

AGT DISPLAYS

Programação do Núcleo OS

Código C51 dentro do display: UARTs, interrupções e protocolo

ESCOPO

Código que roda no próprio display, não no MCU externo

PRÉ-REQUISITO

Projeto DGUS funcionando e protocolo serial conhecido

LINGUAGEM

C51 sobre o núcleo 8051 do ASIC, compilado no Keil

APLICAÇÃO

Módulos com ASIC DWIN rodando DGUS II

1. O NÚCLEO OS

O ASIC tem **dois núcleos**. O núcleo GUI desenha a tela, lê o toque e executa o projeto compilado no DGUS — é ele que faz o display ser um display, e o usuário não programa nada nele. O outro é o **núcleo OS**: um 8051 completo, com as periférias todas, que fica à disposição do cliente.

Na arquitetura usual, o núcleo OS fica vazio: a lógica toda mora num MCU externo, que conversa com o display pela serial. Isso funciona bem e é o caminho mais simples. Programar o núcleo OS faz sentido quando a alternativa é pior:

- **Eliminar o MCU externo.** Em produto com pouca lógica, o núcleo OS faz o trabalho todo — o display passa a ser o controlador. Menos uma placa, menos um firmware, menos uma fonte.
- **Usar as periférias do display.** O núcleo OS alcança IO, AD, PWM, CAN e as UARTs livres. Ler um sensor analógico e mostrar o valor não precisa sair do módulo.
- **Aliviar um MCU que não dá conta.** Deslocar a formatação de tela e a varredura de campos para dentro do display libera o MCU externo para o que é tempo-crítico.
- **Falar dois protocolos.** O núcleo OS tem quatro UARTs. Um produto pode atender um mestre Modbus por uma porta e comandar um instrumento por outra, sem MCU no meio.

Escolha que não se desfaz de graça. O núcleo OS guarda **um programa só**: ou o firmware DWIN OS com o seu programa em assembly DWIN OS, ou o seu código C51. Nunca os dois. Trocar depois exige regravar o núcleo. Decida no início do projeto.

O 8051 do ASIC não é o 8051 de catálogo. A DWIN expandiu o barramento de SFR, acrescentou processador matemático em hardware e otimizou o acesso à memória: na prática ele executa de **130 a 150 instruções por microssegundo**. Para lógica de IHM, sobra folga.

2. DOIS NÚCLEOS, UMA MEMÓRIA

O que faz o núcleo OS valer a pena é simples: **os dois núcleos compartilham o espaço de variáveis**. O mesmo VP que o controle da tela exibe é alcançável pelo código C51, por duas funções:

```
u16 Read_Dgus (u16 vp);           /* lê uma word do espaço de variáveis */
void Write_Dgus(u16 vp, u16 valor); /* escreve uma word no mesmo espaço */

/* para blocos, em bytes: */
void write_dgus_vp(u16 addr, u8 *buf, u16 len);
void read_dgus_vp (u16 addr, u8 *buf, u16 len);
void Write_Dgusii_Vp_byChar(u16 addr, u8 *buf, u16 len);
```

Com isso, exibir um valor na tela deixa de ser um problema de comunicação. Não há quadro para montar, nem cabeçalho, nem CRC, nem espera de resposta: escreve-se no VP e o núcleo GUI mostra na varredura seguinte.

```
/* Mostrar 25,3 °C num Data Variables em 0x1000, com uma casa decimal */
Write_Dgus(0x1000, 253);

/* Ler o que o operador digitou num campo de entrada em 0x1010 */
u16 ajuste = Read_Dgus(0x1010);

/* Trocar de página pela interface de sistema (0x0084) */
Write_Dgus(0x0084, 0x5A03); /* vai para a página 3 */
```

A leitura descartada dentro de Write_Dgus() não é sobra. O barramento compartilhado grava em blocos de **duas words**, e uma escrita de word única toca só metade dos registradores de dado. A leitura feita antes carrega a outra metade, para que ela seja reescrita com o próprio valor. Remover esse passo – é a primeira coisa que parece otimizável – **zera a word vizinha** a cada escrita, com efeito aparentemente aleatório na tela.

A faixa 0x0000–0x0FFF continua sendo a interface de variáveis de sistema, com as mesmas regras e as mesmas posições reservadas. Escrever ali pelo C51 tem exatamente o mesmo efeito que escrever pela serial – inclusive os efeitos indesejados.

O ciclo de 20 ms continua valendo. O núcleo GUI varre os controles a cada 20 ms. Escrever no mesmo VP mais rápido que isso não deixa a tela mais fluida: só o último valor antes da varredura chega a ser exibido. Vale para o C51 tanto quanto para o MCU externo.

2.1. O FRAMEWORK DE BASE

As funções `Read_Dgus()` e `Write_Dgus()`, as estruturas de buffer das UARTs, a tabela de CRC e o código de inicialização do núcleo fazem parte do **framework de base**, que não se escreve do zero — ele acompanha o ambiente de desenvolvimento.

Framework disponível no site da AGT Displays. O **projeto Keil** de partida — com o header de SFR do ASIC, as funções de acesso ao espaço de variáveis e a estrutura de buffer das UARTs já montada — está na área de suporte. Os trechos deste documento presumem esse projeto como ponto de partida.

3. AS CINCO UARTS

O ASIC tem cinco portas seriais, e a primeira coisa a entender é que uma delas não está disponível.

