECTRONICS, 2017a). Assim, esse sinal permanece ativo durante a área ativa da tela, determinado pelos tempos de endereçamento horizontais e verticais.

2.4.4 *CLOCK* DO *PIXEL*

O *clock* do *pixel*, também chamado de *dot clock* é um sinal de referência para todos os outros sinais que a tela utiliza, responsável pela atualização dos estados desses sinais quando ocorre uma borda de subida ou descida. Essas bordas também servem para indicar que os dados presentes no barramento são válidos (STMICROELECTRONICS, 2017a; ILI ..., 2011).

Para calcular a frequência desse sinal, primeiro precisa-se do tamanho total da tela. Para calcular o tamanho total da tela, necessita-se a largura e altura total, que são calculadas com informações extraídas da folha de dados do fabricante, tal qual se encontra na Tabela 1. O cálculo se encontra nas equações (2) e (3) (ILI ..., 2011).

$$L_{total} = H_{SYNC} + HBP + H_{Adr} + HFP \tag{2}$$

$$A_{total} = V_{SYNC} + VBP + V_{Adr} + VFP \tag{3}$$

O *clock* do *pixel* é calculado por (4), em que F_{AT} é a taxa de atualização da tela (STMICROELECTRONICS, 2017a).

$$DOTCLK = L_{total} \times A_{total} \times F_{AT} \tag{4}$$

Assim, usando (2) e (3) em (4), tem-se (5).

$$DOTCLK = (H_{SYNC} + HBP + H_{Adr} + HFP) \times (V_{SYNC} + VBP + V_{Adr} + VFP) \times F_{AT} \tag{5}$$

Que, aplicado a tela presente no *kit* de avaliação, com os valores da Tabela 1 e uma taxa de atualização de 60 Hz, tem-se o *clock* do *pixel* necessário de:

$$DOTCLK = (10 + 20 + 240 + 10) \times (2 + 2 + 320 + 4) \times 60 = 5{,}51 \text{ MHz}$$

2.5 ARQUITETURA

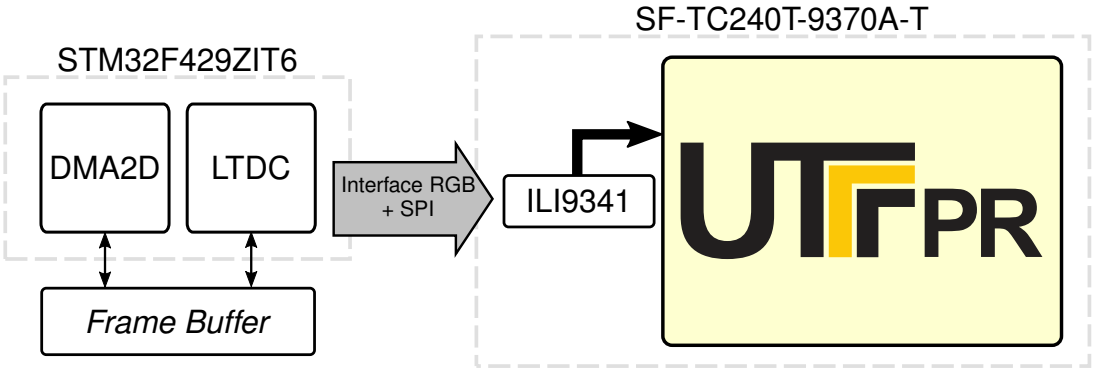
Para realizar a explanação da arquitetura de conexão entre os componentes de um sub-sistema gráfico, procurou-se utilizar como base uma arquitetura que é muito presente na literatura, que são os microcontroladores da série STM32 da *STMicroelectronic*s. Os fabricantes procuram incluir *hardware* que busca otimizar a troca de dados entre a tela e o microcontrolador, como aceleradores gráficos e controladores embutidos com funções específicas, como controle da tela e do painel sensível ao toque.

No caso da *STMicroelectronics*, há um acelerador gráfico, denominado DMA2D, um controlador de tela embutido ao próprio microcontrolador, denominado LTDC. O detalhamento terá enfoque no *kit* de avaliação STM32F429I-DISCO, que além do DMA2D e LTDC, possui uma tela com o controlador ILI9341 integrado e um controlador de painel de toque, com a comunicação de controle com o ILI9341 ocorrendo pelo protocolo SPI. Outro exemplo da utilização desta mesma arquitetura, porém com módulo e controlador de tela diferentes é o *kit* STM32F746G-DISCO, que também faz uso do protocolo SPI. Porém a comunicação de controle com o módulo da tela pode ocorrer por outro protocolo que não seja o SPI, podendo ser qualquer um dos padrões a Seção 2.3, como o *kit* STM32F769I-DISCO, que faz uso de MIPI-DSI.

É importante ressaltar que os módulos DMA2D e LTDC são padrões a arquitetura desenvolvida por esta fabricante e que outros fabricantes possuem soluções próprias para otimizar a conexão e realizar a comunicação. A *Freescale*, por exemplo, possui o módulo 2D-ACE para animação bidimensional e utiliza padrões de aceleração gráfica abertos, o OpenVG e o OpenGL, em suas soluções embarcadas baseadas nos núcleos de processamento ARM Cortex-A (FREESCALE ..., 2015).

Um diagrama básico da interação entre esses componentes é visto na Figura 10, considerando o *frame buffer* presente na memória SDRAM externa ao microcontrolador. Isso resulta em uma integração de componentes com três dispositivos, visto na Figura 7 da Seção 2.2.

Figura 10 – Diagrama de interação entre os controladores presentes no *kit* de avaliação STM32F429I-DISCO



Fonte: autoria própria.

De forma sucinta, o controlador da tela interno ao microcontrolador provê os sinais RGB que servem de entrada ao controlador embutido a tela do dispositivo, o qual, de fato, exibe a imagem. Nesta seção é mostrada não só como os componentes estão conectados uns aos outros, mas também o funcionamento e características de cada um deles.

2.5.1 CONTROLADOR EMBUTIDO A TELA

O ILI9341, da *ILI Technology Coporation*, tem a função de prover um *driver* de 720 canais de fonte (*source*), referentes as 240 linhas da tela com 3 *subpixels* cada, de cores vermelha, verde e azul, resultando no padrão RGB usado. Também há 320 canais de porta (*gate*) referente à quantidade de colunas. Isto faz com que apenas uma das 320 linhas de *pixels* de altura da tela esteja acionada com informação de intensidade luminosa proveniente da memória gráfica.

Porém ao acionar cada uma destas linhas de forma rápida o suficiente, tem-se a impressão de que todas as linhas estão acesas, graças a persistência da visão do olho humano. Assim, tem-se a necessidade de utilizar apenas 720 transistores de acionamento para acender cada linha. Caso contrário, necessitando acionar todas as linhas de uma só vez, seriam necessários 230.400 transistores. (ILI ..., 2011).

Além disso, há uma memória RAM de 240x320 palavras de 18 *bits*, totalizando 172.800 *bytes* para dados gráficos, quantidade suficiente para um quadro em resolução QVGA. Também são gerados os níveis de tensão necessários para o LCD, assim como o gerenciamento modo de economia de energia, controle de brilho e correção de gama (ILI ..., 2011).

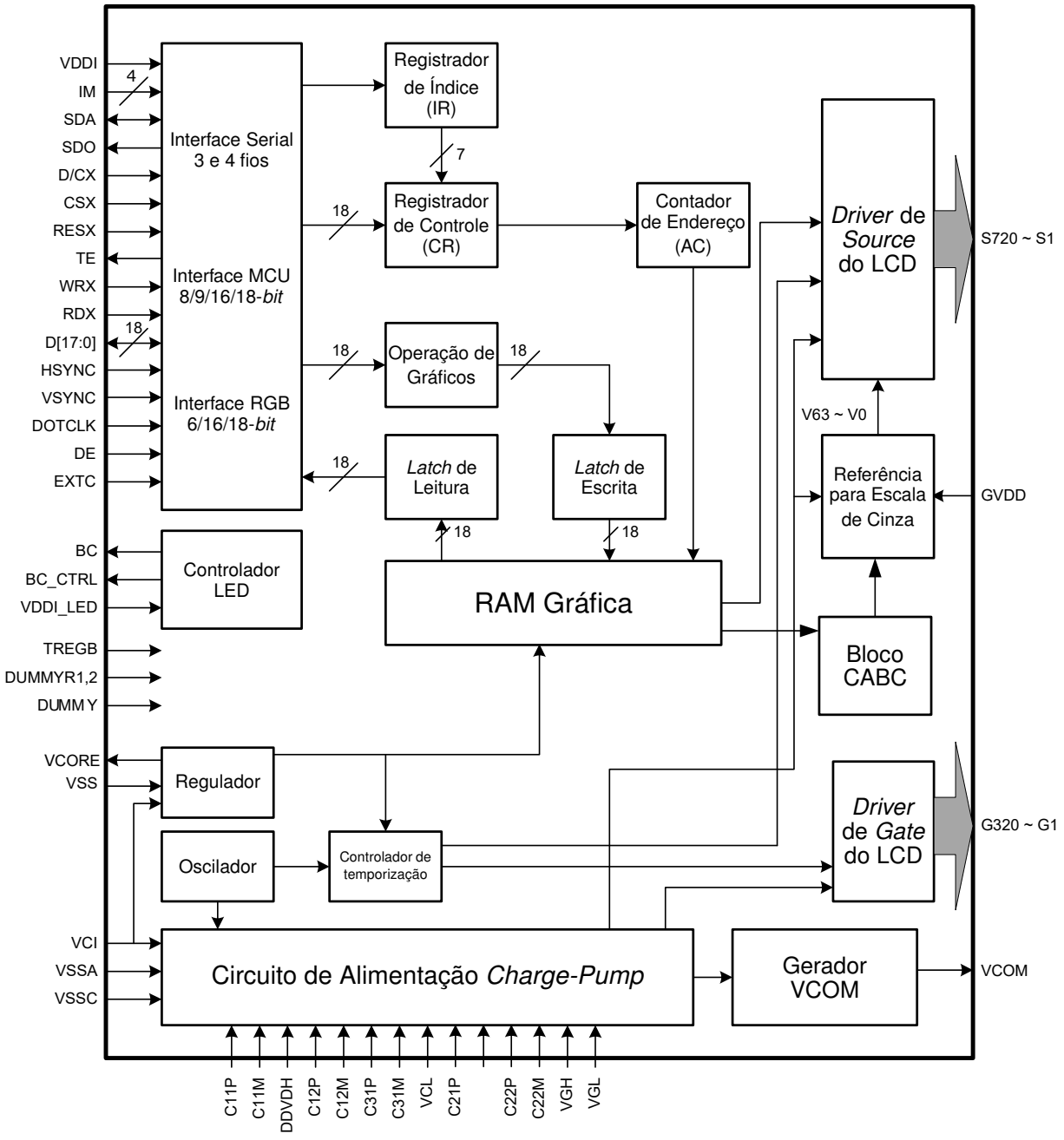
Observa-se na Figura 11 o diagrama de blocos do controlador, mostrando como os dados são recebidos pela interface e escritos na memória gráfica, para que esses sinais sejam levados ao *driver* dos transistores do TFT para que se tornem cores a serem exibidas por meio dos *pixels* no painel de LCD.

2.5.1.1 INTERFACES

O controlador provê várias opções de interface para envio de dados e comandos. Caso opte-se por um barramento paralelo, o modelo da espécie Intel 8080 é oferecido. Essa interface é baseada em quatro sinais de controle e um barramento de dados de largura variável (STMICROELECTRONICS, 2008).

Nesse caso, os sinais de controle são para seleção do *chip* (CSX), seleção entre dados e comandos (D/CX), leitura e escrita (RDX e WRX). As possíveis larguras do barramento de dados são de 8, 9, 16 e 18 *bits*. É realizada uma diferenciação em 8080-I e 8080-II, em que a diferença está em quais *bits* do barramento de dados são usados para acesso a registradores e quais são usados para envio de dados à memória gráfica (ILI ..., 2011).

Figura 11 – Diagrama de blocos do controlador de tela ILI9341



Fonte: adaptado de *ILI Technology Corp.* (2011, p. 9).

Também há a opção de uso de uma interface serial, de três ou quatro sinais, sendo que 9 ou 8 *bits* são enviados por transmissão, respectivamente. O nono *bit* na interface de três linhas é utilizado para a diferenciação entre envio de comando e dados (D/CX), o qual é um sinal separado na interface de quatro sinais. Os três sinais comuns a ambas as interfaces seriais são a seleção de *chip* (CSX), a entrada de *clock* (SCL) e a troca de dados (SDA) (ILI ..., 2011).

Assim como na interface Intel 8080, também há uma distinção entre uma interface do tipo I e II, sendo que a segunda usa dois sinais para a troca de dados ao invés de somente uma, sendo uma para entrada (SDI) e outra para saída (SDO) (ILI ..., 2011).

Para fazer uso de qualquer uma dessas interfaces, é necessário utilizar os pinos de modo de interface, denominados IM[3:0] na Figura 11. A Tabela 2 mostra que valores definem especificamente quais interfaces. Para o *kit* de desenvolvimento em questão, a interface é fixada na serial 4-linhas tipo I por meio de resistores de *pull-up* e *pull-down*, visto no diagrama esquemático da Figura 12.

Além desta interface serial, é notável também que há a conexão com o barramento de dados do controlador. Isso ocorre pois a interface serial não é utilizada para a transmissão de dados de *pixel*, mas para inicializar e configurar o controlador. Nesse caso, após a inicialização, os dados da imagem a ser exibida são enviadas por meio da interface RGB por um módulo interno ao microcontrolador do *kit* de desenvolvimento. Esse módulo é explanado com mais detalhes na Seção 2.5.2.

Tabela 2 – Definição de interfaces para o controlador TFT-LCD ILI9341

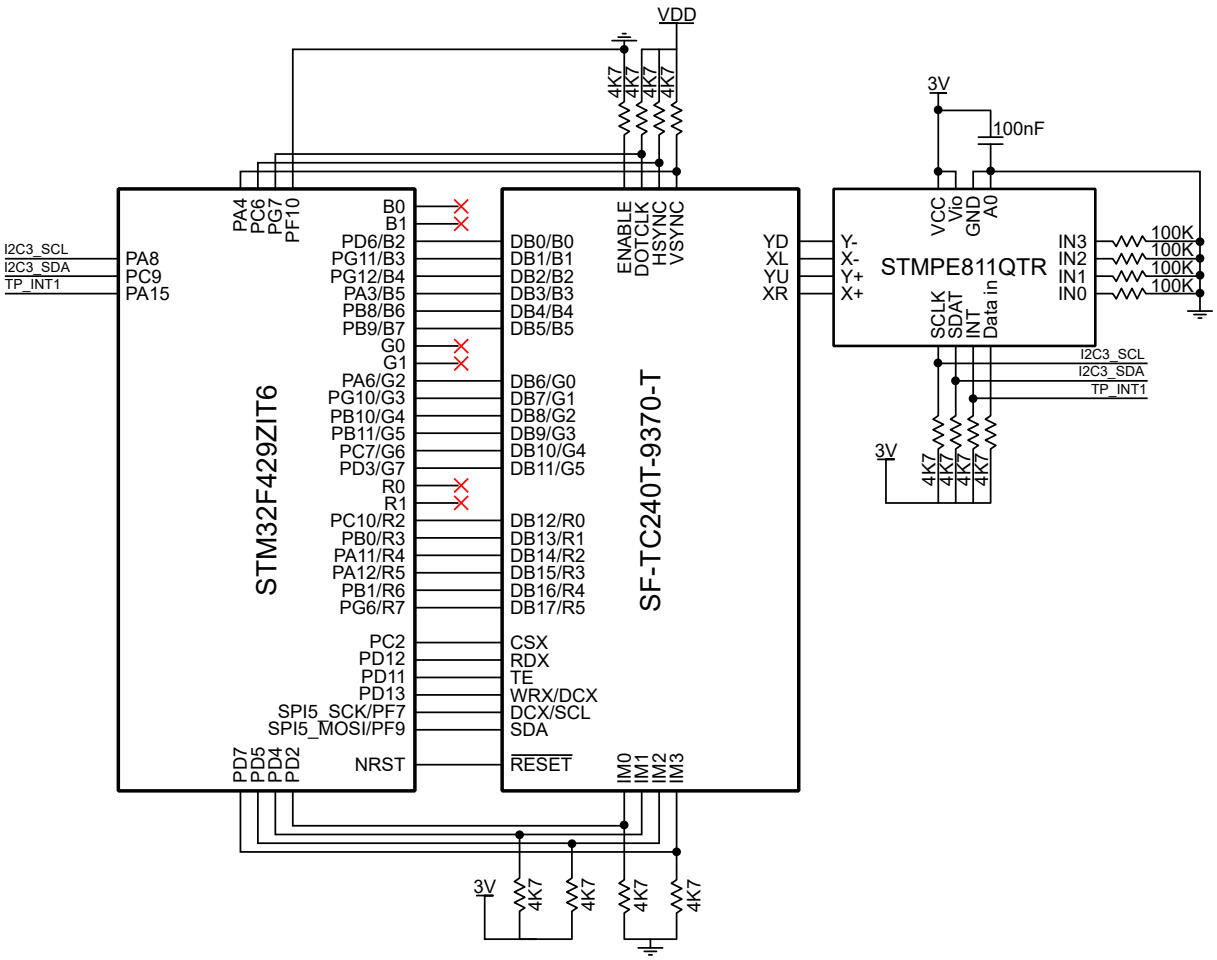
IM[3:0]	Interface de MCU	Pinos em Uso	
		Reg./Dados	Memória Gráfica
0000	8080 8-bit I	D[7:0]	D[7:0], WRX, RDX, CSX, D/CX
0001	8080 16-bit I	D[7:0]	D[15:0], WRX, RDX, CSX, D/CX
0010	8080 9-bit I	D[7:0]	D[8:0], WRX, RDX, CSX, D/CX
0011	8080 18-bit I	D[7:0]	D[17:0], WRX, RDX, CSX, D/CX
0101	Serial 3 linhas 9-bit I	SCL, SDA, CSX	
0110	Serial 4 linhas 8-bit I	SCL, SDA, D/CX, CSX	
1000	8080 8-bit II	D[8:1]	D[17:10], D[8:1], WRX, RDX, CSX, D/CX
1001	8080 16-bit II	D[17:10]	D[17:10], WRX, RDX, CSX, D/CX
1010	8080 9-bit II	D[8:1]	D[17:0], WRX, RDX, CSX, D/CX
1011	8080 18-bit II	D[17:10]	D[17:9], WRX, RDX, CSX, D/CX
1101	Serial 3 linhas 9-bit II	SCL, SDI, SDO, CSX	
1110	Serial 4 linhas 8-bit II	SCL, SDI, SDO, D/CX, CSX	

Fonte: adaptado de *ILI Technology Corp.* (2011, p. 24).

Dentre as interfaces RGB, há dois modos de operação. O primeiro deles é por meio do uso do sinal de habilitação de dados (DE), explicitado na Seção 2.4.3.

O segundo modo, denominado modo de sincronização (SYNC), ignora o sinal DE e determina a validade dos dados a serem enviados pelos intervalos de sincronização, definidos na Seção 2.4.2. Esses são denominados por meio do registrador RCM (ILI ..., 2011).

Figura 12 – Esquemático de conexões entre o microcontrolador, o módulo LCD e o *driver* de interação via toque na tela no *kit* de avaliação STM32F429I-DISCO



Fonte: adaptado de *STMicroelectronics* (2013, p. 5 e 7).

Dentre esses dois modos, a interface RGB pode utilizar 6, 16 ou 18 *bits* do barramento de dados. Para 6 *bits*, é possível trabalhar no padrão RGB565 e RGB666, que dispõem de 65.536 e 262.144 cores, respectivamente, denominados 65K e 262K cores. Em 16 *bits*, somente o padrão RGB565 é possível e para 18 *bits*, RGB666 (ILI ..., 2011). A seleção é feita pelos registradores RIM e DPI. A Tabela 3 ilustra essa definição.

Algo a se notar diante do diagrama esquemático da Figura 12 é a não conexão dos *bits* menos significativos do barramento de dados RGB. Isso ocorre devido ao controlador de tela interno ao microcontrolador ter sua saída no formato RGB888, o

que acarreta em mais sinais do que o disponível para conexão. No caso do uso de telas com uma paleta de cores menor, a conexão do barramento deve ser feita com os *bits* mais significativos (STMICROELECTRONICS, 2017a).

