

AGT DISPLAYS

Programação do MCU

Driver serial completo para comunicação com TFT Módulo Serial

ESCOPO

Código no MCU externo que conversa com o display

PRÉ-REQUISITO

Projeto DGUS já gravado, com VP definidos

LINGUAGEM

C portátil — Arduino, STM32, ESP32, PIC

APLICAÇÃO

Módulos com ASIC DWIN rodando DGUS II

1. O QUE ESTE DOCUMENTO COBRE

O display DGUS não recebe comandos de desenho: ele lê e escreve valores em endereços de memória. Todo o trabalho do MCU se resume a duas operações — gravar uma *word* num VP e receber a *word* que o operador alterou pelo toque. O que separa um protótipo de um produto é o tratamento do que acontece em volta dessas duas operações: quadro partido pela metade, resposta que ninguém esperava, ruído na linha, leitura que não chega.

Este documento traz um driver completo para o lado do MCU, montado em camadas e pronto para copiar. O exemplo mínimo — abrir a serial, montar sete bytes e mandar — está no guia de desenvolvimento e serve para o primeiro teste. Aqui o objetivo é o código que roda em campo.

O QUE O GUIA DE DESENVOLVIMENTO COBRE	O QUE ESTE DOCUMENTO ACRESCENTA
Envio de uma word para o VP	Envio de bloco de words num só quadro, e escrita em SP
Recepção sincronizada pelo cabeçalho	Máquina de estados com validação de comprimento e recuperação de quadro inválido
Descarte da resposta 4F 4B	Despacho por instrução, com <i>callback</i> de aplicação
—	Leitura ativa pela instrução 0x83, com controle de pendência
—	CRC-16 opcional, no envio e na verificação
—	Timeout de quadro e re-sincronização
Loop principal com temporização fixa	Cadência coerente com o ciclo interno do display

Sobre o C51 do display. A DWIN publica um programa C51 de referência para as UARTs, mas esse código roda **dentro do display**, no núcleo OS — não no MCU do cliente. Ele resolve o mesmo problema pelo lado oposto da linha. A arquitetura de recepção deste documento segue a estrutura daquele código, adaptada ao MCU externo; é a mesma máquina de estados, com os nomes traduzidos e o tratamento de resposta invertido.

2. ARQUITETURA DO DRIVER

Quatro camadas, de baixo para cima. Cada uma só conhece a de baixo, o que permite trocar a porta serial sem mexer no protocolo e trocar a aplicação sem mexer no quadro.

CAMADA	RESPONSABILIDADE	O QUE EXPÕE
1. Porta	Ler e escrever bytes crus na UART	<code>port_write()</code> , <code>port_read()</code> , <code>millis()</code>
2. Quadro	Montar quadro na saída; reconhecer quadro na entrada	<code>dgus_rx_byte()</code> e as funções de envio
3. Protocolo	Separar 0x82 de 0x83, conferir CRC, extrair VP e dados	<code>dgus_dispatch()</code>
4. Aplicação	Decidir o que fazer com cada VP recebido	<code>dgus_on_touch()</code> , <code>dgus_on_read()</code>

A camada 1 é a única que muda de plataforma para plataforma. Nas três de cima, o código é o mesmo em Arduino, STM32 ou PIC.

```
/* dgus.h – constantes do protocolo */
#define DGUS_HDR1      0x5A
#define DGUS_HDR2      0xA5
#define DGUS_CMD_WRITE 0x82
#define DGUS_CMD_READ  0x83
#define DGUS_FRAME_MAX 256 /* cabeçalho + len + 249 dados + CRC */
#define DGUS_TIMEOUT_MS 50 /* quadro abandonado após esse silêncio */

#define DGUS_USE_CRC    0 /* 1 somente se o CFG do display tiver CRC ligado */

/* Folga que dgus_send() precisa no fim de todo buffer de saída.
   Todo buffer de quadro é dimensionado com esta folga, sempre – assim ligar
   o CRC é de fato trocar o #define acima, e não revisar cada função. */
#if DGUS_USE_CRC
    #define DGUS_CRC_ROOM 2
#else
    #define DGUS_CRC_ROOM 0
#endif
```

3. CAMADA 1 – A PORTA SERIAL

O formato depende da UART usada. Só a UART2 é fixa em 8N1; as demais aceitam paridade e dois bits de parada, selecionáveis por configuração de OS.