PORTA	DISPONÍVEL AO C51?	OBSERVAÇÃO
UART1	Não	Pertence ao núcleo GUI. Reservada para download, depuração online e módulo WiFi. Na placa de avaliação é a porta USB
UART2	Sim	Multiplexada com IO – precisa ser habilitada por MUX_SEL. É também a porta de depuração do DGUS
UART3	Sim	Multiplexada com IO – precisa ser habilitada por MUX_SEL
UART4	Sim	Dedicada. Em muitos módulos sai como RS485
UART5	Sim	Dedicada

A UART2 e a UART3 dividem pinos com IO de uso geral: sem escrever o bit correspondente em MUX_SEL, o pino continua sendo IO e a porta simplesmente não aparece. A UART4 e a UART5 são dedicadas e não têm essa etapa.

Fora disso, o uso das quatro é praticamente igual. Quem entende uma, entende as outras – as diferenças estão nos nomes dos registradores, nas fórmulas de baud e em um detalhe de aninhamento de interrupção, todos tabelados adiante.

3.1. ONDE AS PORTAS APARECEM NO PRODUTO

Quais portas estão fisicamente acessíveis e em que nível elétrico depende do módulo. O datasheet é a fonte – mas o padrão por tipo de construção ajuda a saber o que esperar.

TIPO	O QUE COSTUMA SAIR	COMO ACESSAR
Placa nua (COB)	UART2 e UART4 no conector de sinal, em geral RS232	Conector do módulo. RX do display no TX do outro lado, TX no RX, mais GND e VCC
Com gabinete	UART2 e UART5 em RS232, UART4 em RS485, às vezes CAN	Bornes identificados. No RS485, A/485+ com 485+ e B/485- com 485-
COF	UART2, UART4, UART5, IO, CAN – tudo no flat	Placa adaptadora HDL662S dá acesso conveniente a todas
Placa de avaliação	Todas, mais IO, AD e PWM	Barra de pinos. A USB é a UART1 e serve para depuração online

"Só após configuração de OS". O datasheet marca assim a UART4 e a UART5 na maioria dos módulos. A frase significa exatamente o assunto deste documento: a porta existe no silício e no conector, mas só passa a funcionar quando há programa no núcleo OS que a inicialize. Sem isso, o borne está morto.

4. CLOCK E BAUD RATE

Todas as fórmulas de baud partem do clock do núcleo, que não é a frequência do cristal:

$$FOSC = \text{frequência do cristal} \times 56 / 3$$

$$\text{Cristal de 11,0592 MHz} \rightarrow FOSC = 206,4384 \text{ MHz}$$

A partir daí, cada família de porta usa um divisor diferente. Essa é a principal fonte de erro de quem migra código de uma UART para outra: a conta muda.

PORTA	REGISTRADOR	FÓRMULA
UART2	SREL0H:L	$1024 - FOSC / (64 \times \text{baud})$ – com SMOD = 0
UART2	SREL0H:L	$1024 - FOSC / (32 \times \text{baud})$ – com SMOD = 1
UART3	SREL1H:L	$1024 - FOSC / (32 \times \text{baud})$
UART4	BODE2_DIV_H:L	$FOSC / (8 \times \text{baud})$ – sem o complemento de 1024
UART5	BODE3_DIV_H:L	$FOSC / (8 \times \text{baud})$ – sem o complemento de 1024

Valores calculados para cristal de 11,0592 MHz, prontos para conferência:

BAUD	UART2 (SMOD 0)	UART3	UART4 E UART5
9600	0x02B0	0x0160	0x0A80
19200	0x0358	0x02B0	0x0540
38400	0x03AC	0x0358	0x02A0
57600	0x03C8	0x0390	0x01C0
115200	0x03E4	0x03C8	0x00E0
230400	0x03F2	0x03E4	0x0070
921600	—	0x03F9	0x001C

Repare em 115200: o valor é 0x03E4 na UART2 e 0x00E0 na UART4. Copiar a constante de uma porta para outra produz uma taxa completamente diferente, e o sintoma é lixo na linha em vez de silêncio – o que engana.

5. REGISTRADORES

5.1. UART2, EM DETALHE

A UART2 segue o modelo do 8051 padrão e é endereçável por bit, o que a torna a mais confortável de programar. As demais seguem o mesmo padrão com nomes diferentes.

SFR	ENDEREÇO	FUNÇÃO
MUX_SEL	0xC9	Bit .6 : 1 = UART2 sai em P0.4 e P0.5; 0 = os pinos ficam como IO
SCON0	0x98	Controle, endereçável por bit. .7 SM0 · .6 SM1 · .5 SM2 (multiprocessador) · .4 RENO (habilita recepção) · .3 TB80 · .2 RB80 · .1 TI0 (marca de envio) · .0 RI0 (marca de recepção)
SBUF0	0x99	Dado — um registrador para enviar e receber
ADCON	0xD8	Origem do baud: 0x00 = temporizador T1 (como no 8051 padrão); 0x80 = SREL0H:L , que libera o T1 para outro uso
PCON	0x87	Bit .7 SMOD: 0 = sem dobra de taxa, 1 = taxa dobrada
SREL0H	0xBA	Divisor de baud, byte alto
SREL0L	0xAA	Divisor de baud, byte baixo
P0MDOUT	0xB7	Configura o pino de TX como <i>push-pull</i> . Sem isso a saída não aciona a linha

5.2. TABELA DE DIFERENÇAS

O que muda nas outras três portas. A lógica do código é a mesma; trocam-se os nomes.