Tabela 3 – Seleção de Interface e Modo RGB no controlador ILI9341

RCM	RIM	DPI	Interface RGB	Modo RGB	Pinos Utilizados
10	0	110	18-bit 262K cores	DE	D[17:0] VSYNC, HSYNC, DE, DOTCLK
10	0	101	16-bit 65K cores	DE	D[17:13], D[11:1] VSYNC, HSYNC, DE, DOTCLK
10	1	110	6-bit 262K cores	DE	D[5:0] VSYNC, HSYNC, DE, DOTCLK
10	1	101	6-bit 65K cores	DE	D[5:0] VSYNC, HSYNC, DE, DOTCLK
11	0	110	18-bit 262K cores	SYNC	D[17:0] VSYNC, HSYNC, DE, DOTCLK
11	0	101	16-bit 65K cores	SYNC	D[17:13], D[11:1] VSYNC, HSYNC, DE, DOTCLK
11	1	110	6-bit 262K cores	SYNC	D[5:0] VSYNC, HSYNC, DE, DOTCLK
11	1	101	6-bit 65K cores	SYNC	D[5:0] VSYNC, HSYNC, DE, DOTCLK

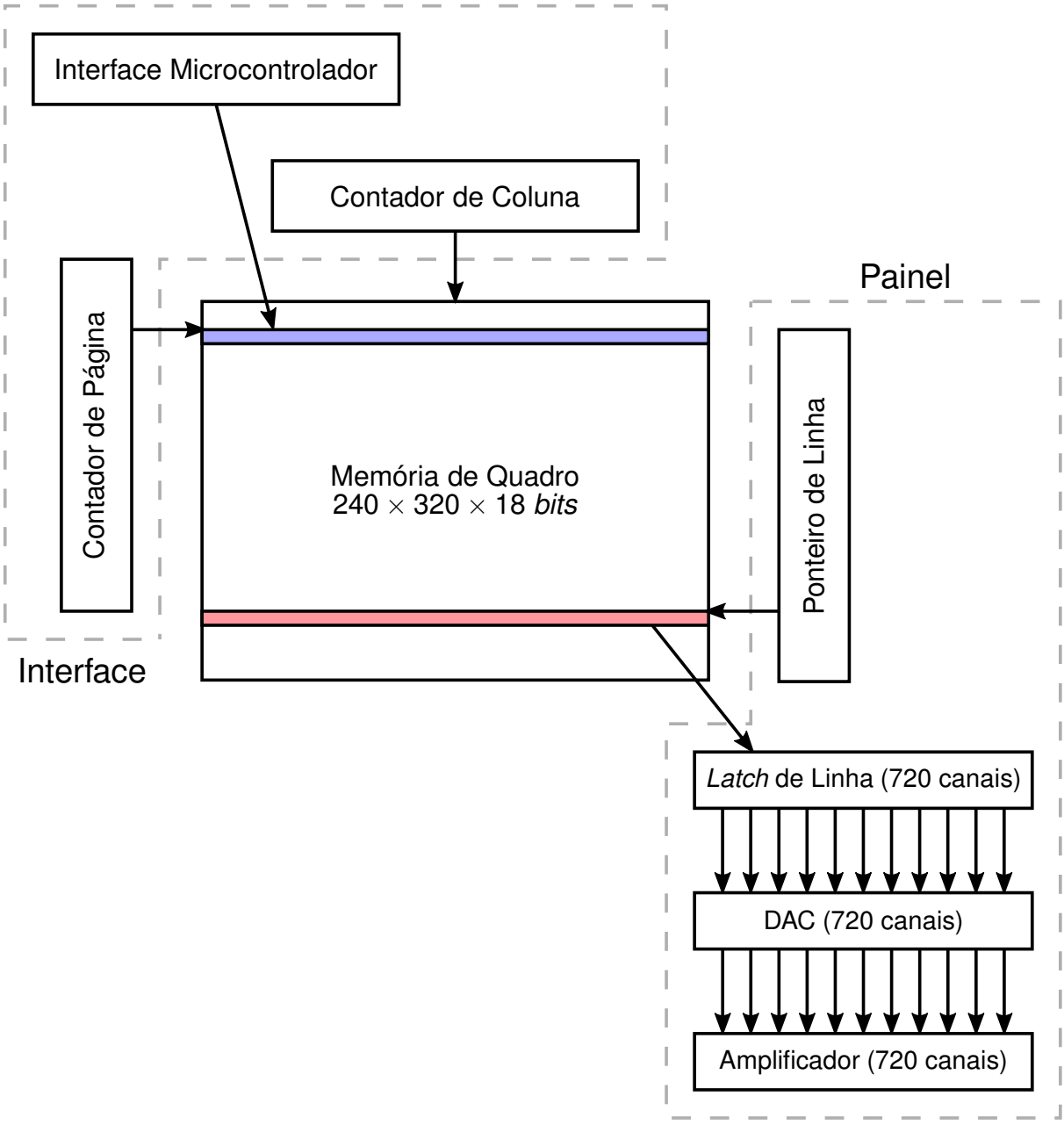
Fonte: adaptado de *ILI Technology Corp.* (2011, p. 24).

2.5.1.2 MEMÓRIA GRÁFICA

Consistindo de 1.382.400 *bits*, a memória gráfica armazena os pontos da tela suficientes para um quadro em resolução QVGA a 18 BPP. Na Figura 13, observam-se os contadores e ponteiros que acessam a GRAM. Os contadores têm o propósito de permitir que o microcontrolador conectado a interface externa escreva os dados de *pixel* na memória do controlador para que então o ponteiro de linha os leia para exibição na tela.

Mesmo que um contador e o ponteiro de linha estejam carregados com o mesmo endereço, não há restrição de acesso à memória. Caso o painel venha a ler um endereço para transferência ao Conversor Digital-Analógico (*Digital to Analog Converter* - DAC) ao mesmo tempo que a interface lê ou escreve dados nesse local, não haverá efeitos visuais anormais na tela (ILI ..., 2011).

Figura 13 – Diagrama de interação da interface e do painel com a memória gráfica



Fonte: adaptado de *ILI Technology Corp.* (2011, p. 203).

É possível notar a partir da Figura 13 de que forma as informações são enviadas da memória gráfica para serem exibidas na tela. Os dados de cada linha da memória são enviados ao *Latch* de Linha de 720 canais (240 *pixels* com três *subpixels* cada), para que então sejam convertidos em uma tensão analógica e amplificados.

O fluxo de dados do microcontrolador para a memória gráfica ocorre sempre na mesma ordem, porém a direção em que esses dados são escritos pode ser modificada, fazendo com que a imagem resultante na tela seja rotacionada, espelhada ou ambas, nos eixos horizontal e vertical. Isso ocorre por meio do controle de como

os contadores de página e coluna se comportam durante a escrita. O controlador também oferece opção para rolar a imagem exibida, que é realizada pela mudança do comportamento do ponteiro de linha que transfere a imagem da memória para o DAC. O comportamento padrão é começar do canto superior esquerdo, escanear da esquerda a direita, linha a linha, até atingir o canto inferior direito (ILI ..., 2011).

2.5.2 CONTROLADOR EMBUTIDO AO MICROCONTROLADOR

Com o intuito de prover sinais de dados e controle para módulos de tela que não possuem controladores internos ou a não presença de memória gráfica, os microcontroladores da série STM32 da *STMicroelectronics* possuem um controlador interno, denominado LTDC (STMICROELECTRONICS, 2017a).

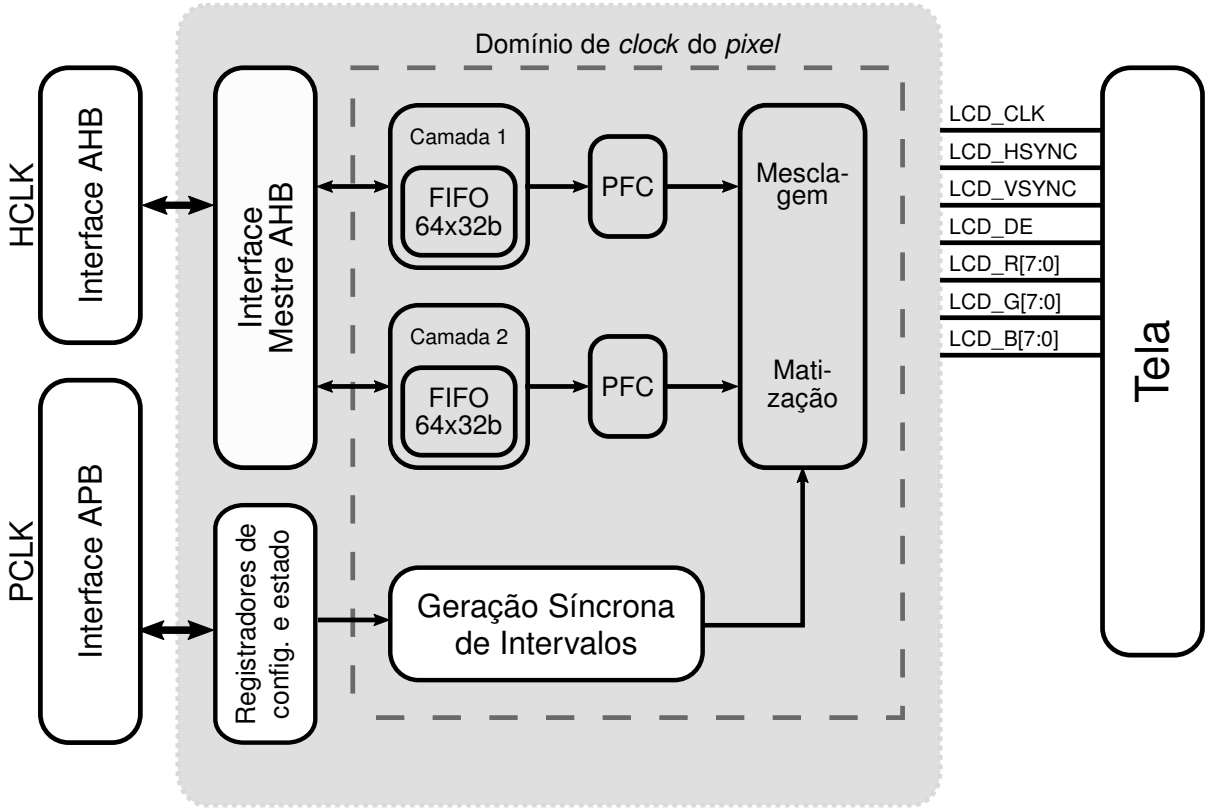
Esse controlador fornece os sinais para a interface RGB, explicitados na Seção 2.4, para vários painéis de TFT e LCD até a resolução XGA (1024x768), por meio do ajuste dos tempos de sincronização, além de suas polaridades, permitindo uma largura do barramento de dados de até 24 *bits*. Além disso, o controlador dispõe de duas camadas de exibição de imagens, que podem ser mesclados de formas diferentes ao gerar a imagem final. A Figura 14 mostra o diagrama de blocos do controlador (STMICROELECTRONICS, 2017b).

Para vários *kits* de desenvolvimento da série STM32, o LTDC é utilizado em conjunto com um módulo de tela que possui um controlador e memória gráfica. Como já explicitado na Seção 2.5.1.1, o *kit* STM32F429I-DISCO utiliza um módulo com o controlador ILI9341 em uma combinação da interface RGB, com sinais gerados pelo LTDC e comunicação serial para configuração do módulo. Outro exemplo é o *kit* de desenvolvimento STM32F769I-DISCO, o qual também utiliza o LTDC, porém em conjunto com outro módulo de tela, o qual possui como base o controlador OTM8009A, tendo como diferença que a comunicação serial é dada pelo protocolo MIPI-DSI, explanados na Seção 2.3.

Além disso, apesar de o nome aludir o controle apenas a módulos LCD-TFT, o fabricante menciona que com o LTDC é possível controlar módulos de outras tecnologias, como as de matriz ativa de diodo orgânico emissor de luz (*Active-Matrix Organic Light-Emitting Diode* - AMOLED), visto que também utilizam a interface RGB para seu controle (STMICROELECTRONICS, 2017a).

O funcionamento geral se dá pela leitura dos dados das imagens linha a linha e o modo de acesso a memória do LTDC é de 64 *bytes* por vez. Mas, caso o final de uma linha é atingido e há menos de 64 *bytes* remanescentes, o controlador transfere a quantidade de *bytes* restantes para completar 64 *bytes*, denominados *dummy bytes*.

Figura 14 – Diagrama de blocos do LTDC



Fonte: adaptado de *STMicroelectronics* (2017, p. 20).

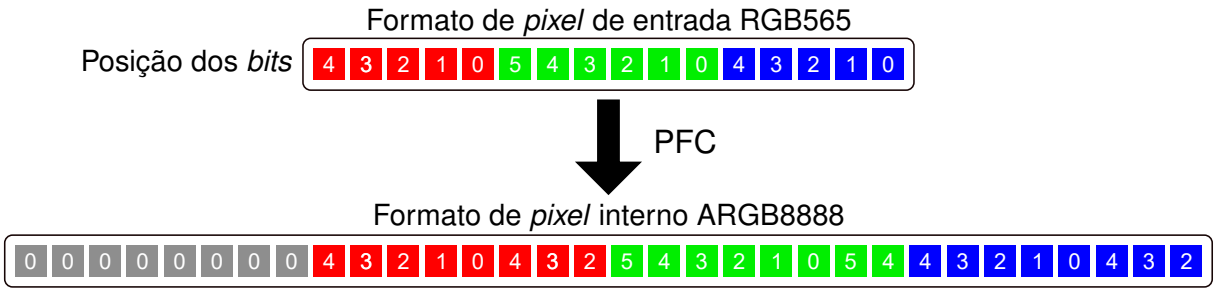
A partir disso, quando se está dentro da área ativa da tela, os dados de um *pixel* são transferidos para a camada correspondente a partir de seu *frame buffer* específico e convertido para um formato de *pixel* interno denominado ARGB8888. Com isso, o *pixel* pode ser mesclado com a outra camada e/ou o plano de fundo, para que resulte em um *pixel* RGB888 levado à interface RGB para ser exibido na tela (STMICROELECTRONICS, 2017a).

Além dos formatos de 16 e 18 *bits* do controlador embutido a tela, o LTDC também oferece suporte ao formato de 24 *bits*, com oito *bits* utilizados por *subpixel*, resultando no padrão RGB888. A Figura 15 ilustra a conversão de um *pixel* RGB565 para o formato interno ARGB8888. Em essência, os oito *bits* mais significativos são escolhidos e caso a componente tenha menos que oito *bits*, a componente é expandida por meio de replicação dos *bits* (STMICROELECTRONICS, 2017a).

Três domínios de *clock* são utilizados para o funcionamento do controlador, sendo o primeiro do *High-speed Clock* (HCLK) que é o *clock* principal do microcontrolador, utilizado para transferência de dados das memórias para a memória FIFO (*First In, First Out*, que significa primeiro a entrar e primeiro a sair). O segundo é do *Peripheral Clock* (PCLK), utilizado para acessar os registradores de configuração e de estado.

Por último, o domínio do *clock* de *pixel*, denotado LCD_CLK é utilizado para gerar os sinais para a interface do painel TFT-LCD (STMICROELECTRONICS, 2017a).

Figura 15 – Conversão de formato de *pixel* no LTDC, de RGB565 para ARGB8888



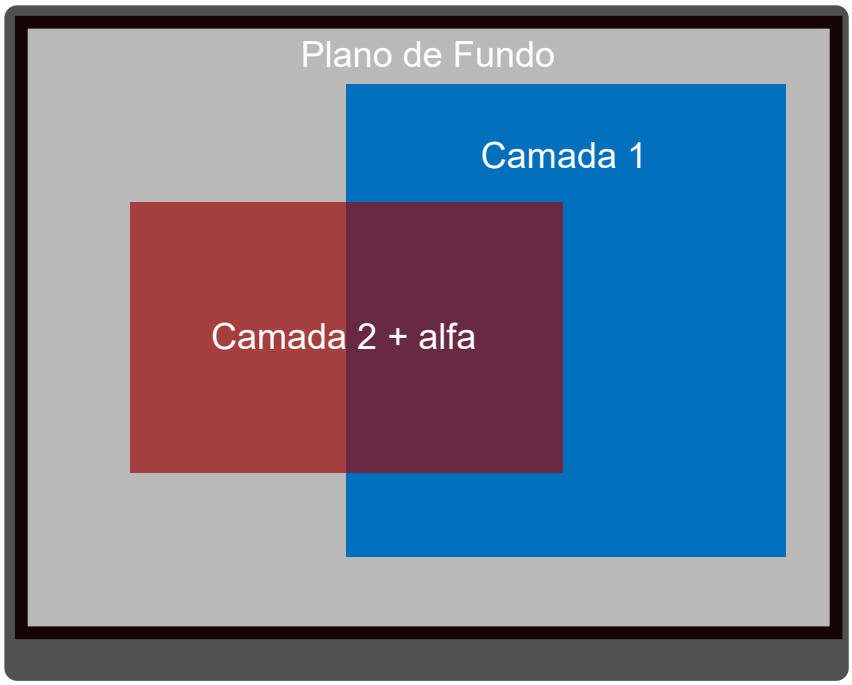
Fonte: adaptado de *STMicroelectronics* (2017, p. 28).

2.5.2.1 CAMADAS E MESCLAGEM

O LTDC permite a utilização de até duas camadas. No que concerne ao armazenamento, cada camada é um *frame buffer* distinto, cada um podendo conter configurações diferentes. Essas configurações incluem o endereço deste *frame buffer*, o qual pode ser uma imagem estática pré-armazenada na memória do dispositivo ou uma área de memória que pode ser modificada. Também permite-se habilitar separadamente cada uma destas camadas, alterar o nível de transparência e o formato de *pixel* presente em cada uma delas.

A ordem de exibição dessas camadas é fixa de modo a ordem seja de baixo para cima, começando pelo plano de fundo, definido por uma cor constante, seguido da primeira camada e então da segunda. O controlador também dispõe de fatores de mesclagem das camadas, usando um valor denominado α. A ordem de mesclagem também segue a ordem de exibição, sendo primeiramente o plano de fundo mesclado com a primeira camada, e o resultado então mesclado com a segunda camada. A Figura 16 ilustra a mesclagem de duas camadas com o plano de fundo (STMICROELECTRONICS, 2017a).

Figura 16 – Exemplo de resultado da mesclagem de duas camadas com o plano de fundo no LTDC



Fonte: adaptado de *STMicroelectronics* (2017, p. 26).

Para cada camada, há dois fatores de mesclagem. A equação que calcula a cor resultante da interação é dada por (6), em que BC é a cor resultante, BF_1 é o primeiro fator de mesclagem, BF_2 o segundo, C é a cor da camada em questão e C_S é a cor da camada subjacente. O processo se repete para as três componentes de cores para obter-se o *pixel* resultante (STMICROELECTRONICS, 2017b).

$$BC = BF_1 \times C + BF_2 \times C_S \tag{6}$$

Duas opções estão disponíveis para cada fator de mesclagem, explicitadas por (7) e (8), em que α_c, denominado α constante, define um valor de transparência constante para a camada inteira e α_p é denominado α do *pixel*, sendo utilizado o valor da componente A do *pixel* atual para cálculo. Esses valores α são divididos por 255 em *hardware* para então serem calculados, fazendo com que os valores de α_p fiquem em um intervalo entre 0 e 1 (STMICROELECTRONICS, 2017b).

$$BF_1 = \alpha_c \text{ ou } \alpha_p \times \alpha_c \tag{7}$$

$$BF_2 = (1 - \alpha_c) \text{ ou } 1 - (\alpha_p \times \alpha_c) \tag{8}$$

Um dos usos mais conhecidos para os recursos de mesclagem e transparência foi no sistema operacional *Windows 7*, denominado *Windows Aero*, que buscou

simular um efeito de vidro translúcido, para as janelas de aplicação (ALENCAR, 2016). A Figura 17 mostra uma captura de tela do sistema operacional utilizando o recurso.

Figura 17 – Transparência e mesclagem no *Windows Aero*

Fonte: TechTudo (2016).

Também é permitido posicionar e redimensionar as camadas durante o tempo de execução, desde que estejam dentro da área ativa da tela. Isso permite definir o primeiro e o último *pixel* de uma linha, assim como a primeira e última linha a serem mostradas nessa janela, sendo possível mostrar uma imagem que tome até todo o tamanho da área ativa. A Figura 18 exemplifica quando apenas uma parte da camada é mostrada na imagem (STMICROELECTRONICS, 2017a).