PORTA	FORMATO	OBSERVAÇÃO
UART2	8N1, fixo	Porta de depuração do DGUS. É a que o projeto usa por padrão, e a única disponível sem configuração de OS
UART4	N81 / E81 / O81 / N82	Só após configuração de OS. Em vários módulos é RS485
UART5	N81 / E81 / O81 / N82	Só após configuração de OS

Baud padrão de fábrica: 115200 bps. Terra comum entre MCU e display é obrigatório, e o nível elétrico precisa ser conferido no datasheet do módulo – TTL 3,3 V, TTL 5 V e RS232 não são intercambiáveis.

```

/* --- Arduino (UART hardware dedicada, não a de depuração) --- */
#define DGUS_PORT Serial1

static void port_init(void) { DGUS_PORT.begin(115200); }
static void port_write(const uint8_t *p, uint8_t n) { DGUS_PORT.write(p, n); }
static int port_read(void) { return DGUS_PORT.available() ?
DGUS_PORT.read() : -1; }
/* millis() já existe */

/* --- STM32 HAL --- */
/* static void port_write(const uint8_t *p, uint8_t n) {
* HAL_UART_Transmit(&huart2, (uint8_t*)p, n, 100);
* }
* A recepção vai por interrupção: no callback, chame dgus_rx_byte(byte, HAL_GetTick());
*/

```

Não use a serial de depuração. Num Arduino UNO, Serial é compartilhada com o USB: cada Serial.print() de diagnóstico entra na linha do display como lixo, e o bootloader manda bytes no reset. Use uma UART dedicada, ou uma placa com mais de uma.


```

/* n de 1 a 122 words (limite de 249 bytes de dado no quadro) */
void dgus_write_words(uint16_t vp, const uint16_t *w, uint8_t n)
{
    uint8_t f[DGUS_FRAME_MAX];
    uint16_t i, k = 0;

    if (n == 0 || n > 122) return;

    f[k++] = DGUS_HDR1;
    f[k++] = DGUS_HDR2;
    f[k++] = (uint8_t)(3 + 2 * n); /* instrução + endereço + dados */
    f[k++] = DGUS_CMD_WRITE;
    f[k++] = (uint8_t)(vp >> 8);
    f[k++] = (uint8_t)vp;
    for (i = 0; i < n; i++) {
        f[k++] = (uint8_t)(w[i] >> 8);
        f[k++] = (uint8_t)w[i];
    }
    dgus_send(f, (uint8_t)k);
}

```

4.3. TEXTO

O controle *Text Display* guarda a string no VP, dois caracteres por word. Preencher o resto com espaço, e não com zero, evita restos do valor anterior aparecerem na tela.

```

/* total_bytes = comprimento reservado no editor, sempre par, até 246 */
void dgus_write_text(uint16_t vp, const char *s, uint8_t total_bytes)
{
    uint8_t f[DGUS_FRAME_MAX];
    uint16_t i, k = 0;

    if (total_bytes & 1) total_bytes++;
    if (total_bytes == 0 || total_bytes > 246) return; /* 3 + 246 cabe no byte de len */

    f[k++] = DGUS_HDR1;
    f[k++] = DGUS_HDR2;
    f[k++] = (uint8_t)(3 + total_bytes);
    f[k++] = DGUS_CMD_WRITE;
    f[k++] = (uint8_t)(vp >> 8);
    f[k++] = (uint8_t)vp;
    for (i = 0; i < total_bytes; i++)
        f[k++] = *s ? (uint8_t)*s++ : ' '; /* padding com espaço */
    dgus_send(f, (uint8_t)k);
}

```

4.4. ESCRITA EM SP

Não existe função separada: a instrução 0x82 não distingue as faixas. Escrever no SP-base somado ao offset do atributo muda a aparência do controle em tempo de execução.

```
#define SP_TEMP      0x9000  /* SP-base do Data Variables da temperatura */
#define SP_OFF_COLOR 0x03    /* offset do atributo de cor */
#define SP_OFF_POS   0x01    /* offset de X, Y – duas words */

#define RGB565_RED    0xF800
#define RGB565_BLACK  0x0000

void temp_set_alarm(bool on)
{
    dgus_write_word(SP_TEMP + SP_OFF_COLOR, on ? RGB565_RED : RGB565_BLACK);
}

void temp_move(uint16_t x, uint16_t y)
{
    uint16_t pos[2] = { x, y };
    dgus_write_words(SP_TEMP + SP_OFF_POS, pos, 2); /* X e Y num quadro só */
}
```

Cada tipo de controle tem sua própria tabela de offsets. A estrutura inicial é comum: offset 0x00 é sempre o VP, 0x01 a posição e 0x03 a cor.