	UART2	UART3	UART4	UART5
Habilitar pino	MUX_SEL . 6	MUX_SEL . 5 (P0.6, P0.7)	dedicada	dedicada
Controle	SCON0 0x98	SCON1 0x9B	SCON2T 0x96 SCON2R 0x97	SCON3T 0xA7 SCON3R 0xAB
Endereçável por bit?	Sim	Não	Não	Não
Dado	SBUF0 0x99	SBUF1 0x9C	SBUF2_TX 0x9E SBUF2_RX 0x9F	SBUF3_TX 0xAC SBUF3_RX 0xAD
Baud	SREL0H 0xBA SREL0L 0xAA	SREL1H 0xBB SREL1L 0x9D	BODE2_DIV_H 0xD9 BODE2_DIV_L 0xE7	BODE3_DIV_H 0xAE BODE3_DIV_L 0xAF
Habilitar interrupção	IEN0 . 4 (ES0)	IEN2 . 0 (ESI)	IEN1 . 3 ES2R (rx) IEN1 . 2 ES2T (tx)	IEN1 . 5 ES3R (rx) IEN1 . 4 ES3T (tx)
Vetor	0x0023 interrupt 4	0x0083 interrupt 16	tx 0x0053 int 10 rx 0x005B int 11	tx 0x0063 int 12 rx 0x006B int 13
Envio e recepção	Mesma ISR	Mesma ISR	ISRs separadas	ISRs separadas
Disparar envio	TI0 = 1	SCON1 = 0x02	SCON2T = 0x01	SCON3T = 0x01
Direção RS485	—	—	TR4 = P0.0	TR5 = P0.1

Os nomes dependem do framework. Os **endereços** acima são do silício e não mudam. Os **nomes**, sim: o header oficial da DWIN nomeia os registradores pelo índice interno da porta (UART4 → SCON2T, UART5 → SCON3T), enquanto frameworks alternativos costumam renomear pelo número comercial (UART4 → SCON4T, UART5 → SCON5T) — mais legível, e sem ambiguidade. Confira o header do seu projeto antes de copiar código de outra fonte. A §5.3 traz a equivalência.

5.3. EQUIVALÊNCIA ENTRE CONVENÇÕES DE NOME

Mesmo endereço, nomes diferentes. Traduzir código de um framework para o outro é substituição direta.

PORTA	ENDEREÇO	ÍNDICE INTERNO (HEADER DWIN)	NÚMERO COMERCIAL (ALTERNATIVO)
UART2	0x98	SCON0	SCON2
	0x99	SBUF0	SBUF2
	0xBA / 0xAA	SREL0H / SREL0L	SREL2H / SREL2L
	IEN0.4 · SCON.1/.0	ES0 · TI0 / RI0	ES2 · TI2 / RI2
UART3	0x9B	SCON1	SCON3
	0x9C	SBUF1	SBUF3
	0xBB / 0x9D	SREL1H / SREL1L	SREL3H / SREL3L
UART4	0x96 / 0x97	SCON2T / SCON2R	SCON4T / SCON4R
	0x9E / 0x9F	SBUF2_TX / SBUF2_RX	SBUF4_TX / SBUF4_RX
	0xD9 / 0xE7	BODE2_DIV_H / _L	BODE4_DIV_H / _L
	IEN1.3 / .2	ES2R / ES2T	ES4R / ES4T
UART5	0xA7 / 0xAB	SCON3T / SCON3R	SCON5T / SCON5R
	0xAC / 0xAD	SBUF3_TX / SBUF3_RX	SBUF5_TX / SBUF5_RX
	0xAE / 0xAF	BODE3_DIV_H / _L	BODE5_DIV_H / _L
	IEN1.5 / .4	ES3R / ES3T	ES5R / ES5T

A confusão que a primeira convenção produz é concreta: SCON3T é a UART5 no header da DWIN, e a UART3 não tem registrador com esse nome. Os trechos deste documento usam a convenção do header oficial.

O SCON1 pede escrita dupla. A UART3 não é endereçável por bit, e limpar uma marca no SCON1 exige **duas escritas consecutivas** – SCON1 &= 0xFD; repetido. A primeira sozinha não pega. É particularidade do registrador, não erro de digitação.

6. INTERRUPTÕES

O núcleo OS tem **12 interrupções**, organizadas em **6 grupos de duas**. Dentro do grupo a ordem é fixa; entre grupos, configurável.

GRUPO	BIT EM IP0 / IP1	PRIORIDADE ALTA NO GRUPO	PRIORIDADE BAIXA NO GRUPO
G0	.0	Externa 0	UART3
G1	.1	Temporizador T0	CAN
G2	.2	Externa 1	UART4 envio
G3	.3	Temporizador T1	UART4 recepção
G4	.4	UART2	UART5 envio
G5	.5	Temporizador T2	UART5 recepção

Entre grupos há quatro níveis, formados pelo par de bits correspondente em IP1 e IP0:

NÍVEL	IP1	IP0
3 – mais alto	1	1
2	1	0
1	0	1
0 – mais baixo	0	0

Com $IP0 = IP1 = 0x00$ – a configuração de partida –, vale a ordem natural: G0 é o grupo mais prioritário e G5 o menos. A escala completa, do mais para o menos prioritário, é EX0 • UART3 • T0 • CAN • EX1 • UART4-TX • T1 • UART4-RX • UART2 • UART5-TX • T2 • UART5-RX.