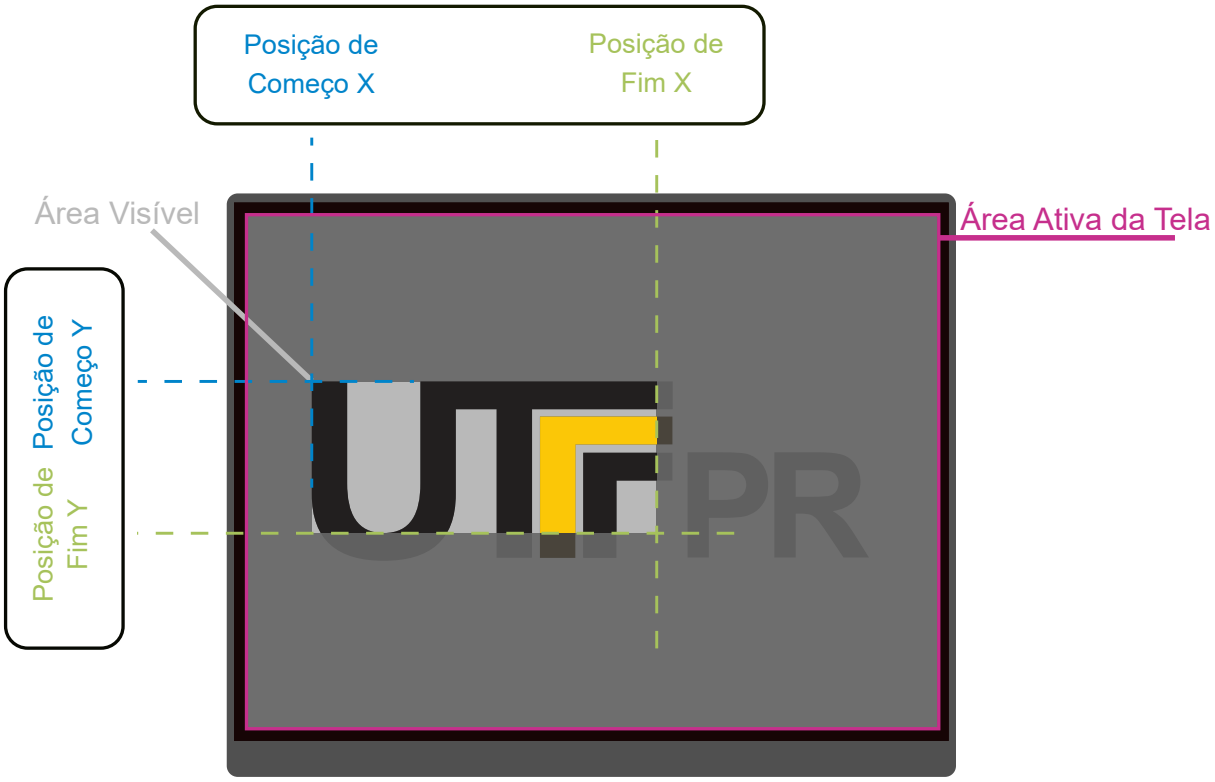
2.5.2.2 MATIZAÇÃO

A técnica de matização (do inglês, *Dithering*), tem o objetivo de criar uma tonalidade intermediária a partir de tons adjacentes, por meio da intercalação entre *pixels* de duas cores diferentes em duas áreas adjacentes. Essa técnica se aproveita da tendência do olho humano em atenuar a distinção entre pontos de cores diferentes, fundindo-os em uma única cor ou tom (ELGSCREEN, 2014).

No caso da tela e do microcontrolador em questão, a matização é aplicável ao exibir imagens de 24 *bits* em uma tela de 18 *bits*, tal qual ocorre na interação do LTDC com o módulo LCD do *kit* de avaliação (STMICROELECTRONICS, 2013).

Essa técnica é realizada por meio de um método pseudo-aleatório, em que um registrador de deslocamento de retroalimentação linear (*Linear Feedback Shift*

Figura 18 – Posicionamento e redimensionamento de uma camada



Fonte: adaptado de *STMicroelectronics* (2017, p. 26).

Register - LFSR) adiciona pequenos valores aleatórios para cada componente de cor do *pixel*, o que às vezes ocasiona no arrendondamento do *bit* mais significativo ao mostrar dados de *24* bits em uma tela de 18 *bits*. O método pseudo-aleatório é o mesmo que comparar alguns *bits* menos significativos a um valor de limiar e adicionar uma unidade ao *bit* mais significativo caso o limiar é excedido, já que os bits menos significativos são desconsiderados, uma vez aplicada a matização (STMICROELECTRONICS, 2017b).

A largura do valor pseudo-aleatório adicionado é de dois *bits* para cada componente de cor do *pixel*. Uma vez que o LTDC é habilitado, o LFSR entra em atividade e opera a partir do primeiro *pixel* da área ativa da tela, continuando a operar mesmo fora desta área (STMICROELECTRONICS, 2017b).

2.5.3 ACELERADOR GRÁFICO

O acelerador gráfico, denotado DMA2D, consiste de um controlador de DMA dedicado a manipulação de imagens. Algumas de suas principais características incluem o preenchimento inteiro ou parcial de uma imagem com uma cor específica, cópia inteira ou parcial de uma imagem fonte para uma imagem destino, com e sem

conversão de formato de *pixel* e realizar a mesclagem completa ou parcial de duas imagens diferentes – mesmo que essas tenham formatos de *pixel* distintas – para outra imagem (STMICROELECTRONICS, 2017b).

Devido a grande quantidade de manipulação de formato de *pixels* e cores que ocorrem dentro do DMA2D, a próxima seção será destinada a explanação desses formatos e o conceito de cor indireta com o uso de tabela de consulta de cores (*Color Look-Up Table* - CLUT).

2.5.3.1 COR DIRETA E COR INDIRETA

O modo de cor direto é o mais simples, pois a memória onde é armazenada a imagem contém todos os bits que representam um pixel, por meio das componentes R, G, B e, opcionalmente, alpha (A). A componente A, denominada α, denota a transparência do *pixel*, sendo que em seu valor máximo ele é completamente opaco e em seu valor mínimo, transparente (STMICROELECTRONICS, 2017b).

A Tabela 4 mostra a ordem dos dados na memória entre as componentes de cores que se pode utilizar no modo de cor direta no DMA2D para entrada de dados, em uma largura de 32 *bits*. $C_n[p:q]$ denota uma componente C do n-ésimo *pixel*, caso haja mais de um *pixel* guardado nesse intervalo de memória, no intervalo de *bits* entre p e q. @ + n se refere a um endereço de memória qualquer, n *bytes* a frente de @.

Tabela 4 – Ordem dos dados nos formatos de *pixel* em modo de cores direto no DMA2D

Modo de Cor	@ + 3	@ + 2	@ + 1	@ + 0
ARGB8888	$A_0[7:0]$	$R_0[7:0]$	$G_0[7:0]$	$B_0[7:0]$
RGB888	$B_1[7:0]$	$R_0[7:0]$	$G_0[7:0]$	$B_0[7:0]$
	$G_2[7:0]$	$B_2[7:0]$	$R_1[7:0]$	$G_1[7:0]$
	$R_3[7:0]$	$G_3[7:0]$	$B_3[7:0]$	$R_2[7:0]$
RGB565	$R_1[4:0]G_1[5:3]$	$G_1[2:0]B_1[4:0]$	$R_0[4:0]G_0[5:3]$	$G_0[2:0]B_0[4:0]$
ARGB1555	$A_1R_1[4:0]G_1[4:3]$	$G_1[2:0]B_1[4:0]$	$A_0R_0[4:0]G_0[4:3]$	$G_0[2:0]B_0[4:0]$
ARGB4444	$A_1[3:0]R_1[3:0]$	$G_1[3:0]B_1[3:0]$	$A_0[3:0]R_0[3:0]$	$G_0[3:0]B_0[3:0]$

Fonte: adaptado de *STMicroelectronics* (2017, p. 343).

Já o modo de cores indireto, também chamado de pseudo-cor ou ainda, cor indexada, cada *pixel* é representado por um índice, normalmente de oito *bits*, que é associado a uma tabela de cores que contém toda a informação referente a cor desejada para então ser transferida a tela. A vantagem dessa abordagem é poder

utilizar menos espaço de armazenamento para guardar imagens, ou até mesmo, o *frame buffer*, mas com isso, a seleção de cores disponíveis é limitada, visto que com oito *bits*, tem-se um universo de 256 cores para escolher dentre todas as disponíveis (POYNTON, 2003; STMICROELECTRONICS, 2017b).

A componente principal do modo de cor indireto é a L, denotada luminância, que é um índice em uma CLUT que contém os valores para as componentes de cores diretas. Assim como no modo de cores direto, também há a existência da componente α_p. Há, ainda, dois modos α, denotados A4 e A8, não havendo informações de cor armazenada, apenas a opacidade, sendo que a cor é escolhida posteriormente (STMICROELECTRONICS, 2017b). A Tabela 5 mostra a ordem dos dados na memória no modo de cor indireto para os formatos de entrada de dados aceito pelo DMA2D.

Tabela 5 – Ordem dos dados nos formatos de *pixel* em modo de cores indireto no DMA2D

Modo de Cor	@ + 3	@ + 2	@ + 1	@ + 0
L8	L_3[7:0]	L_2[7:0]	L_1[7:0]	L_0[7:0]
AL44	A_3[3:0]L_3[3:0]	A_2[3:0]L_2[3:0]	A_1[3:0]L_1[3:0]	A_0[3:0]L_0[3:0]
AL88	A_1[7:0]	L_1[7:0]	A_0[7:0]	L_0[7:0]
L4	L_7[3:0]L_6[3:0]	L_5[3:0]L_4[3:0]	L_3[3:0]L_2[3:0]	L_1[3:0]L_0[3:0]
A8	A_3[7:0]	A_2[7:0]	A_1[7:0]	A_0[7:0]
A4	A_7[3:0]A_6[3:0]	A_5[3:0]A_4[3:0]	A_3[3:0]A_2[3:0]	A_1[3:0]A_0[3:0]

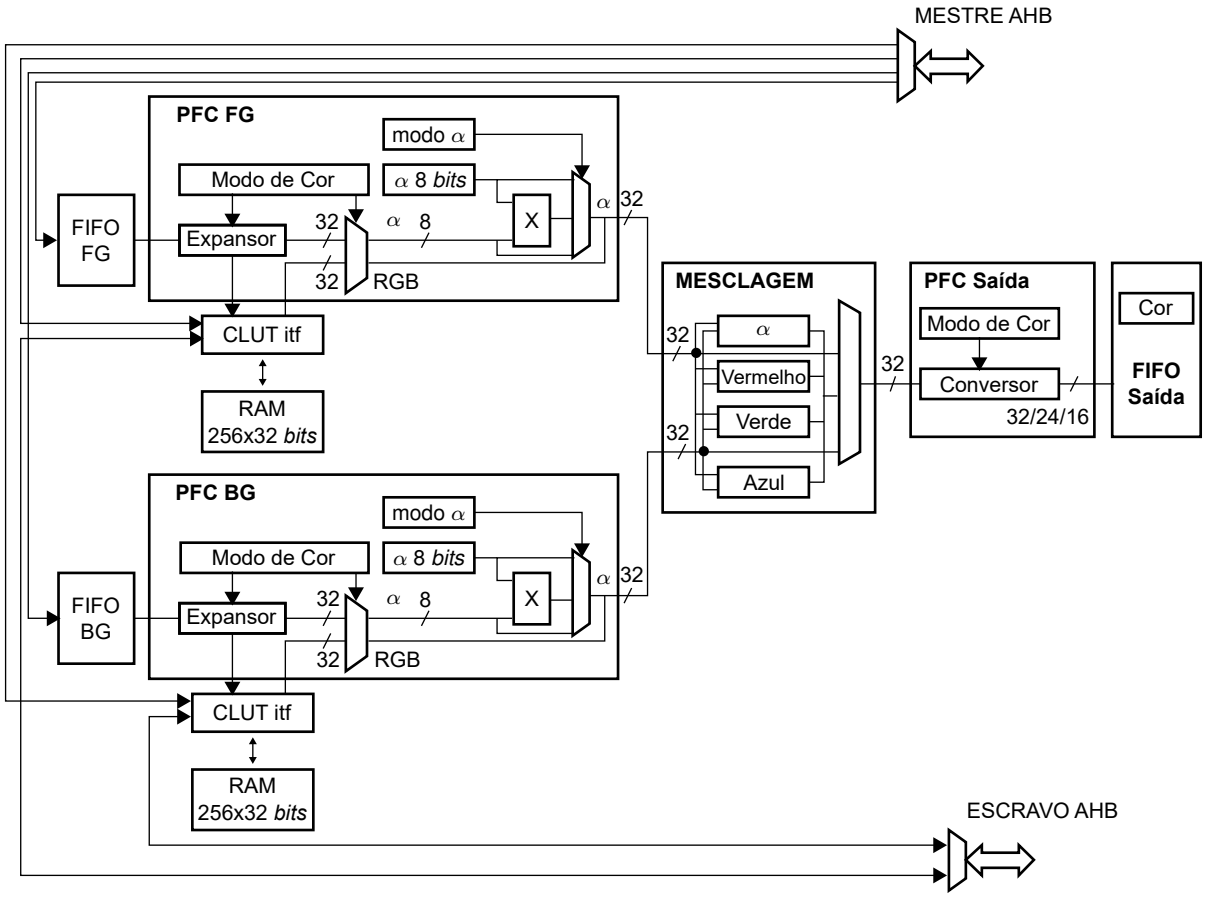
Fonte: adaptado de *STMicroelectronics* (2017, p. 343).

2.5.3.2 MODOS DE OPERAÇÃO

O DMA2D é um controlador que realiza transferências diretas de memória. A transferência de dados tanto de fonte como de destino podem ser destinados a periféricos e memórias em toda a área de memória, desde o endereço 0x00000000 a 0xFFFFFFFF, por meio de quatro modos de operação: registrador a memória, memória a memória, memória a memória com conversão de formato de *pixel* e memória a memória com conversão de formato de *pixel* e mesclagem (STMICROELECTRONICS, 2017b). O diagrama de blocos é apresentado na Figura 19.

As memórias FIFO presentes no controlador se comportam de maneira diferente dependendo do modo de operação em questão e ambos têm a função de transferir os dados de entrada a serem copiados e/ou processados. No modo registrador a memória, nenhum deles é usado. Em operações memória a memória, que não preci-

Figura 19 – Diagrama de blocos do DMA2D



Fonte: traduzido de *STMicroelectronics* (2017, p. 341).

sam de mesclagem ou conversão, somente o FIFO do plano principal (FG) é utilizado em modo de *buffer*. Caso exista a necessidade de conversão, mas sem mesclagem, o FIFO do plano principal é utilizado normalmente (STMICROELECTRONICS, 2017b).

O modo de transferência de registrador a memória é utilizado para preencher uma área com uma determinada cor. Não há busca de dados para entrada, pois o registrador é a fonte de dados contendo a cor específica que a área precisa ser preenchida. No modo memória a memória, o DMA2D não realiza nenhuma transformação gráfica. O FIFO do plano principal de entrada age como um *buffer* e os dados de entrada são transferidos à memória de destino (STMICROELECTRONICS, 2017b).

Caso haja a necessidade de somente converter o formato de *pixel*, o mesmo é convertido e transferido para a memória de destino. No uso de cor direta, as componentes são expandidas para oito *bits*. No caso de cor indireta, a tabela CLUT associada a imagem precisa ser carregada na memória RAM de CLUT do controlador, que é vista no diagrama de blocos da Figura 19. Enquanto a conversão de cores ocorre, um valor α pode ser adicionado, caso a imagem de origem não contenha essa

componente, ou modificado. Por fim, os 32 *bits* resultantes de dados passam pela conversão de saída, pelo bloco de conversão de formato de *pixel* (PFC) de saída e escritos na memória de destino. O formato de saída ocorre somente no modo de cor direta, visto que não há um processo de geração de CLUT pelo controlador (STMICROELECTRONICS, 2017b).

Quando é necessário usar conversão e mesclagem, são necessárias duas fontes de dados, uma transferida para o FIFO de plano de fundo (BG) e a outra para o primeiro plano. Essas imagens podem possuir formatos de *pixel* diferentes e caso seja usado modo de cor indireto, podem também ter suas próprias tabelas CLUT. Uma vez que o bloco PFC converteu um *pixel* de cada FIFO para o formato de 32 *bits*, eles são mesclados de acordo com (9), em que C é a componente R, G ou B do *pixel* em questão e α é a componente A (STMICROELECTRONICS, 2017b).

$$C_{saída} = \frac{C_{FG} \times \alpha_{FG} + C_{BG} \times \alpha_{BG} - C_{FG} \times \alpha_{mult}}{\alpha_{saída}} \tag{9}$$

$$\alpha_{saída} = \alpha_{FG} + \alpha_{BG} - \alpha_{mult} \tag{10}$$

$$\alpha_{mult} = \frac{\alpha_{FG} \times \alpha_{BG}}{255} \tag{11}$$

Para todos os modos de operação, os formatos de *pixel*, endereços de origem, destino e da CLUT (no caso de cor indireta), quantidade de dados, tamanho da CLUT e cor do modo registrador a memória são definidos por registradores específicos. Além disso, o controlador verifica a validade de várias configurações antes de iniciar uma transferência, gerando uma interrupção caso uma configuração inválida seja detectada (STMICROELECTRONICS, 2017b).

2.5.4 TELA SENSÍVEL AO TOQUE

Dependendo da aplicação em questão, pode ser que seja necessário o suporte de uma tela sensível ao toque, o que pode ser feito por meio de um painel separado da tela ou um módulo com a funcionalidade embutida. Em ambos os casos, os sinais advindos devem ser tratados pelo microcontrolador ou por um controlador a parte. Alguns desses sinais são analógicos e outros são digitais, devendo ser decodificados para obter-se as coordenadas de onde o evento de toque ocorreu (MICROCHIP ..., 2011).

Existem várias tecnologias de paineis de toque, a começar pela resistiva, presente no mercado há muitos anos, ainda muito utilizado pelo seu baixo custo e resis-

tência à umidade e sujeira, sendo uma opção indicada a uso externo. Outra tecnologia disponível é a capacitiva, que viu seu uso aumentar de forma significativa com a popularização dos *smartphones*. Apesar de não ser tão resistente a agentes externos, sua precisão e capacidade de registrar múltiplos toques simultâneos contribuem para o aumento da sua popularidade. Apesar dessas duas tecnologias serem as principais, ainda existem outras, como a de ondas acústicas de superfície (*Surface Acoustic Wave* - SAW) e infravermelho (BESSIN, 2017; GRUNSKE, 2017).

Assim como ocorre com os módulos de telas TFT-LCD, os paineis de toque também possuem controladores para o envio de informações sobre as coordenadas do toque além de outras funcionalidades pertinentes a cada tecnologia. Um exemplo disso são os controladores FT5336 e FT6x06 da *FocalTech* para telas capacitivas. Devido a maior complexidade ao possibilitar vários toques e até mesmo gestos, essas vêm acompanhadas de um microcontrolador para possibilitar características como auto-calibração e altas taxas de amostragem do toque – 80 e 120 Hz, respectivamente, no caso desses controladores (FOCALTECH ..., 2012; FOCALTECH ..., 2013).

Em relação as telas resistivas, tem-se como exemplo o módulo LCD SF-TC240T-9370A-T presente no *kit* de desenvolvimento deste trabalho. Esse possui quatro pinos destinados ao painel de toque, logo, sua configuração é de quatro linhas (SAEF ..., 2012). Além disso, vê-se no esquemático da Figura 12 da Seção 2.5.1 a presença do controlador STMPE811QTR, que é um controlador de tela sensível ao toque resistiva e expansor de entradas e saídas de propósito geral (*General Purpose Input/Output* - GPIO).

O controlador tem o intuito de reduzir a carga de processamento do microcontrolador e simplificar o tratamento de dados (STMICROELECTRONICS, 2011b). Assim, primeiramente será explicado brevemente o princípio de funcionamento de um painel capacitivo, seguido do princípio de funcionamento de uma tela de toque resistiva de quatro linhas para então explanar as características do controlador.