5. CAMADA 2 – RECEPÇÃO

É aqui que o protótipo costuma quebrar. O erro típico é ler o quadro inteiro de uma vez, assumindo que ele chegou completo — o que é falso: a UART entrega byte a byte, e a 115200 bps um quadro de oito bytes leva cerca de 0,7 ms para atravessar a linha. Se o código ler no meio disso, pega meio quadro e descarta o resto.

A solução é uma **máquina de estados alimentada byte a byte**, que só entrega o quadro à camada de cima quando ele está completo e válido. Cada byte entra pela mesma função, venha de *polling* ou de interrupção.

ESTADO	ESPERANDO	SE NÃO FOR O ESPERADO
S_HDR1	0x5A	Descarta o byte e continua no mesmo estado
S_HDR2	0xA5	Volta ao início — mas se o byte for outro 0x5A, permanece aqui
S_LEN	Comprimento entre 0x03 e o limite do buffer	Volta ao início: comprimento impossível significa que o "cabeçalho" era dado
S_BODY	Instrução e dados, até completar o comprimento	Quadro pronto → despacha e volta ao início

O detalhe do 0x5A duplo. Em S_HDR2, receber outro 0x5A não é erro: o primeiro era dado, e este pode ser o cabeçalho verdadeiro. Voltar cegamente a S_HDR1 descartaria um cabeçalho legítimo e perderia o quadro seguinte. Vale o mesmo para qualquer valor de dado que colida com o cabeçalho — e 0x5A aparece em dado com frequência.

```

typedef enum { S_HDR1, S_HDR2, S_LEN, S_BODY } dgus_state_t;

static dgus_state_t st = S_HDR1;
static uint8_t rx[DGUS_FRAME_MAX];
static uint8_t rx_len, rx_idx;
static uint32_t rx_last_ms;

void dgus_rx_byte(uint8_t c, uint32_t now)
{
    /* silêncio longo no meio de um quadro: abandona e recomeça */
    if (st != S_HDR1 && (now - rx_last_ms) > DGUS_TIMEOUT_MS)
        st = S_HDR1;
    rx_last_ms = now;

    switch (st) {

    case S_HDR1:
        if (c == DGUS_HDR1) { rx[0] = c; st = S_HDR2; }
        break;

    case S_HDR2:
        if (c == DGUS_HDR2) { rx[1] = c; st = S_LEN; }
        else if (c == DGUS_HDR1) { rx[0] = c; } /* permanece em S_HDR2 */
        else { st = S_HDR1; }
        break;

    case S_LEN:
        if (c < 3 || c > (DGUS_FRAME_MAX - 3)) { st = S_HDR1; break; }
        rx_len = c;
        rx[2] = c;
        rx_idx = 3;
        st = S_BODY;
        break;

    case S_BODY:
        rx[rx_idx++] = c;
        if (rx_idx == (uint8_t)(rx_len + 3)) {
            dgus_dispatch(rx, rx_idx);
            st = S_HDR1;
        }
        break;
    }
}

/* chamada no loop principal quando a recepção é por polling */
void dgus_poll(void)
{
    int c;
    while ((c = port_read()) >= 0)

```

```
    dgus_rx_byte((uint8_t)c, millis());  
}
```



```

/* implementadas pela aplicação – camada 4 */
extern void dgus_on_touch(uint16_t vp, const uint8_t *data, uint8_t nwords);
extern void dgus_on_read (uint16_t vp, const uint8_t *data, uint8_t nwords);

static uint16_t pending_vp    = 0xFFFF;    /* 0xFFFF = nenhuma leitura pendente */
static uint32_t pending_since = 0;

static void dgus_dispatch(const uint8_t *f, uint8_t n)
{
    uint16_t vp;
    uint8_t  nwords;

#ifdef DGUS_USE_CRC
    /* o comprimento inclui os 2 bytes de CRC */
    uint16_t calc = dgus_crc16(&f[3], (uint16_t)(f[2] - 2));
    uint16_t recv = (uint16_t)f[n - 1] << 8 | f[n - 2];    /* byte baixo primeiro */
    if (calc != recv) return;                                /* quadro corrompido */
#endif

    if (f[3] == DGUS_CMD_WRITE) return;    /* 4F 4B – nada a fazer */
    if (f[3] != DGUS_CMD_READ)  return;    /* instrução desconhecida */

    vp    = (uint16_t)f[4] << 8 | f[5];
    nwords = f[6];

    if (vp == pending_vp) {
        pending_vp = 0xFFFF;    /* pedido atendido */
        dgus_on_read(vp, &f[7], nwords);
    } else {
        dgus_on_touch(vp, &f[7], nwords);    /* iniciativa do display */
    }
}

/* atalho para o caso mais comum: uma word */
uint16_t dgus_u16(const uint8_t *data)
{
    return (uint16_t)data[0] << 8 | data[1];
}