Os registradores de habilitação:

SFR	ENDEREÇO	BITS
IEN0	0xA8	.7 EA, chave geral · .6 escrever 0 · .5 ET2 · .4 ES0 (UART2) · .3 ET1 · .2 EX1 (P3.1) · .1 ET0 · .0 EX0 (P3.0). Endereçável por bit
IEN1	0xB8	.7 – .6 escrever 0 · .5 ES3R (UART5 rx) · .4 ES3T (UART5 tx) · .3 ES2R (UART4 rx) · .2 ES2T (UART4 tx) · .1 ECAN · .0 escrever 0. Endereçável por bit
IEN2	0x9A	.7 – .1 escrever 0 · .0 ESI (UART3)
IEN3	0xD1	Escrever 0x00
IP0	0xA9	Prioridade, byte 0
IP1	0xB9	Prioridade, byte 1

Não aninhe interrupção. O núcleo permite até quatro níveis de aninhamento, mas a recomendação da DWIN é não usar: complica a arquitetura sem ganho real, porque o núcleo já é rápido e as rotinas são curtas. O padrão recomendado é **desligar tudo na entrada e religar na saída** – EA = 0 no começo da ISR, EA = 1 no fim. Simples e suficiente.

Em todas as portas, **a marca de interrupção é limpa por software**. Esquecer de limpar deixa a ISR sendo chamada sem parar, e o efeito visível é o display travar por completo – o núcleo GUI continua desenhando, mas nada mais responde.

7. ANATOMIA DO DRIVER

O driver de referência tem quatro peças, e vale entender o papel de cada uma antes de olhar o código. Os exemplos são da UART2; para as outras, aplique a tabela de diferenças da §5.2.

PEÇA	ONDE RODA	PAPEL
Inicialização	Partida	Habilita o pino, calcula o baud, liga a interrupção, zera os buffers
ISR	Interrupção	Move um byte entre o SFR e o buffer circular. Nada além disso
Máquina de estados	Laço principal	Consome o buffer circular byte a byte e reconhece quadros completos
Despacho	Laço principal	Executa o 0x82 ou o 0x83 sobre o espaço de variáveis e responde

A ISR só move bytes. É a regra que sustenta o resto. Montar quadro, calcular CRC ou escrever no espaço de variáveis dentro da ISR trava a porta durante o processamento e perde byte na taxa alta. A interrupção enche o buffer; o laço principal interpreta.

7.1. INICIALIZAÇÃO

```
#if UART2_ENABLE
void Uart2_Init(u32 baud_rate)
{
    u16 i;

    Uart2.id      = 2;
    Uart2.Tx_Read = 0;
    Uart2.Tx_Write = 0;
    Uart2.Tx_Busy = 0;
    Uart2.Rx_Read  = 0;
    Uart2.Rx_Write = 0;
    Uart2.Rx_Busy  = 0;
    Uart2.Rx_Flag  = Uart_Rev_Pre;

    for (i = 0; i < Uart2_Tx_Lenth; i++) Uart2.Tx_Buffer[i] = 0;
    for (i = 0; i < Uart2_Rx_Lenth; i++) Uart2.Rx_Buffer[i] = 0;

    i = 1024 - FOSC / 64 / baud_rate;    /* divisor – ver §4 */

    MUX_SEL |= 0x40;          /* .6: leva a UART2 para P0.4 e P0.5 */
    P0MDOUT &= 0xCF;          /* limpa a configuração dos dois pinos */
    P0MDOUT |= 0x10;          /* TX em push-pull */

    ADCON  = 0x80;             /* baud vem de SREL0, não do T1 */
    SCON0  = 0x50;             /* 8 bits, recepção habilitada */
    SREL0H = (u8)(i >> 8);
    SREL0L = (u8)i;

    ES0 = 1;                   /* habilita a interrupção da UART2 */
    EA  = 1;                   /* chave geral */
}
#endif
```

Os blocos `#if UARTn_ENABLE` são a forma de manter as quatro portas no mesmo projeto e compilar só as usadas. Cada buffer custa RAM, e a RAM do núcleo OS é limitada.

7.2. A ISR

Na UART2 uma só ISR atende envio e recepção, distinguidos pelas marcas **RI0** e **TI0**.

```

void Uart2_TX_RX_ISR(void) interrupt 4
{
    EA = 0;

    if (RI0 == 1) {                /* --- byte recebido --- */
        RI0 = 0;                   /* limpa a marca, sempre */
        Uart2.Rx_Buffer[Uart2.Rx_Write] = SBUF0;
        Uart2.Rx_Write++;
        Uart2.Rx_Write %= Uart2_Rx_Lenth;

    } else if (TI0 == 1) {         /* --- byte enviado --- */
        TI0 = 0;
        if (Uart2.Tx_Read != Uart2.Tx_Write) {
            SBUF0 = Uart2.Tx_Buffer[Uart2.Tx_Read];
            Uart2.Tx_Read++;
            Uart2.Tx_Read %= Uart2_Tx_Lenth;
        } else {
            Uart2.Tx_Busy = 0;      /* fila vazia: envio encerrado */
        }
    }

    EA = 1;
}

```

O envio funciona em cascata: a aplicação enfileira os bytes, e a primeira interrupção é disparada *por software*. Cada byte transmitido gera a interrupção que busca o próximo, até a fila esvaziar. A variável `Tx_Busy` é o que evita disparar uma segunda cascata sobre a primeira.

```

void Uart2_Tx_write2buff(u8 dat)
{
    Uart2.Tx_Buffer[Uart2.Tx_Write] = dat;

    if (Uart2.Tx_Read > Uart2.Tx_Write) {
        /* buffer cheio: espera o ponteiro de leitura avançar */
        while ((Uart2.Tx_Write) + 1 == Uart2.Tx_Read);
    }

    ++Uart2.Tx_Write;
    if (Uart2.Tx_Write >= Uart2_Tx_Lenth) {
        while (Uart2.Tx_Read == 0);    /* não atropela a cabeça */
        Uart2.Tx_Write = 0;           /* dá a volta */
    }

    if (Uart2.Tx_Busy == 0) {
        Uart2.Tx_Busy = 1;
        TI0 = 1;                      /* dispara a cascata */
    }
}

```

7.3. MÁQUINA DE ESTADOS DE QUADRO

A UART entrega byte a byte, sem noção de quadro. Reconhecer o quadro é trabalho da máquina de estados, que roda no laço principal e só entrega ao despacho o que está completo.