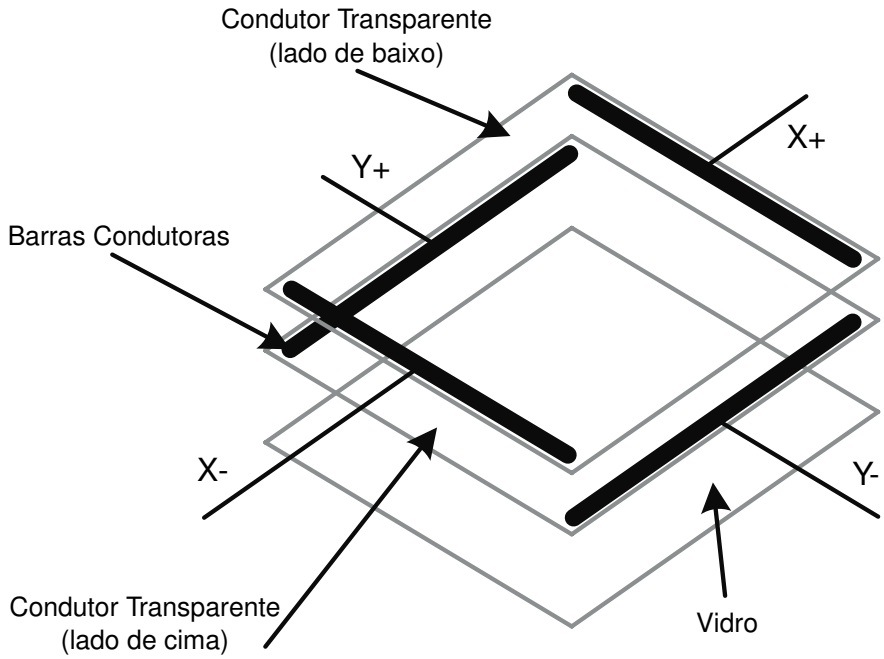
2.5.4.1 PRINCÍPIO DE FUNCIONAMENTO

As telas que usam tecnologia capacitiva são divididas em dois tipos: de projeção e superfície. Apesar da separação, ambas têm como base o uso de material condutivo colocado sobre a tela e então coberta por um material transparente, como vidro. Tomando proveito das propriedades elétricas do corpo humano, o toque de um condutor, como o dedo de uma mão, causa uma mudança do campo eletrostático do material, permitindo que o local de um ou mais toques seja mensurado. Esse princípio complexo faz com que seu custo seja mais elevado, além da fragilidade em relação a

ambientes úmidos e a necessidade de um microcontrolador (BESSIN, 2017; GUDINO, 2018).

Já a construção de uma tela sensível ao toque resistiva é composta de duas camadas transparentes cobertas com material condutor, postas uma sobre a outra. Idealmente, elas possuem resistividade uniforme e normalmente são espaçadas por um isolante. Então a camada superior faz contato com a camada inferior quando há pressão suficiente aplicada sobre a tela, seja com um dedo ou com uma caneta. Nesse ponto de contato, a camada inferior divide a camada superior no equivalente a duas resistências em série e vice-versa. A Figura 20 mostra a construção de uma tela sensível ao toque resistiva de quatro linhas (TEXAS ..., 2010; NXP SEMICONDUCTORS, 2008).

Figura 20 – Construção de uma tela sensível ao toque resistiva de quatro linhas

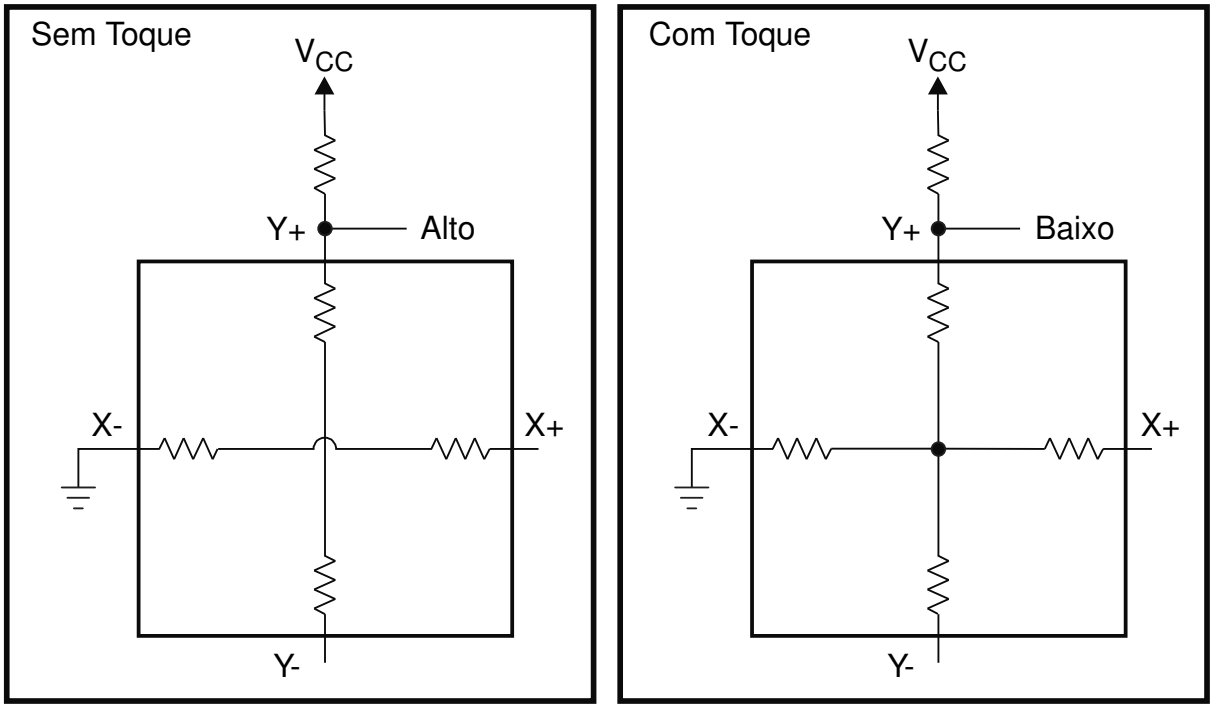


Fonte: traduzido de *Texas Instruments Inc.* (2010, p. 3).

Ao se aplicar uma tensão através de uma das camadas, um divisor de tensão é criado. Assim, as coordenadas do toque podem ser descobertas ao ler a tensão nesse divisor resistivo, ao aplicar uma tensão sobre a camada na direção do eixo coordenado Y e ler a tensão sobre esse divisor de tensão para achar a coordenada vertical. De maneira análoga, ao aplicar uma tensão sobre a camada na direção do eixo coordenado X e ler a tensão sobre o divisor de tensão, encontra-se a coordenada horizontal (TEXAS ..., 2010; NXP SEMICONDUCTORS, 2008).

Antes de realizar qualquer leitura sobre o divisor de tensão, é necessário verificar se a tela está realmente sendo tocada ou não. Isso pode ser feito ao aplicar uma tensão positiva ao pino Y positivo (Y+), com o uso de de um resistor de *pull-up*, enquanto se aterra o pino X negativo (X-). Tal resistor de *pull-up* deve ter resistência significativamente maior do que a resistência total da tela, normalmente por algumas centenas de ohms. Quando não há toque, o pino positivo se mantém na tensão aplicada, caso contrário sua tensão irá para nível lógico baixo. A Figura 21 mostra isso (TEXAS ..., 2010).

Figura 21 – Detecção de toque em uma tela sensível ao toque resistiva de quatro linhas



Fonte: traduzido de *Texas Instruments Inc.* (2010, p. 2).

Duas etapas são necessárias para extrair as coordenadas de um toque em uma tela sensível ao toque resistiva de quatro linhas. Primeiramente, Y+ é levado ao nível lógico alto e Y- ao baixo. Então a tensão em X+ é medida, por meio de um conversor analógico-digital (*Analog-Digital Converter* - ADC). A razão dessa tensão para a tensão do nível lógico alto é igual a razão da coordenada Y em relação a altura total da tela, fazendo com que a coordenada Y possa ser calculada por (12). A segunda etapa é determinar a coordenada X, que de maneira similar, é feita ao levar X+ ao nível lógico alto, X- ao baixo e medir Y+ por meio do ADC, fazendo com que ela seja calculada por (13) (TEXAS ..., 2010).

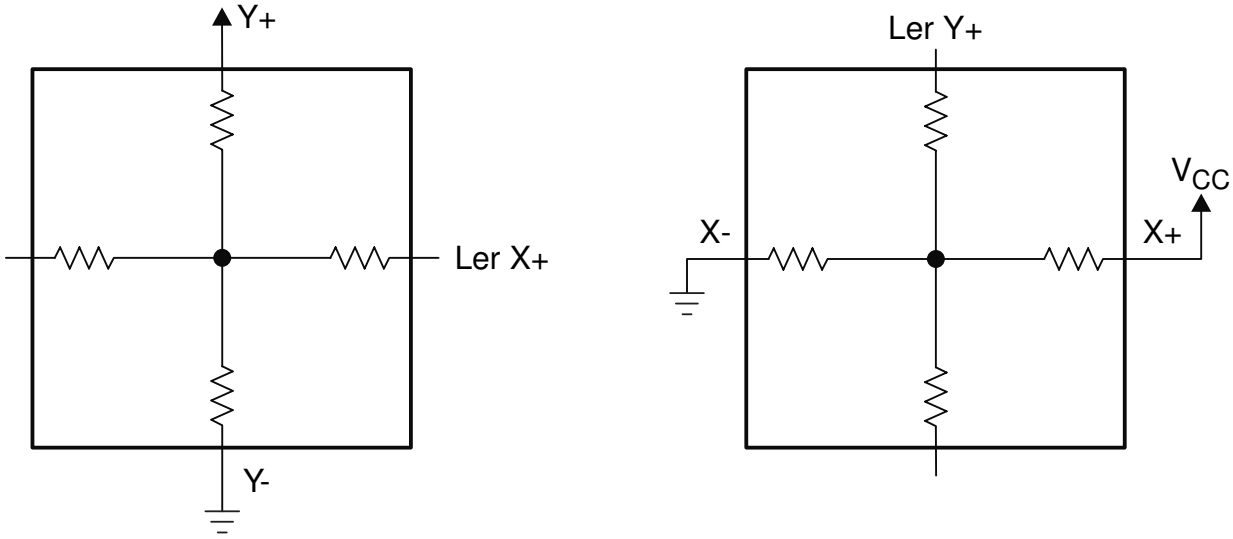
Nas equações, x e y são as coordenadas calculadas, V_{X+} e V_{Y+} são as tensões medidas nos divisores resistivos, V_{CC} é a tensão de nível lógico alto, h_{tela} é a

altura da tela em *pixels* e w_{tela} é a largura da tela em *pixels*. A Figura 22 ilustra o processo.

$$y = \frac{V_{X+}}{V_{CC}} \times h_{tela} \tag{12}$$

$$x = \frac{V_{Y+}}{V_{CC}} \times w_{tela} \tag{13}$$

Figura 22 – Processo de determinação das coordenadas de um toque em uma tela sensível ao toque resistiva de quatro linhas



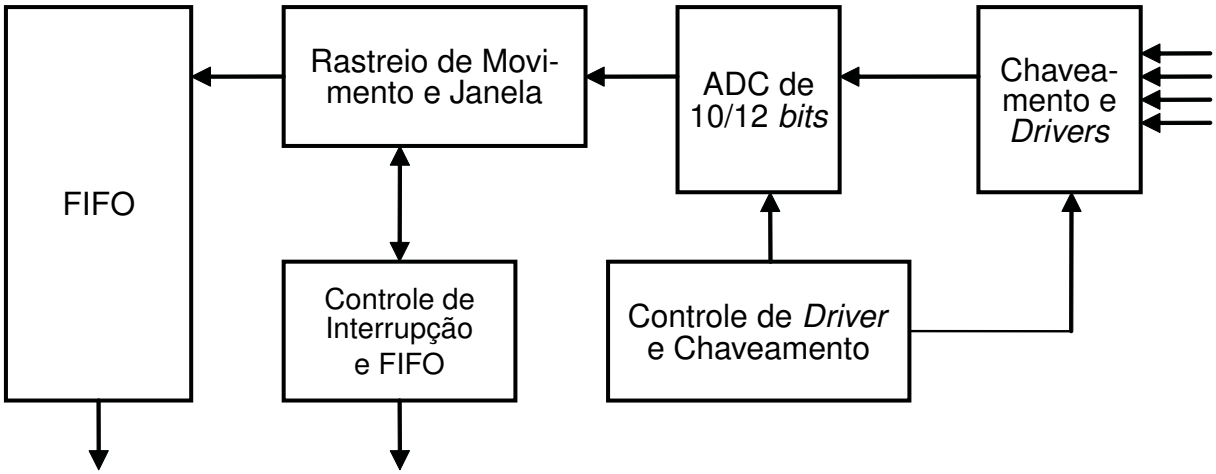
Fonte: traduzido de *Texas Instruments Inc.* (2010, p. 3).

2.5.4.2 CONTROLADOR

As características de controle de tela sensível ao toque do STMPE811 incluem seu funcionamento autônomo, o que diminui a carga de processamento do microcontrolador, interrompendo-o na ocorrência de eventos pré-definidos. O dispositivo também inclui um algoritmo de rastreamento de movimentos, a fim de impedir excesso de dados. Além disso, há um *buffer* FIFO de 128 posições e a possibilidade de limitar a área ativa onde os toques são considerados. O diagrama de blocos do controlador se encontra na Figura 23 (STMICROELECTRONICS, 2011b).

O controlador faz uso do ADC, cuja saída é convertida nas coordenadas de toque. Esses dados então são armazenados no *buffer* FIFO para que o sistema hospedeiro faça acesso a eles conforme necessário, assim reduzindo a utilização do barramento I2C para a transferência de dados (STMICROELECTRONICS, 2011b).

Figura 23 – Diagrama de blocos do controlador STMPE811



Fonte: traduzido de *STMicroelectronics* (2011, p. 34).

A memória FIFO tem tamanho suficiente para armazenar 128 coordenadas na resolução máxima do ADC, em que 24 (2×12) *bits* são utilizados por coordenada. No evento de detecção de toque, os dados são convertidos e armazenados automaticamente e possui registradores de estado e controle para verificação durante sua operação (STMICROELECTRONICS, 2011b). O toque na superfície pode ser checado tanto por *polling* – por meio dos registradores mencionados – quanto por interrupção.

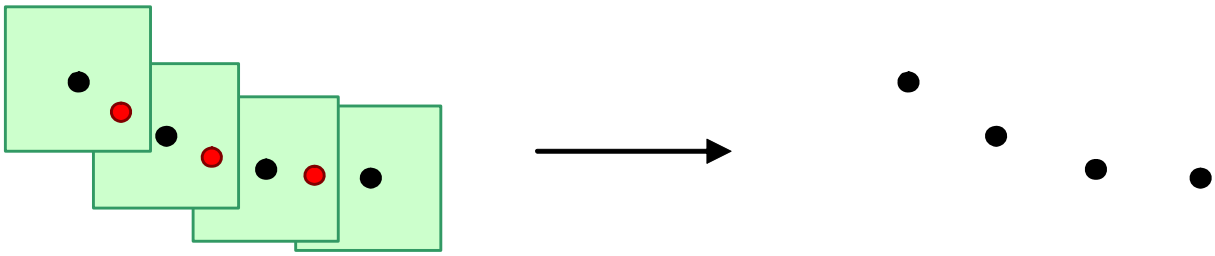
O bloco de *driver* e controle de chaveamento controlam o ADC e o multiplexador presentes no dispositivo, permitindo que um fluxo de dados seja produzido a uma frequência selecionada. Também pode-se configurar o dispositivo para limitar-se em duas correntes de operação – 20 ou 50 mA. Isso se faz necessário caso haja múltiplos toques na tela que causam um curto circuito (STMICROELECTRONICS, 2011a).

Além das coordenadas X e Y, esse controlador também permite que a informação de pressão no toque seja calculada, denotado como coordenada Z. Com isso, os seguintes modos de aquisição de dados estão disponíveis: aquisição de X, Y e Z, aquisição de X e Y ou somente X, Y e Z separadamente (STMICROELECTRONICS, 2011a).

O rastreio de movimento tem a intenção de reduzir a quantidade de dados gerada pelo dispositivo, definindo um critério para a geração desses dados. Oito opções estão disponíveis: 0, 4, 8, 16, 32, 64, 92 e 127, denotados índices de rastreio. Esses números referem-se aos pontos de resolução nativa do ADC, que a 12 *bits* é 4096. No modo XY, se a opção 127 estiver definida, por exemplo, dados novos só serão aceitos se eles estiverem fora de um quadrado de 127 pontos de tamanho, a partir do centro

do quadrado. A Figura 24 mostra esse comportamento (STMICROELECTRONICS, 2011b).

Figura 24 – Funcionamento do índice de rastreio para detecção de movimento



Fonte: *STMicroelectronics* (2011, p. 4).

Já no modo XYZ, novos dados serão considerados caso a pressão atual seja maior que a anterior, mesmo que as coordenadas X e Y estejam dentro do quadrado especificado anteriormente (STMICROELECTRONICS, 2011b).

A fim de otimizar o cancelamento de ruído, pode-se optar por fazer a média de várias amostras para produzir uma coordenada. As opções disponíveis são de 1, 2, 4 e 8 amostras para cálculo da média (STMICROELECTRONICS, 2011b).

Também é permitido definir uma área dentro da tela na qual os dados de toque são validos. Caso ocorra um toque fora dessa área, os dados são descartados. Duas coordenadas precisam ser escritas nos registradores para definir uma área, que são os pontos superior direito e inferior esquerdo (STMICROELECTRONICS, 2011b).

Assim termina a explanação dos dispositivos envolvidos na arquitetura de uma tela TFT-LCD, usando como base o *kit* de avaliação STM32F429I-DISCO. A partir disso, a próxima seção visa estabelecer uma comparação entre o uso de telas de LCD de caracteres alfanuméricos e a tecnologia abordada neste trabalho, para determinar o aumento de complexidade envolvido na migração de uma a outra.

2.6 COMPLEXIDADE DE MIGRAÇÃO

Uma das maneiras mais simples e utilizadas para exibir informações a partir de um sistema microcontrolado usando a tecnologia de LCD é por meio do uso de telas alfanuméricas baseada em caracteres. Essas telas utilizam matrizes de pontos para a exibição de informações e são monocromáticas. A maioria desses módulos de baixo custo usam o controlador Hitachi HD44780 ou similares, podendo exibir vários caracteres em várias linhas.

Dependendo do tipo de informação a ser exibida, as telas de LCD baseadas em caracteres podem limitar o conteúdo a ser exibido, visto que esses podem apenas exibir símbolos.

Caso o custo de migração para telas gráficas coloridas seja inviável, uma solução intermediária é a utilização de telas LCD gráficas monocromáticas. Além de permitir a escrita de caracteres alfanuméricos assim como os módulos com controladores baseados no Hitachi HD44780, é possível desenhar *pixels* em posições arbitrárias da tela, tal qual ocorre também nas telas de TFT-LCD, com a diferença que os *pixels* são definidos por *bits*, visto que a tela é monocromática. Tais módulos LCD utilizam como base o controlador Sitronix ST7920 para o comando da tela, normalmente disponibilizados em duas resoluções, sendo estas 64x64 e 128x64 *pixels*.

Para realizar a migração para dispositivos que fazem uso de telas gráficas coloridas para exibição de informações, como a TFT-LCD, não existem somente aumento no uso de memória, mas também na capacidade de processamento, visto a quantidade de sinais necessários para o uso destes dispositivos. Além disso, Devnath (2014) aponta a necessidade do uso de um sistema operacional de tempo real (*Real Time Operating System* - RTOS) e de um pacote gráfico. Esta seção será dividida em três partes, sendo analisados o consumo de memória, a carga de processamento e o consumo de energia.

2.6.1 MEMÓRIA

Uma das características do HD44780 é o uso de memórias para armazenamento de caracteres e dados de exibição. A memória RAM de dados da tela (*Display Data Random Access Memory* - DDRAM) armazena os dados da tela em caracteres de 8 *bits*, dispensando o armazenamento no formato de *pixels* (HITACHI, LTD., 1998). Isso faz com que apenas seja necessário armazenar no microcontrolador os caracteres a serem utilizados na tela, em código ASCII, para que sejam transferidos à DDRAM, caso não se use caracteres personalizados, que estão presentes na memória RAM do gerador de caracteres (*Character Generator Random Access Memory* - CGRAM).

Caso seja necessário usar todos os caracteres disponíveis para a CGRAM, que no caso do controlador HD44780 permite armazenar até oito caracteres de 5x8 pontos e quatro caracteres de 5x10 pontos e a matriz de pontos é armazenada em formato binário, tem-se um uso de memória de 520 *bits*, visto que os oito caracteres 5x8 tomam 320 *bits* e os quatro 5x10, 200 *bits* (HITACHI, LTD., 1998).

Já para a utilização de módulos baseados no controlador ST7920, será levado em conta a sua utilização em modo gráfico, em que os dados dos *pixels* são armazenados na memória do controlador denominada memória RAM de exibição gráfica (*Graphic Display Random Access Memory* - GDRAM). Esse controlador possui memória de 64 posições de 256 *bits*, totalizando 2.048 *bytes* ou 2 *kilobytes*, o que permite os módulos utilizando o ST7920 possuam uma resolução monocromática de até 64x256 *pixels* (SITRONIX ..., 2002).