```

7. LEITURA ATIVA PELA INSTRUÇÃO 0X83

O toque chega sozinho, mas às vezes o MCU precisa perguntar: ler o estado de um campo de entrada logo depois de trocar de página, recuperar um parâmetro que o operador ajustou antes de o MCU ligar, conferir se uma escrita realmente entrou. Aí a iniciativa é do MCU.

```
/* nwords de 0x01 a 0x7D */
void dgus_read_request(uint16_t vp, uint8_t nwords)
{
    uint8_t f[7 + DGUS_CRC_ROOM];
    f[0] = DGUS_HDR1;
    f[1] = DGUS_HDR2;
    f[2] = 0x04; /* 0x83 + 2 de endereço + 1 de quantidade */
    f[3] = DGUS_CMD_READ;
    f[4] = (uint8_t)(vp >> 8);
    f[5] = (uint8_t)vp;
    f[6] = nwords;

    pending_vp = vp;
    pending_since = millis();
    dgus_send(f, 7);
}
```

Pedir uma word do VP 0x1020

Tx: 5A A5 04 83 10 20 01

Rx: 5A A5 06 83 10 20 01 00 19

Uma leitura por vez. Este driver mantém um só pedido pendente. Disparar dois 0x83 sem esperar a primeira resposta faz o segundo sobrescrever o `pending_vp`, e a resposta do primeiro chega classificada como toque. Para várias leituras, enfileire ou leia um bloco de endereços consecutivos num pedido só.

7.1. LEITURA QUE NÃO VOLTA

Um pedido sem resposta trava a lógica para sempre, se ninguém desistir. O driver precisa expirar a pendência.

```
#define DGUS_READ_TIMEOUT_MS 200

/* chamar a cada volta do loop principal */
void dgus_check_timeout(void)
{
    if (pending_vp != 0xFFFF && (millis() - pending_since) > DGUS_READ_TIMEOUT_MS) {
        uint16_t vp = pending_vp;
        pending_vp = 0xFFFF;
        dgus_on_read_timeout(vp); /* a aplicação decide: repetir, alarmar, seguir */
    }
}
```

Leitura que nunca volta tem três causas prováveis, nessa ordem: o VP pedido está na faixa reservada do sistema; a instrução foi montada com comprimento errado e o display descartou o quadro; ou o display está em modo de download e não responde a nada.

8. CRC-16 OPCIONAL

O CRC é desligado de fábrica e se habilita no arquivo de configuração do display. Ligar significa **ligar nos dois lados**: com o CRC ativo no display, todo quadro sem CRC é descartado sem aviso — o sintoma é um display que simplesmente para de responder depois de uma alteração no CFG.

O polinômio é $x^{16} + x^{15} + x^2 + 1$, o mesmo CRC-16 do Modbus RTU: inicialização em 0xFFFF e polinômio refletido 0xA001. Ele cobre a instrução e os dados — não o cabeçalho nem o byte de comprimento — e vai no fim do quadro com o **byte baixo primeiro**, ao contrário dos dados, que são big-endian.

```
uint16_t dgus_crc16(const uint8_t *p, uint16_t n)
{
    uint16_t crc = 0xFFFF;
    uint8_t i;

    while (n--) {
        crc ^= *p++;
        for (i = 0; i < 8; i++)
            crc = (crc & 1) ? (uint16_t)((crc >> 1) ^ 0xA001) : (uint16_t)(crc >> 1);
    }
    return crc;
}

/* ponto único de saída: acrescenta o CRC quando habilitado.
   Todas as funções de envio passam por aqui.
   O buffer recebido precisa ter DGUS_CRC_ROOM bytes livres no fim. */
static void dgus_send(uint8_t *f, uint8_t n)
{
    #if DGUS_USE_CRC
        uint16_t crc;
        f[2] += 2;                                /* comprimento inclui o CRC */
        crc = dgus_crc16(&f[3], (uint16_t)(n - 3));
        f[n++] = (uint8_t)crc;                      /* byte baixo primeiro */
        f[n++] = (uint8_t)(crc >> 8);
    #endif
    port_write(f, n);
}
```