ESTADO	ESPERA	SE NÃO CONFERE
Uart_Rev_Pre	Primeiro byte do cabeçalho (0x5A)	Descarta e permanece
Uart_Rev_GetFH1	Segundo byte do cabeçalho (0xA5)	Volta ao início
Uart_Rev_GetFH2	Comprimento, entre 3 e o limite do buffer	Volta ao início – comprimento impossível denuncia falso cabeçalho
Uart_Rev_GetLen	Instrução: 0x82 ou 0x83	Volta ao início – exceto 0x80 e 0x81, ver nota abaixo
Uart_Rev_GetCode ... GetAddr2	Endereço e dados, até completar o comprimento	Quadro pronto → Uart_Rev_Done

O gancho morto de 0x80 e 0x81. No estado Uart_Rev_GetLen, o driver de referência também aceita as instruções 0x80 e 0x81 – acrescenta 2 ao comprimento esperado e salta direto para o estado de dados, sem passar pela leitura de endereço. O quadro é montado até o fim e então **descartado em silêncio**: o despachante só trata 0x82 e 0x83. Quem espelhar essa máquina de estados herda o gancho inacabado. Ou complete o tratamento, ou recuse as duas instruções no estado de comprimento – deixar como está gasta ciclo e esconde erro de protocolo.

A validação do comprimento é a parte que mais rende. Um 0x5A 0xA5 que aparece por acaso dentro de dados é seguido por um byte qualquer; se esse byte não for um comprimento plausível, a máquina volta ao início imediatamente, em vez de ficar dezenas de bytes perdida esperando um quadro que não existe.

```

/* trecho central: consome o buffer circular e monta o quadro */
while (((*Uart_number).Rx_Write - (*Uart_number).Rx_Read + Uart_buff_lenth)
      % Uart_buff_lenth) {

    EA = 0;
    c = (*Uart_number).Rx_Buffer[(*Uart_number).Rx_Read++];
    (*Uart_number).Rx_Read %= Uart_buff_lenth;
    EA = 1;

    if ((*Uart_number).Rx_Flag == Uart_Rev_Pre) {
        (*Uart_data).varIndex = 0;
        if (c == DTHD1) { /* 0x5A */
            (*Uart_number).Rx_Flag = Uart_Rev_GetFH1;
            (*Uart_data).varData[(*Uart_data).varIndex++] = c;
        }

    } else if ((*Uart_number).Rx_Flag == Uart_Rev_GetFH1) {
        if (c == DTHD2) { /* 0xA5 */
            (*Uart_number).Rx_Flag = Uart_Rev_GetFH2;
            (*Uart_data).varData[(*Uart_data).varIndex++] = c;
        } else {
            (*Uart_number).Rx_Flag = Uart_Rev_Pre;
        }
    }

    } else if ((*Uart_number).Rx_Flag == Uart_Rev_GetFH2) {
        if (c < 3 || c > Uartx_Frame_Data_Lenth) {
            (*Uart_number).Rx_Flag = Uart_Rev_Pre; /* comprimento inválido */
        } else {
            (*Uart_data).dataLen = c;
            (*Uart_data).varData[(*Uart_data).varIndex++] = c;
            (*Uart_number).Rx_Flag = Uart_Rev_GetLen;
        }
    }
    /* ... demais estados: instrução, endereço, dados ... */
}

if ((*Uart_number).Rx_Flag == Uart_Rev_Done) {
    (*Uart_number).Rx_Flag = Uart_Rev_Pre;
    deal_uart_data((*Uart_data).varData, Uart_num, Response, Crc_ck);
}

```

7.4. DESPACHO

Com o quadro completo, resta separar escrita de leitura. Aqui está a inversão de papéis: é o display que **atende** o pedido.

```

void deal_uart_data(u8* Arr_rece, u8 Uart_num, bit Response, bit Crc_ck)
{
    if ((Arr_rece[0] == DTHD1) && (Arr_rece[1] == DTHD2)) {

        if (Arr_rece[3] == 0x82) {                /* escrita recebida */
            Response_flog = Response;
            Crc_check_flog = Crc_ck;
            deal_82_cmd(Uart_num, Arr_rece);

        } else if (Arr_rece[3] == 0x83) {        /* leitura pedida */
            u8 Val[Uartx_Frame_Data_Lenth] = {0};
            Crc_check_flog = Crc_ck;
            deal_83_cmd(Uart_num, Val, Arr_rece);
        }
    }
}

```

O tratamento do 0x82 escreve no espaço de variáveis e, se a resposta estiver habilitada, devolve a confirmação:

```

/* sem CRC: o dado começa em Arr[6] e tem Arr[2]-3 bytes */
Write_Dgusii_Vp_byChar((arr[4] << 8) + arr[5], arr + 6, arr[2] - 3);

if (Response_flog) {
    u8 Temp_arr[] = { DTHD1, DTHD2, 0x03, 0x82, 0x4F, 0x4B };
    uart_send_str(Uart, Temp_arr, 6);
}

/* com CRC: dois bytes a menos de dado, e a resposta leva CRC
5A A5 05 82 4F 4B A5 EF */