Porém, a maioria dos módulos vêm com telas de resolução 64x64 e 64x128 *pixels*, o que faz com que, caso seja necessário armazenar no microcontrolador o conteúdo de uma tela, tal qual um *frame buffer*, a quantidade de memória necessária seja de 512 e 1.024 *bytes*, respectivamente.

Já para o uso de uma tela de TFT-LCD, além do uso das primitivas gráficas, que tem uso de memória dependente da profundidade de *bits* do *pixel*, há a necessidade de armazenar todo o conteúdo a ser transferido para a tela por meio do *frame buffer*. Visto na Seção 2.1.2, o uso de memória do *frame buffer* para o *kit* de desenvolvimento utilizado neste trabalho utiliza 150 KB, o que por si só já tem uma magnitude significativamente maior do que os módulos discutidos anteriormente.

Na Seção 2.5.2 foi visto que o formato interno de profundidade de cores de *pixel* é maior do que a exibida na tela, o que contribui ainda mais para o aumento de uso de memória. Além disso, vários tipos de memória são utilizados, como a SDRAM para o armazenamento do *frame buffer* e a memória *flash* para armazenamento de primitivas gráficas.

Por fim, não só tem-se a necessidade de armazenar as imagens a serem utilizadas, mas também o uso do RTOS e de bibliotecas gráficas, que fazem o uso de memória aumentar para centenas de *kilobytes* ao realizar esta migração.

2.6.2 PROCESSAMENTO

Devido ao aumento do uso de memória, é de se esperar que também haja um aumento na carga de processamento, visto que as primitivas usadas para a apresentação de dados necessitam ser transferidas pelo barramento que faz a comunicação entre o periférico e o microcontrolador.

As telas alfanuméricas fazem uso de um barramento de dados de no máximo oito *bits*, com alguns sinais auxiliares, com o mesmo se aplicando aos módulos com tela gráfica monocromática. Além disso, o envio dos dados respeita um tempo mínimo. Já as telas de TFT-LCD necessitam *clocks* em frequências específicas, na casa de MHz, para que a tela funcione em uma frequência de atualização específica. Com

isso, vê-se um aumento significativo na frequência em que o microcontrolador precisa funcionar, visto que em uma aplicação normalmente há várias tarefas ocorrendo ao mesmo tempo.

2.6.3 CONSUMO DE ENERGIA

Outra característica importante dessa migração é o consumo de energia, pois dependendo da quantidade de corrente que o dispositivo usa, pode ser necessário utilizar alimentação externa ao microcontrolador, o que pode impactar o projeto significativamente. Levando em consideração que os controladores em si não são responsáveis por todo o consumo de energia do dispositivo, retira-se informações do módulo LCD em si. Na Tabela 6 tem-se as especificações de características elétricas para o módulo LCD modelo JHD659.

Tabela 6 – Características elétricas de operação do módulo LCD JHD659

Característica	Símbolo	Condição	Mín.	Típico	Máx.	Unidade
Tensão de Operação	V_{DD}	-	4,5	-	5,5	V
Corrente de Operação	I_{DD}	Oscilação interna ou *clock* externo (V_{DD} = 5,0 V, f_{osc} = 270 KHz)	-	0,35	0,6	mA

Fonte: adaptado de *Jing Handa Electronics* (2007, p. 6).

A potência dissipada pode ser calculada multiplicando a tensão sobre ele pela corrente. Visto que na linha da corrente de operação é especificado a tensão utilizada para a medição, tem-se a máxima potência dissipada dada pela Equação (14).

$$P = 5{,}0 \times 0{,}6 \times 10^{-3} = 3 \text{ mW} \tag{14}$$

De forma análoga, a Tabela 7 mostra as características elétricas do controlador ST7920.

Tabela 7 – Características elétricas de operação do controlador ST7920

Característica	Símbolo	Condição	Mín.	Típico	Máx.	Unidade
Tensão de Operação	V_{DD}	-	2,7	-	5,5	V
Corrente de Operação	I_{DD}	f_{osc} = 530 KHz, V_{DD} = 3,0 V, R_f = 18 kΩ	-	0,20	0,45	mA

Fonte: adaptado de *Sitronix Technology Corp.* (2002, p. 32).

Assim como em (14), utiliza-se os valores da condição de teste para realizar o cálculo da potência, dada pela Equação (15).

$$P = 3{,}0 \times 0{,}2 \times 10^{-3} = 0{,}6 \text{ mW} \tag{15}$$

Do mesmo modo, na Tabela 8, observa-se as especificações de características elétricas do módulo TFT-LCD do *kit* de avaliação.

Tabela 8 – Características elétricas de operação do módulo LCD SF-TC240T-9370A-T

Item	Símbolo	Mín.	Típico	Máx.	Unidade	Condição de Teste
Tensão de Operação	V_{DD}	2,6	2,8	3,3	V	-
Corrente de Operação	I_{DD}	-	-	5	mA	V_{DD} = 2,8 V e T_a = 25 ℃

Fonte: adaptado de *Saef Technology Limited* (2012, p. 8).

Visto que a tensão utilizada no teste de corrente também é especificado, tem-se a potência dissipada pela Equação (16).

$$P = 2{,}8 \times 5 \times 10^{-3} = 14 \text{ mW} \tag{16}$$

O que, comparado ao módulo alfanumérico, representa um aumento de potência dissipada em quase cinco vezes. É importante ressaltar que essas características não incluem a utilização de retroiluminação em ambos os casos, que podem aumentar significativamente essa quantidade. Considerando a Tabela 9 com os dados da

retroiluminação do módulo TFT-LCD, pode-se observar de antemão uma corrente necessária consideravelmente maior para esse recurso.

Tabela 9 – Características elétricas de operação da retroiluminação LED do módulo LCD SF-TC240T-9370A-T

Item	Símbolo	Mín.	Típico	Máx	Unidade	Condição de Teste
Tensão de Operação	V_f	3,0	3,2	3,4	V	I_f = 60 mA
Corrente de Operação	I_f	-	60	-	mA	-

Fonte: adaptado de *Saef Technology Limited* (2012, p. 9).

E realizando o cálculo de potência, com a tensão de alimentação a 3,2 V, tem-se a Equação (17).

$$P = 3{,}2 \times 60 \times 10^{-3} = 192 \text{ mW} \tag{17}$$

Em que se observa uma dissipação de potência mais de dez vezes maior somente em retroiluminação.

2.6.4 CONSIDERAÇÕES FINAIS

Pode-se observar que, mesmo no cenário de uma migração de telas de matriz de ponto alfanuméricas para uma tela gráfica monocromática, existe um aumento significativo em armazenamento que, consequentemente, resulta em um aumento de processamento. Este aumento é ainda mais vertiginoso quando a migração é para as telas de TFT-LCD coloridas, visto que há o incremento não só para armazenar um *pixel*, o que é proporcional a quantidade de *bits* utilizada por *pixel*, mas também devido as maiores resoluções encontradas nas telas de TFT-LCD.

Por fim, outra característica que demanda atenção devido ao acréscimo considerável é o de consumo de energia, em que observou-se um alto valor ocorrendo em grande parte por conta da retroiluminação dos módulos. Isso são características que devem ter a devida atenção não só por conta da preocupação em utilizar menos energia, mas também devido aos limites de dissipação de potência que o microcontrolador em que esses se encontram conectados pode possuir.

3 DESENVOLVIMENTO

Antes de realizar a comparação com bibliotecas gráficas disponíveis, o método de trabalho será discutido e uma biblioteca gráfica minimalista foi desenvolvida neste trabalho com o intuito de comparar sua ocupação de memória e desempenho com bibliotecas gráficas disponíveis comercialmente ou gratuitamente. Este capítulo tem o objetivo de explicitar quais foram os princípios tomados para chegar a uma biblioteca gráfica com quatro elementos básicos disponíveis para uso: botão, caixa de seleção (*checkbox*), texto e seletor de valor deslizante (*slider*).

3.1 MÉTODO

Nesta seção é apresentado como foi realizado o trabalho, quais componentes foram utilizados e os passos tomados com eles.

3.1.1 APRESENTAÇÃO DO SISTEMA

O sistema envolvido na análise de arquitetura deste trabalho consiste de um sub-sistema gráfico, presente em sistemas microcontrolados e embarcados, responsável por exibir informações ao usuário por meio de uma tela de TFT-LCD colorida. Na Figura 25 tem-se uma visão básica desse sub-sistema.

Figura 25 – Diagrama básico de um sub-sistema gráfico

Fonte: autoria própria.

O microcontrolador é responsável por realizar os cálculos necessários do que necessita ser exibido a tela. Com isso, o sistema de controle da tela recebe esses dados, converte eles no formato apropriado para que então os sinais corretos sejam enviados a tela, que exibe a imagem na forma de uma matriz de *pixels*.

3.1.2 MATERIAIS

O microcontrolador a ser utilizado encontra-se embutido em um *kit* de avaliação, nesse caso, o *kit* escolhido foi o modelo STM32F429I-DISCO da ST Microelec-

tronics. A tela incorporada a esse modelo fornece um ponto de partida para a procura sobre a arquitetura de conexão do microcontrolador com a tela de TFT-LCD.

O ambiente de desenvolvimento (*Integrated Development Environment* - IDE) escolhido para o desenvolvimento do *firmware* a ser codificado para esse microcontrolador é o *Eclipse*, em sua versão 4.5.2 de codinome *Mars*. Junto com essa IDE, é utilizado o *plugin System Workbench for STM32*, que permite o desenvolvimento e depuração de códigos para os microcontroladores da série STM32. O *plugin* em questão é desenvolvido pela *Ac6 Tools* e a linguagem utilizada para o código é o C.

Por fim, também é usado o *software STM32CubeMX*, da ST Microelectronics, uma ferramenta de configuração de *software*, que permite a geração de códigos de inicialização em C por meio de assistentes gráficos. Essa ferramenta é utilizada para gerar os códigos iniciais para o microcontrolador e proceder com o desenvolvimento a partir delas.

3.1.3 MÉTODO

Primeiramente foi necessária uma revisão bibliográfica para realizar o estudo acerca da operação, de modo geral, do controle de uma tela de TFT-LCD por um microcontrolador. Partindo dessas informações, o estudo aprofundou-se ao analisar detalhadamente a arquitetura de conexão entre a tela e o microcontrolador presentes no *kit* de avaliação STM32F429I-DISCO, explanando o *hardware* presente e a interação que permite o controle mencionado anteriormente.

Visto que há um *kit* de avaliação envolvendo um microcontrolador específico, foi indispensável a utilização de material disponibilizado pelo fabricante, como folhas de dados (*datasheets*) do *hardware* envolvido e notas de aplicação.

Uma vez elucidado o funcionamento básico do interfaceamento entre a tela e o microcontrolador, também foi realizada a comparação analítica com o interfaceamento de um *display* monocromático de matriz de pontos. Isso possui o intuito de ilustrar a diferença de complexidade envolvida ao migrar de um tipo de tela para outra.

Completada a parte de estudo de arquitetura, iniciou-se a implementação básica do *driver* para o microcontrolador supracitado. Para isso, foi necessário estudar um método de implementação de *driver* em um RTOS, visto que seu uso para a utilização de bibliotecas gráficas é indispensável, já que é comum que aplicações que façam uso de bibliotecas gráficas tenham que lidar com várias tarefas executando concorrentemente.

Com o *driver* em funcionamento, o próximo passo foi a implementação de uma interface gráfica sobre o *driver*, permitindo a confecção de uma biblioteca básica para aplicações gráficas.

Sucedendo o funcionamento da biblioteca, o próximo passo consistiu comparação entre algumas soluções existentes no mercado. Foram escolhidas duas opções para realizar esta comparação, uma paga e uma gratuita. Para a biblioteca paga, foi utilizada a *Embedded Wizard*, porém em sua versão de testes, eliminando o custo de comprá-la. No lado gratuito, foi utilizada a biblioteca livre e de código aberto *LittlevGL*, o qual se caracteriza por possuir baixo uso de memória e permitir elementos gráficos fáceis de utilizar e com suporte a funcionalidades avançadas, como animações e *anti-aliasing*.

Essa comparação se dará por meio de testes de desempenho e monitoramento do uso de recursos pelas bibliotecas, além de uma análise subjetiva da facilidade de uso com o *kit* de avaliação em questão neste trabalho.

3.2 CONFIGURAÇÃO DE *HARDWARE*

Primeiramente, foi feita a utilização do *software STM32CubeMX*, conforme mencionado na Seção 3.1.2, para gerar os códigos iniciais de desenvolvimento. Esses códigos são de inicialização do RTOS, definição dos valores de *clock* do sistema e quais periféricos serão inicializados para utilização posterior.

O RTOS utilizado foi o *FreeRTOS* – por conta do suporte que esse já possui pela fabricante do microcontrolador, além do uso e configuração já presentes no *software* – e essa sequência de geração de código inicial foi seguida para a utilização de todas as bibliotecas gráficas envolvidas no trabalho, visto que elas definem a frequência em que o processador operará, além da frequência dos *clocks* envolvidos no uso da tela.

De forma resumida, para todos os casos, a frequência de operação principal do microcontrolador é de 168 MHz, com o *clock* da tela definido em 6 MHz, o qual está próximo com o valor do *clock* do *pixel* calculado na Seção 2.4.4.

Como ponto de partida, foi utilizada a *Board Support Package* (BSP), um conjunto de APIs disponibilizada para cada modelo do *kit* de desenvolvimento com o intuito de facilitar a inicialização e uso do *hardware* presente. Uma das partes que compõe esse conjunto é a de componentes, com *drivers* relacionados aos componentes externos ao do *kit*, como o módulo ILI9341, o componente de interação via toque na tela e a memória SDRAM (STMICROELECTRONICS, 2019). As bibliotecas

gráficas envolvidas neste trabalho utilizam este conjunto de *drivers* para inicializar o módulo no caso deste *kit* de desenvolvimento, então o mesmo método foi utilizado para a implementação desta.

O módulo é inicializado utilizando comunicação serial para o envio dos comandos de inicialização, que definem a quantidade de linhas e colunas que serão exibidas, frequência de atualização, ordem de envio e apresentação dos *pixels* entre outras várias características. Uma delas é a interface utilizada para o envio dos dados a serem exibidos na tela. Seguindo a metodologia utilizada na BSP e também nas outras bibliotecas, a interface paralela em 16 *bits* de profundidade de cor foi utilizada, visto que no *kit* de avaliação os componentes estão conectados para utilização desta maneira, como visto na Figura 12. O uso do periférico LTDC é empregado junto com essa interface, com o *frame buffer* sendo armazenado na memória SDRAM presente no *kit*. Assim, o LTDC envia os dados do *frame buffer* para a memória do módulo utilizando a interface paralela, bastando a biblioteca gráfica apenas manipular os dados no espaço de memória na SDRAM para que as alterações sejam exibidas na tela.

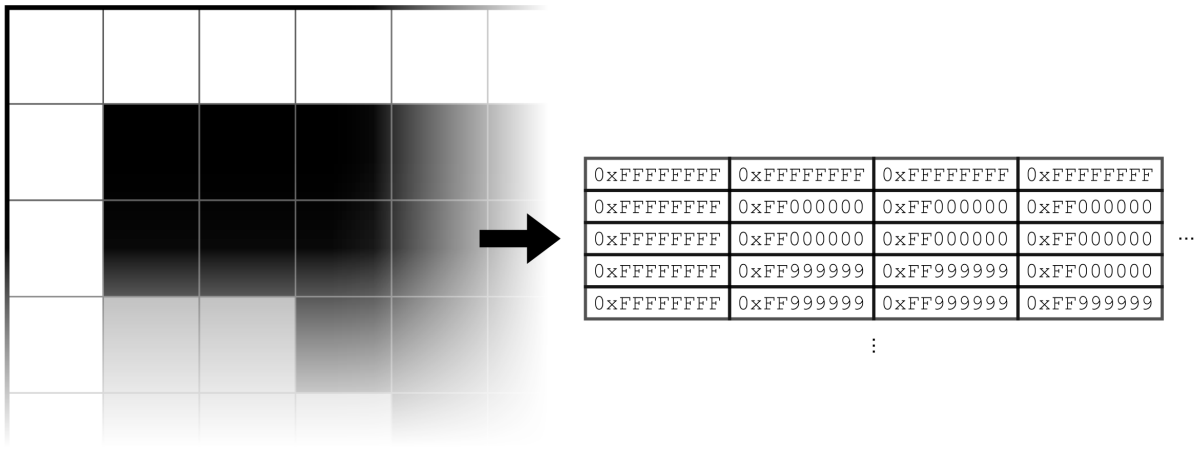
3.3 OPERAÇÕES DE DESENHO

Uma vez funcional a comunicação entre o módulo ILI9341 e o periférico LTDC, pode-se escrever um valor na memória do *frame buffer* para que um *pixel* apareça na tela. O espaço de memória em que os *pixels* estão armazenados encontram-se no formato ARGB8888, o qual possui profundidade de 32 *bits*. A Figura 26 ilustra como o que está atualmente sendo exibido na tela, à esquerda, é representado em valores armazenados no *frame buffer*, à direita. Os valores em hexadecimal representam *pixels* no formato mencionado.

Apesar de haver diferença na quantidade de *bits* por *pixel* do *framebuffer* para a memória do módulo, a conversão para o formato apropriado é realizado pelo LTDC antes da transferência de dados, como foi explicitado em mais detalhes na Figura 15 da Seção 2.5.2.

Tendo em vista a habilidade de colocar um *pixel* no *frame buffer*, uma das primeiras tarefas a ser desempenhada durante a inicialização da biblioteca é preencher o espaço com o que seria considerado uma tela vazia. Isso é necessário pois quando o dispositivo é alimentado, o conteúdo de sua memória não é conhecido, podendo conter quaisquer resquícios de uma utilização anterior, como por exemplo, o último quadro que foi exibido antes do desligamento. Ao preencher todo o espaço de memória do *frame buffer*, não somente se torna conhecido seu conteúdo, como tam-

Figura 26 – Representação dos *pixels* na tela, a esquerda, correspondente aos seus valores armazenados no *framebuffer*, a direita



Fonte: autoria própria.

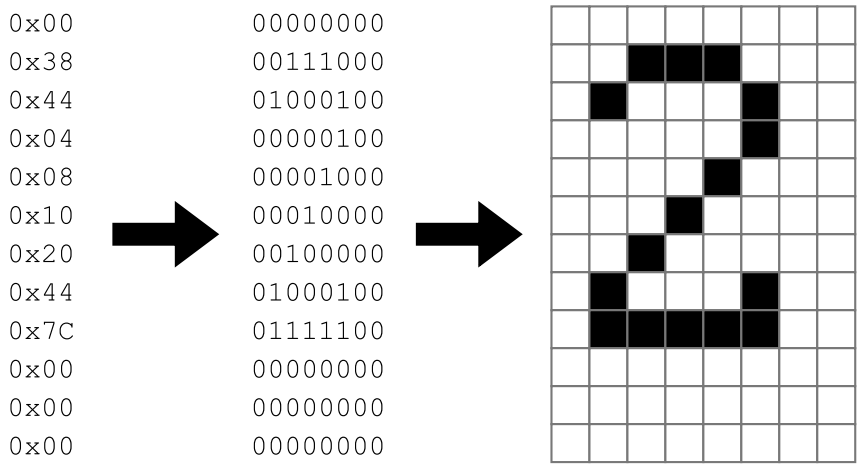
bém o prepara para a sua utilização. Preenchendo todo o espaço de memória com zeros, por exemplo, tem como resultado uma tela com a cor de fundo preta, no formato ARGB8888. Porém, pode ser que, com o intuito de permitir a personalização da cor de fundo, escolheu-se a abordagem de permitir que se defina uma cor de fundo para preencher o *frame buffer* e inicializar o conteúdo gráfico onde as operações de desenho serão realizadas.

As operações de desenho envolvem a criação de formas primitivas que então são usadas para construir elementos mais complexos. As formas primitivas têm em comum a utilização da função de inserção de um *pixel*. Assim, foram desenvolvidas funções para desenhar as seguintes formas: linha, contorno de um retângulo, retângulo preenchido, contorno de um círculo, círculo preenchido, caractere e conjunto de caracteres (denominado *string*).

Para o desenho de caracteres, precisou-se fazer uso de uma fonte. Essa necessidade foi suprida com a BSP, que possui arquivos definindo uma fonte de quatro tamanhos diferentes por meio de um vetor de valores hexadecimais. Cada *bit* de um desses valores define um *pixel*, sendo que quando esse *bit* está alto, o *pixel* será escrito à tela, não ocorrendo caso contrário. Isso faz com que os caracteres sejam desenhados com a profundidade de 1 BPP.

Para ilustrar essa representação, tem-se a Figura 27, mostrando como os valores hexadecimais da fonte armazenada se traduzem em um caractere desenhado na tela. À esquerda tem-se os valores hexadecimais, ao meio os mesmos são mostrados em binário por conveniência e a direita, a representação gráfica.