Por isso todas as funções de envio chamam `dgus_send()` em vez de `port_write()`: ligar ou desligar o CRC no projeto passa a ser trocar uma linha de `#define`, e não revisar cada quadro montado. Para que essa promessa se cumpra, **todo buffer de quadro é declarado com a folga `DGUS_CRC_ROOM`** — é por isso que `dgus_write_word()` usa `f[8 + DGUS_CRC_ROOM]` e não `f[8]`. Buffer dimensionado justo para o quadro sem CRC estoura a pilha no instante em que o `#define` muda, e o defeito não aparece na linha serial: aparece como travamento aleatório, longe da causa.

Vale a pena? Em cabo curto e TTL, o CRC raramente muda o resultado — a máquina de estados já rejeita quadro malformado. Em RS485 longo, com inversor de frequência do lado, muda bastante: o erro de um bit num dado passa despercebido sem CRC, e o operador vê um valor errado na tela sem nenhum sinal de que houve falha.

9. CAMADA 4 – APLICAÇÃO

Aqui entra o produto. O padrão que sustenta projeto grande é o **despacho por VP**: uma tabela que liga cada endereço a uma função, em vez de uma cadeia de if que cresce sem fim.

```
/* --- mapa de VP: o contrato entre o projeto DGUS e o firmware --- */
#define VP_TEMP_ATUAL      0x1000  /* MCU escreve */
#define VP_TEMP_AJUSTE     0x1010  /* display escreve, por toque */
#define VP_BOTAO_LIGA      0x1020  /* display escreve, Return Key Code */
#define VP_PAGINA          0x0084  /* interface de sistema: trocar de página */

typedef struct {
    uint16_t vp;
    void (*handler)(uint16_t valor);
} vp_entry_t;

static void on_ajuste(uint16_t v) { setpoint = v; }
static void on_botao (uint16_t v) { if (v == 1) maquina_liga(); else maquina_desliga(); }

static const vp_entry_t vp_table[] = {
    { VP_TEMP_AJUSTE, on_ajuste },
    { VP_BOTAO_LIGA,  on_botao   },
};

void dgus_on_touch(uint16_t vp, const uint8_t *data, uint8_t nwords)
{
    uint8_t i;
    (void)nwords;
    for (i = 0; i < sizeof(vp_table) / sizeof(vp_table[0]); i++) {
        if (vp_table[i].vp == vp) {
            vp_table[i].handler(dgus_u16(data));
            return;
        }
    }
    /* VP não mapeado: ignore em produção, registre em bancada */
}
```

O mapa de VP é o contrato. Mantenha os #define num arquivo só, com os mesmos nomes usados no projeto DGUS. Endereço escrito na mão em dois lugares diferentes é a causa mais comum de "o display não responde": o controle aponta para um VP e o firmware escreve em outro, e nada no sistema acusa o erro.

10. LOOP PRINCIPAL E CADÊNCIA

O display varre os controles num ciclo interno de 20 ms. Escrever mais rápido que isso não produz atualização mais fluida – produz fila e quadro perdido. A cadência de 100 a 200 ms para dados de processo é suficiente, e visualmente indistinguível de algo mais rápido.

A recepção é o oposto: precisa ser drenada em toda volta do loop. Qualquer `delay()` no caminho enche o buffer da UART e perde toque do operador – o sintoma é o botão que "às vezes não pega".

```
void loop(void)
{
    static uint32_t t_dados = 0;
    uint32_t agora = millis();

    dgus_poll();           /* sempre primeiro, sem delay no caminho */
    dgus_check_timeout();

    if (agora - t_dados >= 200) {
        t_dados = agora;
        uint16_t bloco[2] = { (uint16_t)(temperatura * 10), (uint16_t)pressao };
        dgus_write_words(VP_TEMP_ATUAL, bloco, 2);
    }

    controle_processo();    /* lógica da aplicação, sem bloquear */
}
```