```

O tratamento do 0x83 lê o bloco pedido e devolve no mesmo formato do quadro de entrada, com o comprimento recalculado – $2 \times (\text{n}^\circ \text{ de words}) + 4$, mais 2 quando há CRC:

```

for (i = 0; i < 7; i++) arr[i] = arr1[i];          /* copia o cabeçalho recebido */

Read_Dgusii_Vp((arr[4] << 8) + arr[5], (u8*)&arr[7], arr[6]);
arr[2] = (2 * arr[6]) + 4;

uart_send_str(Uart, arr, arr[2] + 3);

```

A resposta é opcional, e às vezes é melhor desligá-la. O sinalizador Response controla se a escrita devolve 4F 4B. Desligar resolve o caso do MCU externo que não filtra a confirmação e a interpreta como dado – sem tocar no firmware dele.

8. ENVIO POR INICIATIVA DO DISPLAY

Até aqui o display respondeu a pedidos. Mas o caso mais interessante é o inverso: o display avisa o dispositivo externo de que algo mudou, sem ser perguntado. É isso que elimina a necessidade de o outro lado ficar consultando.

O mecanismo usa **duas words da interface de sistema como caixa de pedido**: 0x0F00 e 0x0F01. O projeto DGUS escreve o que quer enviar nesses dois endereços, e o C51 – que verifica a caixa a cada volta do laço – monta o quadro e transmite.

ENDEREÇO	BYTE ALTO	BYTE BAIXO
0x0F00	0x5A – marca de pedido pendente	Byte alto do VP a enviar
0x0F01	Byte baixo do VP a enviar	Quantidade de words

```

void Read_0xF00(void)
{
    u16 Va = Read_Dgus(0x0F00);
    u16 V1 = Read_Dgus(0x0F01);

    if (((u8)(Va >> 8)) == 0x5A) {          /* há pedido pendente */
        u8 i = 0;
        u16 Temp = 0;
        u8 Val[100] = {0};                /* 5A A5 len 83 ADDR Len XX XX */

        Val[0] = DTHD1;
        Val[1] = DTHD2;
        Val[2] = (((u8)V1) << 1) + 4;      /* 2 x words + 4 */
        Val[3] = 0x83;
        Val[4] = (u8)Va;                   /* VP, byte alto */
        Val[5] = (u8)(V1 >> 8);            /* VP, byte baixo */
        Val[6] = (u8)V1;                   /* nº de words */

        for (i = 0; i < (u8)V1; i++) {
            Temp = Read_Dgus(((Val[4] << 8) + Val[5] + i));
            Val[7 + 2 * i] = (u8)(Temp >> 8);
            Val[8 + 2 * i] = (u8)(Temp);
        }

        #if UART2_ENABLE
        uart_data_send(Val, 2, DATA_UPLOAD_UART2, CRC_CHECK_UART2);
        #endif
        /* ... idem para UART3, UART4, UART5 ... */

        Write_Dgus(0x0F00, 0);             /* limpa a caixa de pedido */
        Write_Dgus(0x0F01, 0);
    }
}

```

Limpar as duas words no fim é o que fecha o ciclo: sem isso o mesmo bloco seria enviado sem parar, a cada volta do laço.

Por que isso é útil. O quadro enviado tem o formato de uma resposta 0x83 — ou seja, é indistinguível, para quem recebe, de uma leitura atendida. Um MCU externo que já saiba tratar 0x83 passa a receber notificação do display sem uma linha de código nova. E o disparo pode vir de um controle de toque no próprio projeto DGUS: o operador aperta um botão, o botão escreve na caixa de pedido, o bloco vai para a linha.

9. RS485: CONTROLE DE DIREÇÃO

RS485 é meio-duplex: o mesmo par de fios serve para enviar e receber, e um transceptor por vez pode acionar a linha. Nas portas que saem em RS485 — tipicamente a UART4 — o núcleo OS controla o sentido por um pino de IO: **TR4 é o P0.0** e **TR5 é o P0.1**. Não são bits de registrador de UART, e sim saídas comuns ligadas ao pino de direção do transceptor — por isso precisam ser configuradas como *push-pull* em P0MDOUT na inicialização, como qualquer outra saída.

A regra tem dois pontos, e ambos precisam estar certos:

```
/* 1) ao enfileirar o primeiro byte, assume a linha */
if (Uart4.Tx_Busy == 0) {
    Uart4.Tx_Busy = 1;
    TR4 = 1;                /* habilita o transmissor RS485 */
    SCON2T |= 0x01;         /* dispara a cascata de envio */
}

/* 2) na ISR de envio, ao esvaziar a fila, libera a linha */
if (Uart4.Tx_Read != Uart4.Tx_Write) {
    SBUF2_TX = Uart4.Tx_Buffer[Uart4.Tx_Read];
    Uart4.Tx_Read++;
    Uart4.Tx_Read %= Uart4_Tx_Lenth;
} else {
    TR4 = 0;                /* devolve a linha à escuta */
    Uart4.Tx_Busy = 0;
}
```

Soltar a linha cedo ou tarde. Baixar TR4 antes do último byte terminar de sair corta o fim do quadro, e o outro lado vê comprimento inconsistente. Não baixar deixa o display segurando o barramento: nenhum outro dispositivo consegue falar, e o sintoma é uma rede RS485 que "morre" depois que o display transmite. A liberação tem de ficar no ramo da ISR que constata fila vazia — nunca na função que enfileira.