Figura 27 – Exemplo do caractere “2” armazenado em memória como uma série de valores, seguido pela representação dos mesmos em binário e, por fim, o resultado após ser desenhado na tela



Fonte: autoria própria.

Conforme explicitado na Seção 2.5.3, o *kit* de desenvolvimento possui um periférico para aceleração gráfica, o qual é um periférico DMA com foco em operações de imagem, como conversão de formato de *pixel* durante as transferências. Sua intenção é de acelerar as transferências de dados, diminuindo a carga de processamento da CPU. As operações de preenchimento de toda a tela e de retângulo foram escolhidas para utilizarem a DMA2D, visto que essas se encaixam no modo de operação registrador para memória, já que a intenção desse é para o preenchimento de retângulos (STMICROELECTRONICS, 2017c).

3.4 ELEMENTOS DE INTERAÇÃO COM O USUÁRIO

Com as operações de desenho definidas e implementadas, buscou-se escolher um conjunto de elementos de interação básicos para serem construídos utilizando essas operações. Foram escolhidos quatro desses elementos para serem disponibilizados nessa biblioteca:

- **Botão:** indica uma ação que ocorre ao ato de pressionar sobre ele, normalmente rotulado por um texto, ícone ou ambos (U.S. DEPARTMENT ...,);
- ***Checkbox:*** permite o usuário selecionar uma ou mais opções de um conjunto (U.S. DEPARTMENT ...,);
- **Texto:** texto para a visualização de informações;

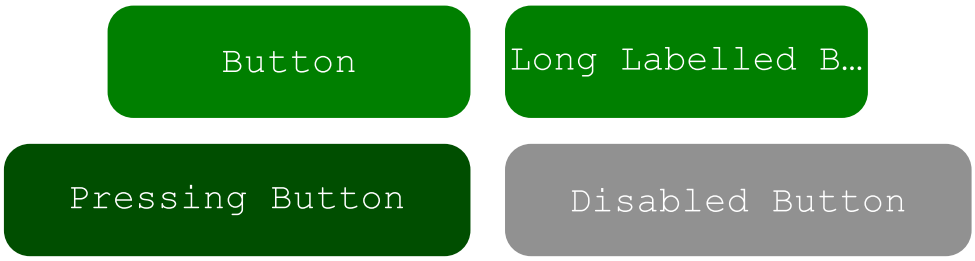
- ***Slider:*** elemento que permite o usuário definir ou ajustar um valor entre dois intervalos por meio de uma barra deslizante (U.S. DEPARTMENT ...,).

Esses elementos foram escolhidos por desempenharem as ações mais básicas para a visualização de informações e interação com o usuário. Incorporado a esses elementos está também a definição de propriedades e a possibilidade de alterar essas propriedades durante o tempo de execução do aplicativo. As propriedades de cada elemento estão definidas nas Seções 3.4.1 a 3.4.4.

3.4.1 BOTÃO

No contexto dessa biblioteca gráfica, botão é um elemento retangular preenchido com cantos arredondados, contendo um texto em seu interior que faça descrição da ação que irá ocorrer quando o usuário pressioná-lo.

Figura 28 – Alguns possíveis estados visuais do botão



Fonte: autoria própria.

A Figura 28 mostra visualmente o componente alguns possíveis estados desse, como a contenção de um texto mais longo que a largura, estado visual ao pressionar e ao desabilitar. As seguintes propriedades foram implementadas e são passíveis de serem modificadas:

- **Posição horizontal:** quantidade de *pixels* a partir da borda esquerda da tela, em direção horizontal, em que o botão começará a ser desenhado;
- **Posição vertical:** quantidade de *pixels* a partir da borda superior da tela, em direção vertical, em que o botão começará a ser desenhado;
- **Largura:** quantidade de *pixels* a partir da posição horizontal que será ocupado na tela e considerado para interação;
- **Altura:** quantidade de *pixels* a partir da posição vertical que será ocupado na tela e considerado para interação;

- **Cor de Fundo:** cor do elemento retangular preenchido, em formato ARGB8888;
- **Cor de Texto:** cor do texto descritivo, em formato ARGB8888;
- **Fonte:** fonte utilizada para o texto descritivo;
- **Texto:** conjunto de caracteres representando o texto descritivo;
- **Estado:** estado do botão, podendo ser um de três possíveis – **Desativado**, em que não responde a interações, **Ativado** e em **Clique**;
- **Ação de Clique:** função a ser executada quando se atinge o estado de clique;

Para o botão se encontrar em estado de clique, o usuário terá que estar pressionando sobre ele. Para prover um retorno visual de que o botão se encontra nesse estado e que a ação foi executada, a cor de fundo é escurecida.

3.4.2 *CHECKBOX*

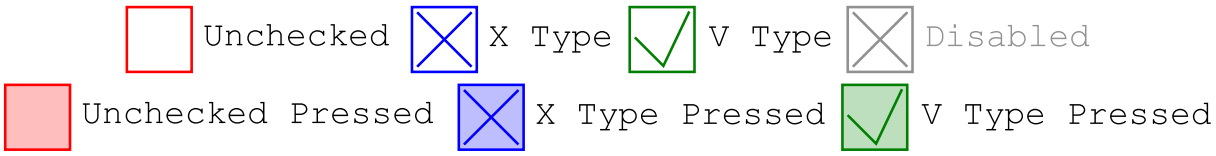
Aqui, o *checkbox* é representado por um quadrado contendo um marcador quando esse se encontra selecionado, seguido de um texto descritivo sobre o elemento que está sendo modificado. As seguintes propriedades foram implementadas:

- **Posição horizontal:** quantidade de *pixels* a partir da borda esquerda da tela, em direção horizontal, em que o *checkbox* começará a ser desenhado;
- **Posição vertical:** Quantidade de *pixels* a partir da borda superior da tela, em direção vertical, em que o *checkbox* começará a ser desenhado;
- **Tamanho:** quantidade de *pixels* para definir o tamanho do quadrado e, por consequência, o marcador que estará contido dentro desse;
- **Cor de Fundo:** cor do quadrado contendo o marcador em formato ARGB8888;
- **Cor do Marcador:** cor do marcador em formato ARGB8888;
- **Cor de Texto:** cor do texto descritivo em formato ARGB8888;
- **Fonte:** fonte utilizada para o texto descritivo;
- **Texto:** conjunto de caracteres representando o texto descritivo;
- **Tipo de Marcador:** define o tipo de marcador a ser utilizado, podendo ser dois possíveis – em “X” ou “V”;

- **Estado Funcional:** define se o marcador responde ou não a interações recebidas;
- **Estado de Marcação:** define se o marcador está inserido ou não no quadrado que o contém e se ele está sendo pressionado ou não pelo usuário;
- **Ação de Clique:** função a ser executada quando o *checkbox* é pressionado.

O elemento possui quatro estados possíveis de marcação: **Marcado**, **Desmarcado**, **Marcado e Pressionado** e **Desmarcado e Pressionado**. Para comunicar ao usuário que o *checkbox* foi pressionado e que a ação será executada ao soltar dele, a caixa de marcação é preenchida com a cor de fundo, porém escurecida.

Figura 29 – Estados visuais do *checkbox*



Fonte: autoria própria.

A Figura 29 ilustra esses estados e a comunicação com o usuário que a caixa está sendo pressionada, os dois tipos de marcador e o estado desabilitado em que esse componente pode se encontrar.

3.4.3 TEXTO

Figura 30 – Representação visual do texto e simbólica da fronteira que o contém

Fonte: autoria própria.

Esse elemento consiste de uma cadeia de caracteres que está contido dentro de um quadrado definindo suas fronteiras. Esse quadrado não é desenhado e é utilizado apenas para conter o texto e limitá-lo caso o seu conteúdo extrapole as fronteiras impostas. A Figura 30 ilustra como o texto pode estar contido dentro dessa fronteira de quatro maneiras diferentes. Com isso, as seguintes propriedades foram implementadas:

- **Posição horizontal inicial:** quantidade de *pixels* a partir da borda esquerda da tela, em direção horizontal, em que o texto começará a ser desenhado;

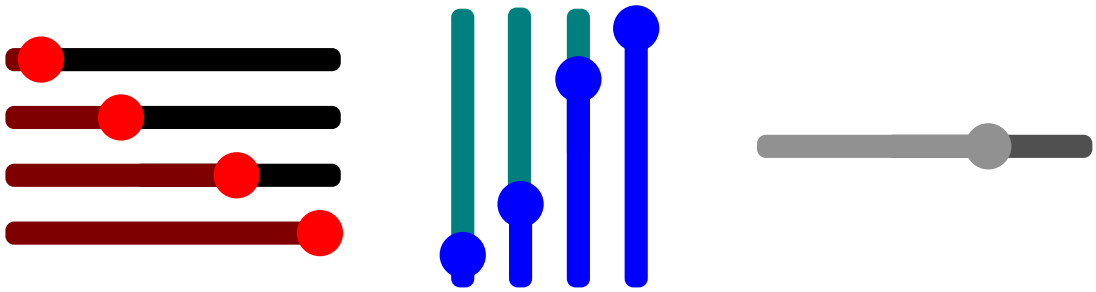
- **Posição vertical inicial:** quantidade de *pixels* a partir da borda superior da tela, em direção vertical, em que o texto começará a ser desenhado;
- **Posição horizontal final:** quantidade de *pixels* a partir da borda esquerda da tela, em direção horizontal, definindo o limite até onde o texto será desenhado;
- **Posição vertical final:** quantidade de *pixels* a partir da borda superior da tela, em direção vertical, definindo o limite até onde o texto será desenhado;
- **Cor:** cor do texto em formato ARGB8888;
- **Fonte:** fonte utilizada;
- **Texto:** o texto propriamente dito contido em um conjunto de caracteres;
- **Alinhamento Horizontal:** caso o texto ocupe menos espaço do que a fronteira que o contém, ele poderá ser alinhado em relação a essa, com três possibilidades – à esquerda, ao centro e à direita.

Quando o texto não cabe no quadrado que define sua fronteira, a parte não mostrada é retirada da exibição e reticências são concatenadas ao final da parte visível do conjunto.

3.4.4 *SLIDER*

A representação gráfica do *slider* ocorre com dois quadrados preenchidos de cantos arredondados de cores diferentes – um que representa os valores maiores do que o atual e outro, os menores.

Figura 31 – Estados visuais do *slider*



Fonte: autoria própria.

Esses quadrados se encontram alinhados dependendo do seu tipo e no encontro desses quadrados é colocado um círculo preenchido na posição representativa do valor atual, também servindo como ponto de interação para que o usuário possa arrastá-lo para algum dos lados. As seguintes propriedades foram implementadas:

- **Tipo:** define o tipo de *slider*, dentre dois – horizontal ou vertical. Caso horizontal, como no lado esquerdo da Figura 31 os retângulos serão posicionados um ao lado do outro. Na vertical, ilustrado ao centro da Figura 31, são posicionados um em cima do outro;
- **Posição horizontal:** quantidade de *pixels* a partir da borda esquerda da tela, em direção horizontal, em que o *slider* começará a ser desenhado;
- **Posição vertical:** quantidade de *pixels* a partir da borda superior da tela, em direção vertical, em que o *slider* começará a ser desenhado;
- **Tamanho:** quantidade de pixels que representará a escala do valor mínimo ao máximo. Quando o tipo é horizontal, isto definirá a largura do slider. Do contrário, definirá a altura;
- **Cor de Fundo:** cor em formato ARGB8888 do quadrado com os valores maiores que o atual;
- **Cor do *Slider*:** cor em formato ARGB888 do quadrado com os valores menores que o atual;
- **Valor Atual:** valor atual definido para o *slider*;
- **Valor Máximo:** maior que poderá ser definido;
- **Valor Mínimo:** menor que poderá ser definido;
- **Estado:** estado funcional do elemento, podendo ser **ativado** ou **desativado** – caso desativado, ele não responderá a interações. O estado **desativado** encontra-se ilustrado ao lado direito da Figura 31;
- **Ação de Troca de Valor:** ação a ser executada quando um valor diferente do atual é definido.

Quando um toque é detectado dentro do elemento, a posição do toque é utilizada para definir o novo valor. A partir disso, pode-se usar a ação de troca de valor para atualizar uma variável associada.

Para todos os elementos que possuem implementação de estado funcional, isto é, pode ser habilitado ou desabilitado, quando esses se encontram no estado desabilitado, suas cores definidas são trocadas por tons pré-definidos de cinza para comunicar ao usuário que aquele componente não responderá a interações.

3.5 CONTROLE DE OBJETOS E TELA

Para manter o controle de quais objetos estão na tela, foi criada uma estrutura de dados do tipo lista encadeada. Nela, não somente é mantido o endereço para o elemento em si, mas também suas coordenadas iniciais e finais onde serão considerados as interações e ponteiros para o elemento anterior e o próximo que foram inseridos nesta estrutura. Também foi necessária a criação de um *mutex* para o controle de inserção e exclusão desses elementos, visto que o RTOS possui várias *threads* em execução, sendo possível que essas funções podem ser executadas de *threads* distintas.

Para que um elemento seja inserido na tela, é necessário chamar sua função de criação, que recebe o elemento e suas propriedades para a inserção na lista encadeada. De modo análogo, para remover esse elemento, a função de remoção precisa ser executada.

Visto que a propriedade dos elementos pode ou não mudar entre uma atualização da tela e outra, foi criada uma propriedade de estado da tela, em que um de três estados pode ser assumido:

- **Não inicializado:** a biblioteca foi inicializada mas não a estrutura de dados. Seu conteúdo é desconhecido e consequentemente precisa ser inicializado para que se encontre em um estado conhecido;
- **Limpo:** todos os componentes estão atualizados e não é necessário redesenhar os componentes da tela;
- **Necessita Atualização:** um ou mais componentes passaram por mudanças e com isso se faz necessário o redesenho dos elementos.

Toda vez que um elemento tem uma de suas propriedades atualizadas, seja por interação do usuário ou por uma das funções que altera suas propriedades, o estado da tela é marcado para a necessidade de atualização. Assim, a próxima vez que a função de atualização da tela é chamada, sabe-se que os componentes necessitam ser redesenhados.

3.6 INICIALIZAÇÃO E ROTINA DE ATUALIZAÇÃO

Após a inicialização dos dispositivos necessários para o funcionamento da tela, discutidos na Seção 3.2, a inicialização da biblioteca primeiro é dada por limpar o conteúdo do *frame buffer*. Isto se faz necessário pois não é conhecido o que há dentro

dessa área de memória. Caso esse passo venha a ser ignorado, é possível que tanto artefatos aleatórios quanto o último conteúdo processado sejam mostrados. A Figura 32 ilustra um exemplo de como aparenta um *frame buffer* não inicializado.

Figura 32 – Capturas de imagem de *frame buffers* não inicializados

Fonte: elaborado pelo autor.

A partir disto, é necessário inicializar a estrutura de dados responsável pelo controle dos objetos na tela, visto na Seção 3.5. Como não há nenhum objeto nesse momento, a lista é inicializada vazia e o estado da tela muda de não inicializado para limpo, uma vez que não há nada a desenhar. Desse momento em diante, pode-se utilizar as funções de criação dos componentes e alterar suas propriedades conforme necessário.

Durante a inicialização do RTOS, é definida uma tarefa que fará o controle e atualização da tela, sendo executada junto com as outras tarefas do sistema. Essa tarefa, daqui em diante denominada tarefa de controle, primeiramente inicializa a função de toque do módulo da tela, por meio do conjunto de APIs BSP, mencionado na Seção 3.2. Uma vez inciada, três ações são executadas e após a conclusão, a tarefa é posta em um tempo de espera pré-definido, definido nesse caso em dez milissegundos, sendo um valor menor que o período de atualização da tela escolhido arbitrariamente. Essas três ações consistem em buscar os dados de estado do painel de toque, para então realizar a checagem dos objetos presentes de acordo com essas informações, finalizando com a atualização da tela propriamente dita, onde ocorre o redesenho dos componentes.

Assim como na inicialização do painel de toque, a BSP também é utilizada para obter os dados acerca da tela. Três informações são extraídas: se ocorreu ou não um toque no painel. Caso o último tenha ocorrido, as outras duas informações são as coordenadas horizontal e vertical deste evento, mensuradas em quantidade de *pixels* a partir do canto superior esquerdo da tela.

A partir desses dados é realizado a verificação dos objetos e alteração das propriedades pertinentes de acordo com as coordenadas do toque. Para isso, a tela tem que estar em um estado que não seja o de "não inicializado". Com isso, a lista encadeada é percorrida item a item e é verificado se as coordenadas do toque se encontram dentro do objeto. Caso positivo, as propriedades são alteradas de acordo com o objeto em questão. No caso do botão, será de acordo com o exposto na Seção 3.4.1, alterando a propriedade de estado. O mesmo se aplica para o *checkbox* na Seção 3.4.2, com o estado de marcação. O último elemento que é alterado com essas informações é o *slider*, mencionado na Seção 3.4.4, alterando-se o valor de acordo com a posição. Mesmo que não haja toque, ainda tem-se objetos que podem ter suas propriedades alteradas com a não ocorrência de um toque. Esses casos são dos objetos de botão e *checkbox*, que ocorre quando eles estão em estado de clique e o usuário deixa de tocar neles, necessitando fazê-los voltar ao estado anterior.

Tendo as propriedades pertinentes de cada objeto modificadas com as informações de toque, a tela é redesenhada. Para realizar o redesenho, primeiramente é verificado a propriedade de estado da tela. Assim, o redesenho só é realizado sob o estado de necessidade de atualização. Esse estado é definido toda vez que qualquer propriedade de qualquer componente da lista passa por modificações. Então, a lista de objetos é percorrida, primeiramente apagando a área do mesmo em questão com a cor de fundo definida. Depois, é executada a função de desenho do tipo de objeto em questão.

Buscando diminuir os efeitos de *tearing*, descrito na Seção 2.1.2, tentou-se aplicar a técnica de *double buffering*, utilizando do recurso de camadas presente no LTDC, o que permitiu que cada camada fosse um *frame buffer*. Assim, após percorrer a lista de objetos, a camada em que será desenhada a tela é trocada e o endereço do *frame buffer* a ser exibido na tela também. Porém, a troca não é feita de imediato. Utiliza-se a biblioteca HAL para definir que essa troca ocorra no intervalo em que dados de *pixel* não são enviados. Esse intervalo é denominado bloqueio vertical (*vertical blanking*), e consiste dos tempos VFP, V_{SYNC} e VBP expostos na Seção 2.4.2.

Por fim, já que todos os elementos foram atualizados, o estado da tela é passado para "limpo" e inicia-se a espera para que o processo se inicie novamente. Du-

rante esse intervalo, as outras tarefas de usuário podem alterar as propriedades dos objetos, os quais serão redesenhados no próximo ciclo, caso haja alguma alteração.

4 RESULTADOS

O capítulo de resultados tem por objetivo realizar uma comparação entre duas soluções disponíveis com modelos de licenciamentos distintos, sendo uma não gratuita de código fechado e outra gratuita de código aberto, quando considerado o quesito de uso comercial. Além disso, é explicitado o teste realizado entre essas duas soluções mais a que foi desenvolvida neste trabalho, no quesitos de uso de memória e processamento e os resultados obtidos a partir dos testes.

4.1 COMPARAÇÃO ENTRE BIBLIOTECAS GRÁFICAS

Conforme visto no Capítulo 1, duas bibliotecas gráficas foram escolhidas para realizar a comparação. A primeira a ser detalhada é a *Embedded Wizard*, desenvolvida pela empresa *TARA Systems*. Para utilização comercial, é uma solução paga e a distribuição de seu código é fechada, com o fornecimento de código compilado para seus utilizadores.

O que permitiu a utilização dessa solução foi a existência de uma versão gratuita. Enquanto todos os recursos estão presentes, o tamanho permitido do projeto é limitado, desde que não seja utilizado de forma comercial. O escopo deste trabalho ficou dentro dessa limitação de tamanho, o que permitiu realizar a comparação e os testes.

Para desenvolver projetos utilizando essa biblioteca, se utiliza um ambiente de desenvolvimento denominado *Embedded Wizard Studio*. Uma das características nesse ambiente é a independência de plataforma de *hardware*. Isso ocorre por possuir uma linguagem de orientação a objetos própria, denominada **Chora**, pois todo o código escrito nessa linguagem não está atrelado a nada específico de nenhum fabricante ou sistema operacional, podendo inclusive executar em sistemas embarcados sem fazer uso de sistema operacional, comumente chamado de *bare metal* (TARA SYSTEMS GMBH, 2019b).