10.1. PARTIDA DO SISTEMA

O display leva alguns instantes para iniciar e mostrar a primeira página. Quadro enviado antes disso se perde sem erro – e o sintoma engana, porque a tela funciona e só os dados não aparecem.

```
void setup(void)
{
    port_init();
    delay(500);           /* deixa o display terminar de iniciar */

    dgus_write_word(VP_PAGINA, 0x5A00); /* 0x0084: força a página 0 */
    delay(50);

    dgus_read_request(VP_TEMP_AJUSTE, 1); /* recupera o ajuste que já estava na tela */
}
```

Forçar a página na partida evita um estado confuso: se o MCU reiniciou sozinho, o display continua na página onde o operador estava, sem que o firmware saiba disso.

11. ARMADILHAS

Ordenadas por frequência em atendimento.

Tratar o 4F 4B como dado. A confirmação de escrita tem o mesmo cabeçalho de um quadro de dados. Código que não filtra por instrução interpreta 0x4F4B como valor – e o número 20299 aparece na variável do processo, sem causa aparente.

Contar o comprimento errado. O byte de comprimento conta instrução, dados e CRC; não conta o cabeçalho nem ele próprio. Errar por dois é o erro clássico – e o display descarta o quadro em silêncio, sem responder nada.

Inverter os bytes do dado. O dado é big-endian: byte alto primeiro. Um MCU little-endian que despeja um `uint16_t` na linha manda trocado. O sintoma é inconfundível – escrever 1 mostra 256.

Escrever na faixa reservada. A faixa 0x0000–0x0FFF é a interface de variáveis de sistema, e dentro dela há posições reservadas. Escrever ali não dá erro: reconfigura o hardware. Perda de toque e brilho zerado depois de uma alteração de código quase sempre é isso.

Sobrepôr VP e SP. Os dois dividem o mesmo espaço de 128 KB. Sobreposição corrompe a exibição de forma intermitente e difícil de rastrear. Mantenha VP em 0x1000–0x7FFF e SP em 0x8000–0xFFFF, e reserve 0x10 endereços por controle.

Bloquear o loop. `delay()`, leitura lenta de sensor e conversão A/D em espera ativa fazem o buffer da UART transbordar. O toque do operador é perdido no meio, e o defeito é intermitente – o pior tipo para diagnosticar.

Usar a serial de depuração. Compartilhar a porta do display com o console de *debug* injeta texto na linha. O display ignora o que não é quadro válido, mas o cabeçalho pode aparecer por acidente dentro de uma mensagem de texto.

12. ROTEIRO DE DIAGNÓSTICO

Quando nada funciona, a ordem abaixo isola a causa em poucos passos. Cada teste elimina uma camada.

#	TESTE	SE FALHAR, O PROBLEMA ESTÁ EM
1	Tocar um controle e ver se o display manda algo – qualquer coisa – pela linha	Fiação, nível elétrico, terra comum, ou o controle não tem VP definido
2	Mandar 5A A5 04 83 10 00 01 por um terminal e ver se volta resposta	Baud, formato da porta, ou CRC ligado no display e não no envio
3	Mandar 5A A5 05 82 10 00 00 19 e conferir se chega o 4F 4B	Comprimento mal contado no quadro
4	Conferir se o valor apareceu na tela	O VP do controle no projeto DGUS não é o endereço que você escreveu
5	Repetir o mesmo pelo firmware, em vez do terminal	Montagem do quadro no código, ou ordem dos bytes
6	Deixar rodando alguns minutos com escrita periódica	Loop bloqueado, buffer transbordando, ou cadência abaixo de 20 ms

O passo 2 é o mais valioso: um terminal serial separa por completo o problema de protocolo do problema de firmware. Se o quadro digitado à mão funciona e o do código não, a causa está no código – e o inverso também vale.

13. FONTES

- **T5L_DGUSII Application Development Guide V2.9** – DWIN Technology. Estrutura do quadro, instruções 0x82 e 0x83, interface de variáveis de sistema, offsets de descritor SP.
- **DWIN T5L Smart LCMs' UART C51 Program** – DWIN Technology. Implementação de referência da fábrica. Roda no núcleo OS do display, não no MCU externo; a máquina de estados de recepção deste documento segue a estrutura dela.
- **Development Guide of T5L ASIC** – DWIN Technology. Formatos aceitos por UART e configuração de porta.