10. LAÇO PRINCIPAL

Com as peças no lugar, o laço fica curto. A máquina de estados de cada porta habilitada é chamada por uma função única:

```
void Uart_Handle_all(void)
{
    Uart_Handle_Frame(&Uart2, &Uart2_Frame, 2, Uart2_Rx_Lenth,
                      RESPONSE_UART2, CRC_CHECK_UART2);
    Uart_Handle_Frame(&Uart3, &Uart3_Frame, 3, Uart3_Rx_Lenth,
                      RESPONSE_UART3, CRC_CHECK_UART3);
    Uart_Handle_Frame(&Uart4, &Uart4_Frame, 4, Uart4_Rx_Lenth,
                      RESPONSE_UART4, CRC_CHECK_UART4);
    Uart_Handle_Frame(&Uart5, &Uart5_Frame, 5, Uart5_Rx_Lenth,
                      RESPONSE_UART5, CRC_CHECK_UART5);
}

void main(void)
{
    Uart2_Init(115200);
    /* ... inicializa as outras portas usadas ... */

    while (1) {
        Uart_Handle_all();      /* atende o que chegou          */
        Read_0xF00();           /* verifica a caixa de pedido */
        aplicacao();             /* a lógica do produto         */
    }
}
```

A aplicação não deve bloquear. Espera ativa longa no laço principal enche o buffer circular e perde byte – o mesmo problema do MCU externo, com o agravante de que aqui o núcleo GUI continua funcionando, então a tela parece normal e só a comunicação falha.

11. COMPILAR E GRAVAR

O núcleo OS não se programa pelo DGUS. Há dois caminhos, e eles diferem justamente na etapa que costuma travar o iniciante.

11.1. KEIL MVISION

ETAPA	FERRAMENTA	RESULTADO
1. Compilar	Keil C51, sobre o projeto de base	Arquivo em formato HEX
2. Converter	DownLoadFor8051 , da DWIN	T5L51.bin — o HEX não serve para gravar
3. Gravar	UART Download Tool do DGUS, ou cartão SD	Código no núcleo OS do display

A etapa 2 é obrigatória e é onde a maioria trava na primeira vez: o HEX do Keil não é gravável. O **DownLoadFor8051** converte para o binário que o display aceita, e o nome do arquivo importa — T5L51.bin.

11.2. CADEIAS ABERTAS

O Keil é a recomendação da DWIN, mas é pago e roda só em Windows. O núcleo do ASIC é um 8051, então nada impede compilar com **SDCC**, que é livre e multiplataforma.

Montar um projeto SDCC do zero para o ASIC dá trabalho — o código de inicialização, os vetores de interrupção, os segmentos e o mapa de memória têm de ser escritos à mão. Há um projeto público que já resolve isso: o **T5L + SDCC Starter Template**, de Recep Şenbaş, em github.com/recepsenbas/DWIN-T5L-SDCC-Template. Traz **Makefile** com detecção automática dos caminhos do SDCC, código de partida com suporte a interrupção, bibliotecas de UART, CRC, temporizador e RTC, relatório de uso de memória ao fim da compilação e binários pré-compilados para as combinações usuais de taxa, CRC e resposta.

A diferença prática está em duas coisas. Primeiro, a **conversão para binário desaparece**: o **makebin** do próprio SDCC produz o T5L51.bin ao fim da compilação, e o DownLoadFor8051 deixa de ser necessário. Segundo, os **nomes dos registradores** costumam seguir o número comercial da porta em vez do índice interno — a equivalência está na §5.3.

A configuração que importa é o modelo de memória: *large*, com a XRAM em 0x8000 e 32 KB de tamanho.

A licença do template SDCC é não-comercial. O projeto é publicado sob **CC BY-NC-SA 4.0**: livre para estudar, avaliar e adaptar, com atribuição — mas a cláusula *NonCommercial* não cobre firmware de produto que se vende. Para uso comercial, o autor mantém uma edição estendida e atende por contato direto. A cadeia SDCC em si é livre e sem restrição: o que é licenciado é aquele projeto de partida, não o compilador.

A gravação é a mesma em qualquer cadeia. O que se grava é sempre um T5L51.bin, pelo mesmo UART Download Tool ou pelo mesmo cartão SD. Trocar de compilador não muda nada do lado do display.

11.3. ORÇAMENTO DE MEMÓRIA

O núcleo OS é um 8051, com os limites de um 8051. Vale conhecê-los antes de projetar, porque estourar o de 256 bytes é fácil e o erro aparece como comportamento errático, não como falha de compilação.

REGIÃO	TAMANHO	O QUE VIVE NELA
CODE	64 KB	O programa. Folgado para lógica de IHM
XDATA	32 KB	Memória externa — é onde ficam os buffers das UARTs
DATA	256 bytes	RAM interna de acesso rápido. É o recurso escasso
IDATA	256 bytes	RAM indireta, compartilhando espaço com a pilha
BIT	256 bits	Variáveis de um bit e os sinalizadores do driver

Dai a razão de os buffers de UART serem declarados em XDATA e de cada porta ficar atrás de um `#if`: compilar as quatro sem precisar consome RAM que vai faltar na aplicação. Um projeto de IHM típico fica em torno de 30% do CODE e 20% do XDATA — mas o DATA passa de 40% com facilidade.