Além disso, seu *software* segue um paradigma de programação assistida visualmente (*Visually Aided Programming*), em que componentes podem ser arrastados a tela e modificados de forma mais visual do que por código. Também é possível simular a interface gráfica desenvolvida pela biblioteca por meio do ambiente de desenvolvimento (TARA SYSTEMS GMBH, 2019c).

Para que tudo isso se traduza a uma interface executável em um sistema embarcado, a desenvolvedora oferece o que pode ser chamado de pacote de plataforma

(*Platform Package*), que é responsável por realizar a abstração da linguagem **Chora** e converter isso para código-fonte utilizável na plataforma de *hardware* de destino (TARA SYSTEMS GMBH, 2019a). A disponibilidade para plataformas de *hardware* depende do fabricante e da licença adquirida, visto que os pacotes de plataforma são disponibilizados pelo mesmo. Na versão gratuita e para pequenas empresas – que tem o custo de € 2990 –, sete plataformas estão disponíveis. Na versão de grandes empresas, mais de 60, com custo de € 5000 o *software*, mais € 2400 cada pacote (TARA SYSTEMS GMBH, 2017).

Por fim, a *Embedded Wizard* possui uma vasta quantidade de funcionalidades visuais, que vão desde o suporte a vários tipos de imagens e formatos de cores presentes em diversos sistemas embarcados até o uso de gráficos vetoriais e animações. Na versão gratuita, além de possuir limitação de projeto, um emblema de marca d'água é mostrado na tela por curtos intervalos de tempo, explicitando o uso de uma versão de avaliação do *software* (TARA SYSTEMS GMBH, 2019e).

A segunda biblioteca a ser detalhada é a *LittlevGL*, desenvolvida e mantida de forma independente. É uma biblioteca gráfica gratuita para qualquer tipo de uso. Não faz venda de nenhum produto ou suporte, que se mantém por meio de doações e oferece suporte por meio de um fórum no *site* oficial do projeto. Além disso, seu código é aberto e contribuições de usuários são aceitas (KISS-VÁMOSI, 2018a).

Para o desenvolvimento do projeto de interface gráfica, há duas opções. A primeira é a utilização do que é denominado *PC Simulator*, um *software* para simular a execução da biblioteca gráfica usando código da própria biblioteca, portável para o *hardware* alvo quando necessário. Para esse simulador, é utilizado a biblioteca *Simple DirectMedia Layer* (SDL) para simular um *display* TFT-LCD e um painel de toque. A segunda opção é utilizar diretamente a biblioteca no *hardware*, realizando chamadas de funções e verificando os resultados direto no dispositivo (KISS-VÁMOSI, 2018b).

Visto que esse projeto é de código aberto, a disponibilidade para integração com sistemas embarcados não depende apenas dos responsáveis pela biblioteca (KISS-VÁMOSI, 2018c). Apesar de alguns *ports* estarem disponíveis para algumas plataformas, é disponibilizado na documentação o que se faz necessário para que a biblioteca seja portada para qualquer plataforma, desde que use a linguagem C ou C++, escolhida para prover a maior compatibilidade possível com microcontroladores. O processo de portar a biblioteca ocorre pela criação de funções para a escrita de um *pixel* no *frame buffer*, além de funções de preenchimento e mesclagem de cores (KISS-VÁMOSI, 2018d).

A biblioteca também dá suporte a vários recursos avançados, como animações, temas e suporte a vários tipos de dispositivos de entrada. Também possui suporte a RTOS, aceleração gráfica e memória externa, sendo esses opcionais, caso não disponíveis.

Considerando um ponto de vista comercial, é necessário pontuar quais as necessidades do projeto, visto que a biblioteca de código fechado possui um custo elevado de licenciamento, apesar de possuir recursos gráficos avançados que não podem ser encontrados na outra plataforma, um exemplo disso sendo gráficos vetoriais. Mesmo com o licenciamento do *Embedded Wizard*, há limites de produção anual de dispositivos que executam o *firmware* com a biblioteca, então pode ser algo a considerar quando se envolve produção em larga escala (TARA SYSTEMS GMBH, 2017).

Mesmo possuindo uma grande gama de suporte a dispositivos embarcados, é possível que o projeto necessite de um dispositivo não disponível nessa lista, o que faz a adição do suporte dependente da desenvolvedora, no caso do *Embedded Wizard*. Isso pode fazer a *LittlevGL* uma solução mais atraente, caso haja conhecimento suficiente sobre o *hardware* para poder portar a execução da biblioteca para uma plataforma nova.

Um ponto positivo para a *Embedded Wizard* é que essa não se encontra disponível apenas para sistemas embarcados, mas também pode ser executado em ambientes móveis como o *Android*. O sistema operacional Linux embarcado também é uma opção, para execução em sistemas como o *Raspberry Pi*. Isso permite, por exemplo, que o mesmo projeto seja colocado em um aplicativo de *smartphone*, sendo útil no caso de equipamentos que podem ser controlados pelo mesmo (TARA SYSTEMS GMBH, 2019d).

Ambas as bibliotecas afirmam ter um baixo uso de memória e apresentam métodos distintos para atualizar o conteúdo apresentado na tela. Para verificar isso de forma mais quantitativa, um teste foi desenhado para a execução nessas duas bibliotecas junto com a que foi criada no Capítulo 3. Isso tem o objetivo de averiguar as diferenças entre o uso de recursos computacionais de bibliotecas que são utilizadas no mercado frente uma solução própria.

4.2 TESTES DE DESEMPENHO

Esta seção tem por finalidade definir um teste de desempenho para execução nas três bibliotecas explicitadas neste trabalho: a que foi criada no Capítulo 3 e as duas soluções disponíveis da Seção 4.1.

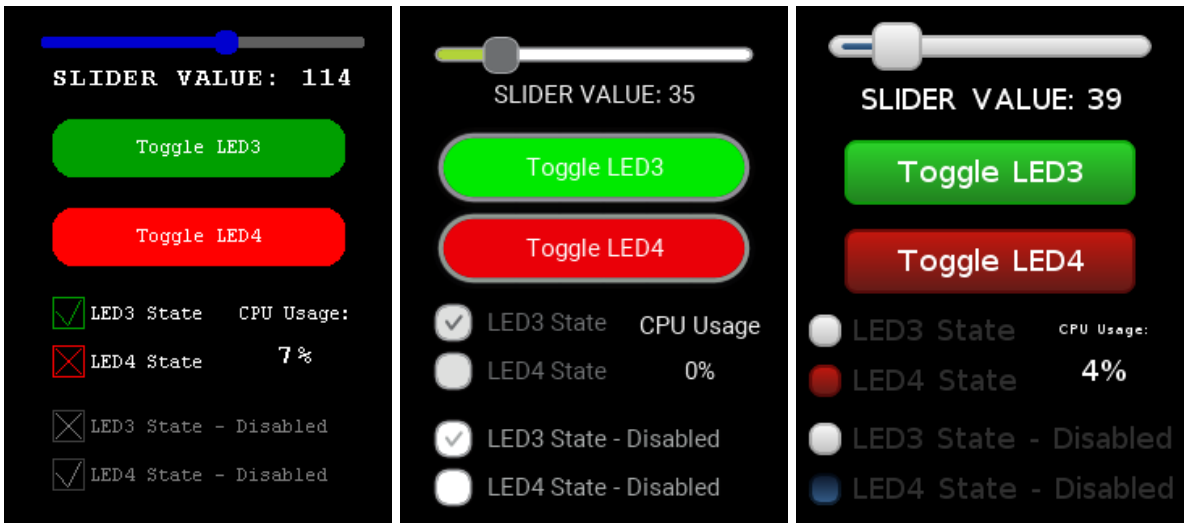
4.2.1 CONSTRUÇÃO

Conforme explicitado no Capítulo 1, a comparação entre as bibliotecas ocorreu em dois aspectos, de utilização do processador e de memória. Esses foram escolhidos por impactar significativamente no caso de uma migração, como explicitado ao decorrer do trabalho pela necessidade de armazenar uma ou mais cópias da tela em um *frame buffer*, armazenamento de dados de objetos das bibliotecas e enviar dados de *pixels* com alta frequência de atualização.

O *kit* de desenvolvimento utilizado neste trabalho, o STM32F429I-DISCO, utiliza o microcontrolador STM32F429ZI como base, trabalhando a uma frequência de 168 MHz nos testes. Possui 256 KB de memória RAM e 2 MB de memória *flash*. Externo ao microcontrolador existe 64 Mb, ou 8 MB, de memória SDRAM. Porém, isso não será incluso no teste de memória pois, para todos os casos, essa memória é utilizada exclusivamente para o armazenamento do *frame buffer* com a mesma profundidade de cor, o que não impacta a quantidade desse tipo de memória utilizada nos testes.

Com o objetivo de manter um ambiente de teste constante entre as três bibliotecas, foi construída uma interface gráfica utilizando os quatro elementos da biblioteca criada no Capítulo 3, então replicada da maneira mais similar possível nas outras duas. A Figura 33 mostra capturas de tela da interface criada nas três bibliotecas, sendo que a primeira refere-se a criação deste trabalho, a segunda ao *Embedded Wizard* e a terceira ao *LittleVGL*.

Figura 33 – Interface gráfica utilizada nos testes das três bibliotecas envolvidas



(a) Biblioteca desenvolvida neste trabalho **(b) Biblioteca *Embedded Wizard*** **(c) Biblioteca *LittleVGL***

Fonte: elaborado pelo autor.

Essa interface é composta de um *slider*, com intervalo de valores de 0 a 200, acompanhando de um texto abaixo para explicitar o valor atual da variável controlada pelo componente. Abaixo desse texto, encontra-se dois botões para alternar o estado dos dois LEDs presentes no *kit*. Abaixo desse botão, encontram-se quatro *checkboxes* mostrando o estado de ativação dos LEDs controlado pelos botões, sendo dois de estado habilitado e dois desabilitados e, desses dois, um para cada LED. Os *checkboxes* habilitados permitem controlar os LEDs da mesma forma que os botões. Por fim, duas caixas de texto são utilizadas para mostrar a medição de uso da CPU, sendo a primeira como texto estático para rotulação e a segunda dinâmica que é atualizada de acordo com o valor medido.

Para simular interação com esses componentes, foi utilizado código para escrever uma tarefa de usuário que modifica as propriedades desses componentes de forma constante. O movimento do *slider* de segurar e arrastar para modificar seu valor foi simulado ao atualizar seu valor constantemente, começando da metade do intervalo, incrementando até atingir seu valor máximo, então diminuindo até chegar seu valor mínimo, aumentando até chegar no valor máximo novamente e recomeçar o ciclo. Os botões e *checkboxes* foram alternados de estado a um intervalo de tempo pré-definido para simular o usuário pressionando os elementos de forma constante.

A mensuração da utilização do processador é realizada no tempo de execução do código, com o intuito de quantificar o tempo de utilização da CPU com elementos presentes na tela em constante atualização. O algoritmo que mede esse uso é fornecido junto com o *software STM32CubeMX*, ao transferir o pacote de *drivers* do *kit* de desenvolvimento em específico, então esse foi utilizado para realizar esse cálculo, que é feito uma vez por segundo e então mostrado na interface.

Com isso, criou-se quatro casos de teste para medir o uso de CPU, descritos abaixo. Para todos eles, há a atualização do texto que mostra a porcentagem de uso de processamento.

- **Ocioso:** não há interação simulada;
- **Botões e *Checkboxes*:** interação simulada com esses elementos, sendo que eles são pressionados constantemente a cada 500 ms;
- ***Slider*:** Movimento do *slider* simulado com a alteração de seu valor a cada 50 ms;
- **Ambos:** Contempla os casos de botões, *checkboxes* e *slider* descritos acima, ocorrendo ao mesmo tempo.

Cada caso manteve um valor constante de uso de processamento, exibido na interface de teste. Esse valor então foi considerado para os resultados do teste.

Para mensurar uso de memória, foi necessária uma divisão devido a utilização de dois tipos de memória diferente: a memória RAM, visto que é utilizada em pequena quantidade para utilização imediata de informações; e a memória *flash*, utilizada como armazenamento de longo prazo e não volátil. A medição da quantidade de memória utilizada em cada caso foi feita em tempo de compilação, visto que o *plugin* que provê o compilador utilizado fornece essas informações uma vez sucedida essa tarefa.

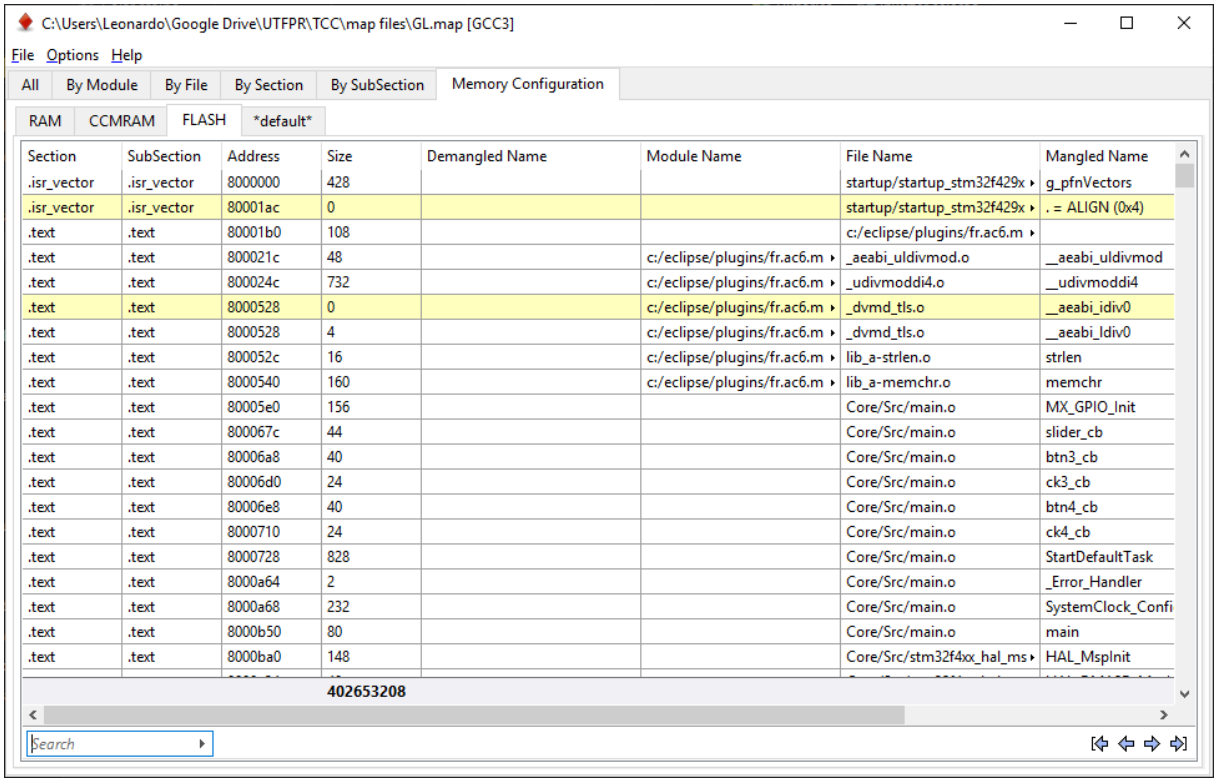
Outro quesito levado em consideração foi como a compilação foi feita. Em tempo de desenvolvimento, normalmente se realiza a compilação em modo de depuração (*debug*), o que incluí recursos junto com o código compilado para facilitar o desenvolvimento da aplicação, como visualização do conteúdo da memória, inserção de pontos de quebra no código (*breakpoints*) e análise em caso de erros em tempo de execução. Porém, para a medição de memória, foi necessário a remoção desses recursos, o que foi feito ao realizar a compilação em modo de produção (*release*).

Ainda no quesito de compilação, um dos resultados após esse processo é um arquivo denominado **mapa de ligação** (*Linker Map Files*). Esse arquivo, gerado uma vez por compilação e por projeto, tem utilidade de verificar o tamanho e os endereços utilizados por seções de código e dados da aplicação, assim como o carregamento de dados e módulos carregados de bibliotecas externas. Com esse arquivo presente, para auxiliar na tarefa de distinguir em que tipo de memória cada seção de código e dados se encontra, utilizou-se um *software* gratuito para análise desses arquivos. Esse *software* é denominado AMAP, mantido de forma independente por Sergey Sikorskiy e disponível em <http://www.sikorskiy.net/prj/amap>.

Ao carregar um desses arquivos na interface do *software*, são apresentados os dados extraídos em uma tabela, podendo ser ordenados por nome, grupo, seção, endereço ou tamanho de cada objeto, além de poder agrupá-los por módulo de biblioteca carregado, arquivo de código-fonte ou, no caso necessário para teste, o tipo de memória em que o símbolo se encontra inserido. A Figura 34 mostra a interface desse *software*.

Com isso, após a compilação do *firmware* utilizando cada uma das bibliotecas que foram testadas, os arquivos de mapa de ligação foram carregados nesse *software* e separados por tipo de memória. Então, as seções pertinentes a biblioteca em questão foram extraídas para calcular a quantidade de memória utilizada e inseridas em uma planilha para a realização do somatório.

Figura 34 – Interface gráfica do *software* AMAP



Fonte: captura de tela da aplicação no sistema operacional Windows 10.

4.2.2 VALORES OBTIDOS

Os valores de uso de processador foram obtidos por meio da interface utilizada nos testes, ao executar cada caso de teste de maneira separada e então observando o valor apresentado na tela, visto que esse manteve-se em valor constante para permitir a observação. A Tabela 10 reúne as informações obtidas com esse teste.

Tabela 10 – Resultados obtidos para utilização de processador nos casos de testes

Caso de Teste	**Solução Própria**	***Embedded Wizard***	***LittleVGL***
Ocioso	1%	0%	0%
Botões e Checkboxes	2%	0%	3%
Slider	11%	4%	10%
Ambos	13%	4%	12%

Fonte: autoria própria.

A princípio, observa-se que a *Embedded Wizard* possuiu uso significativamente mais baixo do que as outras duas bibliotecas. O único caso de teste que teve uso significativo foi o que movimenta o *slider*, visto que no estado ocioso e pressionamento dos botões e *checkboxes* não houve diferença mensurável. Isso reflete no valor obtido no teste de ambos os casos, que foi igual ao caso de teste do *slider*.

A biblioteca criada e a *LittleVGL* obtiveram resultados similares quanto a esse quesito, com a biblioteca de código aberto possuindo um pouco menos de utilização nos últimos dois quesitos. Também é notável que a criação desse trabalho foi a única que registrou um valor acima de 0% quando se tratou do uso em ociosidade.

Algo a ser considerado é que, apesar de ter atingido utilização similar a uma das soluções comerciais, as duas têm complexidade dos gráficos gerados significativamente maior. Exemplos disso são a utilização de fontes com maior profundidade de *pixel* e recursos como *anti-aliasing*, componentes mais complexos e em maior quantidade, além de proverem suporte a utilização de imagens.

Porém, isso têm uma consequência na utilização de memória, visto a seguir. Conforme explicitado na Seção 4.2.1, a medição do uso de memória foi feito fora do tempo de execução, por meio da análise dos arquivos gerados no tempo de compilação dos *firmwares*. Assim, não houve casos de teste para a utilização de memória, com a medição ocorrendo sem a inclusão de nenhum código de simulação de interação, o que permitiu medir a memória utilizada apenas com a biblioteca e os objetos utilizados para o uso da interface construída. A Tabela 11 mostra as informações obtidas nesse quesito.

Tabela 11 – Resultados obtidos para utilização de memória nos casos de testes

Tipo de Memória	Solução Própria	***Embedded Wizard***	***LittleVGL***
RAM	72 bytes	980 *bytes*	64.604 *bytes*
Flash	9.975 *bytes*	215.129 *bytes*	58.132 *bytes*
Total	10.047 *bytes*	216.109 *bytes*	122.376 *bytes*

Fonte: autoria própria.

O que se destaca nessa medição é a quantidade de memória *flash* usada na solução desenvolvida, cerca de 21 vezes menor do que a utilização de memória da *Embedded Wizard*. Como explicitado, as soluções comerciais dispõem de recursos mais complexos, além de vários componentes pré-existentes e o suporte para variados

dispositivos de entrada além do painel de toque, ante os quatro elementos vistos no *software* desenvolvido nesse trabalho.

Ainda no quesito de memória *flash*, identifica-se um uso mais elevado da solução *Embedded Wizard*. Em análise com o *software* AMAP, viu-se uma utilização de 98.649 *bytes* do fornecimento do código compilado, sendo 45,86% da utilização desse tipo de memória. Além disto, 104.459 *bytes* foram oriundos de códigos gerados pela ferramenta de desenvolvimento utilizada para criar a interface de teste, o que resulta em 48,6%. Isso mostra que a ocupação elevada de memória dessa solução ocorre pela complexidade dos elementos disponíveis para uso e as funcionalidades para permitir um bom aproveitamento do *hardware* disponível.

Uma funcionalidade que demanda uma quantidade de armazenamento considerável são as fontes para exibição de caracteres, o qual é indispensável para as três bibliotecas e o teste realizado nelas. Com isso, foi efetuada uma análise da proporção de memória utilizada para o armazenamento das fontes. Na solução própria, foram necessários 4.940 *bytes*, enquanto na *Embedded Wizard*, 8.740 *bytes* e, por fim, 22.880 *bytes* na *LittleVGL*, o que resulta em proporções de 49,52%, 4,06% e 39,36% respectivamente, da utilização total de memória *flash* de cada solução analisada.

Já em relação a memória RAM, identifica-se o uso largamente elevado da *LittleVGL* em relação as outras duas. Isso ocorre por dois motivos. O primeiro deles é a utilização do que é denominado *Virtual Display Buffer* (VDB). Isso é um *frame buffer* alocado na memória RAM para permitir utilizar recursos avançados como *anti-aliasing*, transparência e sombreamento.

O autor da biblioteca sugere que o tamanho do VDB seja de um décimo do valor total do tamanho do *frame buffer*, calculado a partir da altura em *pixels*. No caso de um *frame buffer* de tamanho 320×240 *pixels* a 32 BPP, isso seria um décimo da altura de 320 *pixels*, resultando em 32 *pixels*, totalizando um acréscimo de 30.720 *bytes* ($32 \times 240 \times 32 \div 8$). No caso do teste, utilizou-se 30 *pixels* de altura para o VDB, o que resultou em um acréscimo de 28.800 *bytes* ($30 \times 240 \times 32 \div 8$).

O segundo motivo é a alocação de memória para uso dos objetos. Essa é feita em tempo de compilação quando se opta por utilizar a própria biblioteca para gerenciar a memória. No teste, esse valor estava definido em 32.768 *bytes*. Em testes realizados foi possível executar o ensaio utilizando 4.096 *bytes*, ocorrendo problemas com alocação de memória ao utilizar valores menores que esse. Todavia, esse valor necessitaria ser mais elevado conforme a complexidade da GUI aumenta. Assim, visto que o VDB é opcional, o uso de memória RAM cairia para cerca de 7 KB caso não fizesse seu uso e diminuísse a quantidade de memória de objetos para 4 KB.

Por fim, apesar da quantidade limitada de componentes envolvidos na criação da solução própria, isso faz com que seu uso seja mais viável no cenário de uma limitação de memória consideravelmente alto, em que deve-se usar apenas o mínimo necessário para a construção de uma interface gráfica.

Caso a limitação venha a ser de capacidade de processamento, os resultados indicam que é mais favorável o uso da solução comercial *Embedded Wizard*, por mostrar eficiência na utilização do processador e do acelerador gráfico disponível no dispositivo. Isso acaba por se tornar mais evidente levando em conta da análise da quantidade de memória utilizada para armazenar código relativo a funcionalidade crítica da biblioteca. Além disso, no seu ambiente de desenvolvimento, além de dispor de componentes já construídos para a utilização, dispõe de funcionalidades de efeitos, animação e uso de imagens que fazem parte do que é denominado o núcleo (*core*) da estrutura denominada *Mosaic*. Essa parte é protegida contra alterações no ambiente e acaba por ser incluído no código utilizado.

5 CONCLUSÃO

Considerando a proposta do projeto de apresentar a arquitetura de interface das telas de TFT-LCD e realizar a construção de uma interface gráfica para sistemas embarcados, o trabalho pode ser dividido em duas partes quanto a análise da conclusão dos objetivos. Primeiramente são analisado os três primeiros objetivos específicos apresentados na Seção 1.1.2 por englobarem o referencial teórico do trabalho, para então proceder com a análise dos três últimos objetivos que envolvem a parte de desenvolvimento e resultado.

Em relação a primeira parte, foi possível expor o estado da arte do uso dessa tecnologia envolvendo as maneiras diferentes de integrar os componentes dos mesmos, padrões de conexão utilizados na indústria, interfaces utilizadas e parâmetros necessários para o funcionamento da tecnologia. Por mais que a apresentação possuiu enfoque em um fabricante com um modelo específico da plataforma e um controlador de tela, exemplos com modelos diferentes foram fornecidos, incluindo soluções disponíveis de outros fabricantes. Isso mostra o quão difundido a tecnologia se encontra, com padrões de conexões bem definidos que permite a interoperabilidade de um módulo de tela TFT-LCD para mais de uma plataforma.

Além disso, a comparação analítica não só se limitou as telas de LCD alfanuméricas de matriz de pontos, também provendo um ponto mediano entre essa tecnologia e a TFT-LCD, que foram as telas gráficas monocromáticas, elucidando mais uma possível opção de uso para exibição de informações. O aumento de complexidade envolveu não só a análise de uso de memória e processamento, mas também o uso energético, o que revelou que a maioria do uso de energia não ocorre pelo uso do controlador ou a tela em si, mas por uso de retroiluminação.

Outra tecnologia que também foi discutida durante o andamento do trabalho, além da TFT-LCD, é a interação do usuário com a tela por meio do toque. Ao explorar algumas formas disponíveis de permitir a interação com a tela, foi brevemente apresentando o funcionamento da tecnologia capacitiva, com enfoque maior a opção resistiva, visto a presença desse no *kit* de desenvolvimento que norteou este projeto.

Com um grande enfoque no *kit* de desenvolvimento STM32F429I-DISCO, da STMicroelectronics, a arquitetura de conexão entre o microcontrolador disponível nesse *kit* e seu módulo de tela TFT-LCD foi exposto conjuntamente com a arquitetura, ao ser utilizado como exemplo durante as discussões do referencial teórico.

Também foi necessário detalhar brevemente o funcionamento de um acelerador gráfico. Apesar de não fazer parte da tecnologia de TFT-LCD e a explicação ter

focado na solução de um fabricante, é possível ver que seu uso também é difundido na indústria por conta do auxílio que esse provê ao diminuir a carga de processamento e permitir uma experiência mais satisfatória ao usuário.

Já o que concerne ao desenvolvimento e os resultados do trabalho, foram alcançadas as propostas de implementação, com um *driver* que realiza a comunicação por meio do protocolo SPI, utilizando o componente LTDC do microcontrolador para a transferência dos dados RGB para a memória do controlador ILI9341. Uma vez sucedida a escrita de um *pixel* no *frame buffer*, resultando nesse aparecendo corretamente na tela do módulo presente no *kit*, prosseguiu-se com a criação para algoritmos de desenho de formas primitivas, como linha, círculo e quadrado, além da implementação básica de exibição de caracteres na tela. Com essas funcionalidades completas, designou-se quatro componentes para utilização nessa biblioteca, que foram expostos na Seção 3.4.

Tendo em vista as implementações concluídas, teve-se em vista as soluções disponíveis, que no caso foram as bibliotecas *Embedded Wizard* e *LittleVGL*. Primeiramente elas foram analisadas de acordo com as informações disponíveis em seus respectivos *websites*, o que permitiu verificar o quesito de facilidade de uso por meio do ambiente de desenvolvimento que proporcionam, a disponibilidade para plataformas e recursos gráficos disponíveis.

O que se concluiu dessa comparação é que a *Embedded Wizard*, apesar do preço relativamente elevado, possui disponibilidade para plataformas que estão até mesmo fora do espaço de sistemas embarcados. Além disso, dentre as soluções testadas, foi a que mais trouxe recursos gráficos, um exemplo desse sendo a utilização de gráficos vetoriais. Também é notável a utilização de uma linguagem própria para facilitar o ciclo de desenvolvimento. Porém, dependendo da finalidade do projeto, a barreira de preço para licenciamento seja alta e ainda há a dependência da desenvolvedora caso seja necessário trazer a implementação para uma plataforma nova.

Caso esses quesitos sejam uma preocupação, a opção gratuita e de código aberto pode ser levada em consideração. Isso não se dá apenas pela gratuidade, mas o código aberto permite a possibilidade de realizar o *port* da biblioteca para novas plataformas tendo apenas como barreira o conhecimento disponível sobre a plataforma alvo de utilização. E mesmo que se opte pela *LittleVGL*, não há perda significativa de funcionalidades, pois ainda existem características como *anti-aliasing*, utilização de imagens e animações.

Isso conclui a comparação subjetiva das soluções existentes. Assim, resta a parte de desempenho, que foi realizada com o desenho e execução do teste. Para

manter em igualdade esse teste, foi criado uma interface que pudesse ser replicada nas três bibliotecas envolvidas neste projeto, com o intuito de mensurar a utilização de recursos computacionais. Por conta de possuir a quantidade menor de recursos, essa tela foi desenhada com os componentes da solução própria, sendo então replicado para as soluções comerciais.

A partir disso, foram explicitadas as maneiras para obter dados de utilização de processador e memória do microcontrolador presente no *kit* de desenvolvimento. Uma vez explicitados os resultados oriundos dos testes de desempenho, foi possível realizar mais conclusões sobre a soluções e definir quando é mais vantajoso utilizar uma solução comercial frente a uma solução própria.

Devido a menor quantidade de componentes criados e a não presença de recursos avançados de desenho, a criação de uma solução própria é uma viabilidade caso haja uma restrição considerável de espaço de armazenamento na plataforma de desenvolvimento a ser utilizada, caso outras soluções não sejam viáveis a encaixarem na hipotética limitação de memória do projeto. Isso foi uma vantagem percebida ao notar a utilização cerca de dez vezes menor de memória *flash* da solução própria. A solução comercial paga obteve a maior utilização de memória *flash* e a gratuita obteve a maior utilização de memória total. Porém, foi explicitado que recursos opcionais foram utilizados no teste que eram passivos de serem desativos.

Porém, isso deve ser acompanhado da análise de limitação de processamento que pode ocorrer junto com a limitação de memória. Pois, ao passo que a solução própria possui menos dados armazenados, sua utilização de processamento foi consideravelmente maior, ainda mais por não possuir recursos gráficos avançados. Ainda no quesito de processamento, a solução não gratuita, a *Embedded Wizard*, foi a que possuiu o menor uso de processamento de todas, o que remete a otimização de utilização de recursos que essa traz. A solução própria e a *LittleVGL* ficaram em termos similares, todavia deve ser exaltado novamente que a posterior possui recursos mais avançados sendo utilizados, o que mostra que, apesar do nível de otimização não ser tão considerável quanto a *Embedded Wizard*, esse ainda é satisfatório.

Isso conclui a análise de completude dos objetivos apontados no trabalho, explicitando a natureza complexa não só da tecnologia de TFT-LCD em si, mas também da utilização, do processamento e armazenamento de dados envolvido para manter a imagem na tela atualizada de forma constante e com uma experiência de usuário satisfatória. Pode-se concluir também que o desenvolvimento de uma solução própria toma uma quantidade considerável de tempo para atingir um nível de otimização sa-

tisfatório, quesito que é realçado ao levar em consideração que a *Embedded Wizard* é uma solução disponível desde 2003 e a *LittleVGL*, desde 2009.

A tecnologia para exibição de imagens está em constante avanço e novas tecnologias estão sempre surgindo, um exemplo desse sendo os diodos emissores luz orgânico (*Organic Led Emitting Diode* - OLED), apontado por Fujitsu ... (2006) como uma potencial tendência do futuro. Com essas tecnologias avançando, mais recursos serão requisitados das GUIs, o que fará com que seja necessário mais recursos de armazenamento e processamento.

REFERÊNCIAS

ALENCAR, Felipe. **Windows 7 Aero: o que é, quais os benefícios e por que evitar no PC**. 2016. Disponível em: <https://www.techtudo.com.br/dicas-e-tutoriais/noticia/2016/01/windows-7-aero-o-que-e-quais-os-beneficios-e-por-que-evitar-no-pc.html>. Acesso em: 27 mai. 2019.

BESSIN, Geoff. **Top Touch Display Technologies On The Market**. 2017. Disponível em: <https://www.intuiface.com/blog/top-touch-display-technologies-on-the-market>. Acesso em: 29 mai. 2019.

DEVNATH, Varadarajan. Challenges in migrating to an LCD-based design. **Industrial Embedded Systems**, 2014. Disponível em: <http://industrial.embedded-computing.com/articles/challenges-migrating-an-lcd-based-design/>. Acesso em: 5 set. 2017.

DRAUPNER GRAPHICS A/S. **Licenses**. 2017. Disponível em: <https://touchgfx.com/services-pricing/licenses/>. Acesso em: 24 nov. 2017.

ELGSCREEN. **Série "A linguagem técnica traduzida": dithering (matização)**. 2014. Disponível em: <http://blog.elgscreen.com/dithering-matizacao/>. Acesso em: 5 nov. 2017.

FAIRCHILD SEMICONDUCTOR CORP. **Devices with a Synchronous Pixel Interface**. [S.l.], 2007. Rev. 1.3.4.

FREESCALE SEMICONDUCTOR, INC. **Introduction to Embedded Graphics with Freescale Devices**. [S.l.], 2015. Rev 0.

FUJITSU MICROELECTRONICS AMERICA, INC. **Fundamentals of Liquid Crystal Displays – How They Work and What They Do**. [S.l.], 2006.

GRUNSKE, Joseph. **The 5 Most Popular Touch Screen Technologies Available Today**. 2017. Disponível em: <http://www.generaldigital.com/blog/the-five-most-popular-touch-screen-technologies-available-today/>. Acesso em: 29 mai. 2019.

GU, Changyi. **Building Embedded Systems: Programmable Hardware**. [S.l.]: Apress, 2016.

GUDINO, Miguel. **How Touch Sensors Work**. 2018. Disponível em: <https://www.arrow.com/en/research-and-events/articles/how-touch-sensors-work>. Acesso em: 29 mai. 2019.

GUPTA, R.G. **Television Engineering and Video Systems**. [S.l.]: Tata McGraw-Hill, 2005. (McGraw-Hill Electrical and Electronic Engineering Series). ISBN 9780070585966.

HITACHI, LTD. **HD44780U (LCD-II) Dot Matrix Liquid Crystal Display Controller/Driver**. [S.l.], 1998. Rev 0.0.

ILI TECHNOLOGY CORP. **a-Si TFT LCD Single Chip Driver 240RGBx320 Resolution and 242K color**. [S.l.], 2011. Version v1.11.

KISS-VÁMOSI, Gábor. **Donate - Support LittlevGL's development with a donation**. 2018. Disponível em: <https://littlevgl.com/donate>. Acesso em: 17 abr. 2019.

KISS-VÁMOSI, Gábor. **LittlevGL Documentation - PC Simulator**. 2018. Disponível em: <https://docs.littlevgl.com/#PC-simulator>. Acesso em: 17 abr. 2019.

KISS-VÁMOSI, Gábor. **LittlevGL Documentation - Porting**. 2018. Disponível em: <https://docs.littlevgl.com/#Porting>. Acesso em: 17 abr. 2019.

KISS-VÁMOSI, Gábor. **LittlevGL Documentation - Welcome**. 2018. Disponível em: <https://docs.littlevgl.com/#Welcome>. Acesso em: 17 abr. 2019.

KUO, Yue. Thin film transistor technology – past, present, and future. **The Electrochemical Society Interface**, The Electrochemical Society, v. 22, n. 1, p. 55–61, 2013.

MICROCHIP TECHNOLOGY INC. **Developing Embedded Graphics Applications using PIC®Microcontrollers with Integrated Graphics Controller**. [S.l.], 2011.

NXP SEMICONDUCTORS. **Interfacing 4-wire and 5-wire resistive touchscreens to the LPC247x**. [S.l.], 2008. Rev. 02.

OHSHIMA, Hiroyuki. Mobile display technologies: Past, present and future. In: IEEE. **Solid-State Circuits Conference (A-SSCC), 2014 IEEE Asian**. [S.l.], 2014. p. 1–4.

POYNTON, C. **Digital Video and HD: Algorithms and Interfaces**. [S.l.]: Elsevier Science, 2003. (The Morgan Kaufmann Series in Computer Graphics). ISBN 9780080504308.

SAEF TECHNOLOGY LIMITED. **Specification of LCD Module No.: SF-TC240T-9370A-T**. [S.l.], 2012.

SEGGER MICROCONTROLLER GMBH & CO. KG. **emWin Graphic Library with Graphical User Interface User & Reference Guide**. [S.l.], 2015.

SELF-CAPACITIVE Touch Panel Controlle. [S.l.], 2012.

SHARP CORPORATION. **Interfacing LCD Panels to Microcontrollers**. [S.l.], 2007. Reference Code SMA07001.

SHIM, Hojun. Low-power lcd display systems. **tech. rep**, 2006.

SILICON LABORATORIES INC. **Interfacing Graphical Displays**. [S.l.], 2016. Rev. 1.07.

SITRONIX TECHNOLOGY CORP. **ST7920 Chinese Fonts Built-In LCD Controller/DRiver**. [S.l.], 2002.

STMICROELECTRONICS. **TFT-LCD interfacing with the high-density STM32F10xxx FSMC**. [S.l.], 2008. Rev 2.

STMICROELECTRONICS. **S-Touch®advanced resistive touchscreen controller with 8-bit GPIO expander**. [S.l.], 2011. Doc ID 14489 Rev 5.

STMICROELECTRONICS. **S-Touch®STMPE811 resistive touchscreen controller advanced features**. [S.l.], 2011. Doc ID 15023 Rev I.

STMICROELECTRONICS. **Embedded Graphics on STM32F4**. 2013. V0.1. Disponível em: <http://www.compel.ru/wordpress/wp-content/uploads/2013/11/1_LTDC_ChromeART.pdf>. Acesso em: 1 out. 2017.

STMICROELECTRONICS. **Getting Started with STemWin Library**. [S.l.], 2014. Doc ID 024959 Rev 4.

STMICROELECTRONICS. **LCD-TFT display controller (LTDC) on STM32 MCUs**. [S.l.], 2017. Doc. ID 029237 Rev. 2.

STMICROELECTRONICS. **STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced ARM®-based 32-bit MCUs**. [S.l.], 2017. Doc. ID 018909 Rev 15.

STMICROELECTRONICS. **STM32L4 - Chrom-ART**. 2017. Disponível em: <https://www.st.com/content/ccc/resource/training/technical/product_training/group0/a9/3a/26/e0/bf/f2/4a/01/STM32L4-System-ChromART%28DMA2D%29/files/STM32L4-System-ChromART%28DMA2D%29.pdf/_jcr_content/translations/en.STM32L4-System-ChromART%28DMA2D%29.pdf>. Acesso em: 17 abr. 2019.

STMICROELECTRONICS. **Getting started with STM32CubeF4 MCU Package for STM32F4 Series**. [S.l.], 2019. Rev 14.

TARA SYSTEMS GMBH. **Pricing Model**. 2017. Disponível em: <https://www.embedded-wizard.de/pricing.html>. Acesso em: 24 nov. 2017.

TARA SYSTEMS GMBH. **Basic concepts: Platform Package**. 2019. Disponível em: <https://doc.embedded-wizard.de/platform-package>. Acesso em: 18 abr. 2019.

TARA SYSTEMS GMBH. **Basic concepts: Programming language Chora**. 2019. Disponível em: <https://doc.embedded-wizard.de/programming-language-chora>. Acesso em: 18 abr. 2019.

TARA SYSTEMS GMBH. **Basic concepts: Visually aided programming**. 2019. Disponível em: <https://doc.embedded-wizard.de/visual-aided-programming>. Acesso em: 18 abr. 2019.

TARA SYSTEMS GMBH. **Features**. 2019. Disponível em: <https://www.embedded-wizard.de/features/>. Acesso em: 19 abr. 2019.

TARA SYSTEMS GMBH. **Welcome to Embedded Wizard**. 2019. Disponível em: <https://doc.embedded-wizard.de/welcome-to-embedded-wizard>. Acesso em: 18 abr. 2019.

TECHTUDO. **O Windows Aero traz mais elegância ao sistema**. 2016. Disponível em: <https://www.techtudo.com.br/dicas-e-tutoriais/noticia/2016/01/windows-7-aero-o-que-e-quais-os-beneficios-e-por-que-evitar-no-pc.html>.

TEXAS INSTRUMENTS INC. **4-Wire and 8-Wire Resistive Touch-Screen Controller Using the MSP430™**. [S.l.], 2010.

TRUE Multi-Touch Capacitive TouchPanel Controller. [S.l.], 2013.

U.S. DEPARTMENT OF HEALTH & HUMAN SERVICES. **User Interface Elements**. Disponível em: <https://www.usability.gov/how-to-and-tools/methods/user-interface-elements.html>. Acesso em: 17 abr. 2019.