Para gravar pelo DGUS, a aba *TSL Download* do *UART Download Tool* seleciona o diretório, lista os arquivos, confirma a porta e a taxa, e inicia. Por cartão SD, valem as mesmas regras de sempre: FAT32 com unidade de alocação de 4 KB, arquivos na pasta `DWIN_SET` na raiz, e o ciclo completo de energia sem troca a quente.

A gravação por serial usa a UART1 — a que o C51 não pode usar. É a mesma porta reservada ao núcleo GUI (§3), e nem todo módulo a traz num conector: em vários ela fica nos *pads* previstos para o módulo Wi-Fi. Havendo esses pads, um conversor USB-serial em **3,3 V TTL** resolve — mas atenção à rotulagem, que às vezes é do ponto de vista do próprio ASIC, e nesse caso a ligação correta é **TX no TX e RX no RX**, além do terra comum. Confira o esquema do módulo antes. **Por isso o cartão SD é o caminho preferido:** funciona em qualquer módulo, em qualquer sistema operacional, e não depende de pad nenhum.

Baud e CRC gravados passam a valer na hora. As opções de comunicação ficam no binário e persistem entre reinícios. Depois de gravar, o outro lado da linha tem de estar na mesma taxa e com a mesma configuração de CRC e de resposta — senão a comunicação cai, e o caminho de volta é regravar uma variante conhecida. Manter à mão um binário em 9600, sem CRC e sem resposta, poupa muito tempo nessa situação.

Depuração com hardware. A interface HME05 liga ao JTAG do núcleo OS e permite simulação dentro do Keil, com ponto de parada. O código roda na Code RAM, sem ser gravado na Flash – conveniente para desenvolver, mas nada do que foi simulado persiste: gravar de verdade continua sendo por serial ou cartão.

12. ARMADILHAS

Tentar usar a UART1. Ela é do núcleo GUI e não está exposta ao C51. Não há registrador a configurar: a porta simplesmente não existe do lado do núcleo OS.

Esquecer o MUX_SEL. Na UART2 e na UART3, sem habilitar a multiplexação o pino continua sendo IO. O código compila, inicializa sem erro e não produz nada na linha.

Esquecer o *push-pull* no TX. Configurar a porta e não configurar o pino de saída em P0MDOUT dá o mesmo sintoma do item anterior: recepção funciona, envio não sai.

Reaproveitar a constante de baud. As três fórmulas da §4 dão valores diferentes para a mesma taxa. Copiar o divisor de uma porta para outra é o erro mais comum ao acrescentar uma segunda UART ao projeto.

Não limpar a marca de interrupção. A limpeza é por software em todas as portas. Sem ela a ISR é reentrada indefinidamente e o núcleo OS para de executar o laço — a tela continua viva, o resto morre.

Uma escrita só no SCON1. A UART3 exige duas escritas consecutivas para limpar a marca. Com uma, o comportamento é errático e difícil de associar à causa.

Trabalhar dentro da ISR. A interrupção move um byte e sai. Montar quadro, calcular CRC ou chamar `Write_Dgus()` ali perde byte na taxa alta.

Gravar o HEX. O HEX do Keil não é aceito. Passe pelo `DownloadFor8051` e grave o `T5L51.bin`. Pela cadeia SDCC o passo não existe — o `make` já entrega o binário.

Atropelar o watchdog ao escrever em MUX_SEL. O mesmo registrador que habilita a UART2 e a UART3 também controla o *watchdog*: o bit `.1` liga e o `.0` alimenta. Por isso a inicialização usa `MUX_SEL |= 0x40`, e nunca `MUX_SEL = 0x40` — a atribuição direta zera os bits do watchdog junto.

Misturar convenções de nome. `SCON3T` é a UART5 no header da DWIN e não existe em frameworks que nomeiam pelo número comercial. Copiar código de outra fonte sem conferir o header produz erro de compilação no melhor caso, e configuração da porta errada no pior. Ver §5.3.

Confiar no tamanho do buffer de envio espontâneo. A rotina de `0x0F00` monta o quadro num vetor local de tamanho fixo, enquanto o byte baixo de `0x0F01` aceita pedir muito mais words do que cabe ali. No driver de referência o limite prático fica em torno de **46 words**: acima disso a rotina escreve fora do vetor e corrompe a pilha, e o sintoma é travamento do núcleo OS sem relação aparente com a serial. Ou limite o pedido na aplicação, ou aumente o vetor junto com o buffer de envio.

Escrever na faixa reservada. `Write_Dgus()` alcança a interface de variáveis de sistema com o mesmo efeito da serial. Escrever numa posição reservada reconfigura o hardware, sem erro nenhum.

13. FONTES

- **DWIN T5L Smart LCMs' UART C51 Program** – DWIN Technology. Base deste documento: registradores das UART2 a UART5, código de inicialização, ISRs, máquina de estados de quadro, tratamento de 0x82 e 0x83, gatilho de envio espontâneo.
- **Development Guide of T5L ASIC** – DWIN Technology. Arquitetura dos dois núcleos, mapa de interrupções, periférias e configuração de IO.
- **T5L_DGUSII Application Development Guide V2.9** – DWIN Technology. Estrutura do quadro, instruções 0x82 e 0x83, interface de variáveis de sistema, gravação por cartão SD.
- **T5L + SDCC Starter Template** – Recep Şenbaş, github.com/recepsenbas/DWIN-T5L-SDCC-Template. Cadeia aberta de compilação: modelo de memória, posição da XRAM, geração do binário sem o DownLoadFor8051. Licença CC BY-NC-SA 4.0 – ver §11.2.