

AGT DISPLAYS

Ambiente de Desenvolvimento C51

Keil, projeto de partida, compilação e gravação do núcleo OS

PARA QUEM

Quem vai escrever código C51 que roda dentro do display

PONTO DE PARTIDA

Projeto de partida da AGT Displays, nunca um projeto vazio

FERRAMENTA

Keil µVision com compilador C51 — não MDK-ARM

RESULTADO

Um binário de núcleo OS gravado no módulo

1. O QUE ESTE DOCUMENTO COBRE

Programar o núcleo OS do display em C51 não é como programar um microcontrolador comum. Não existe pacote de suporte que o Keil reconheça, o dispositivo não está na lista de dispositivos, o arquivo que o compilador produz não é o arquivo que o módulo aceita, e o código só roda se um punhado de registradores tiver sido escrito na ordem certa *antes* da primeira linha de C.

Nada disso é difícil, mas nada disso é óbvio. Este documento descreve a cadeia inteira, do instalador do Keil ao módulo rodando o código: o que instalar, o que o projeto de partida já resolve, o que se configura, como compilar, como gravar e como depurar. A lógica do programa em si – UARTs, interrupções, protocolo, acesso ao espaço de variáveis – é assunto de outro documento.

Antes de instalar qualquer coisa, confirme que você precisa disso. A maioria dos projetos nunca toca no núcleo OS: a lógica mora num MCU externo e o display só recebe comandos pela serial. Esse caminho não usa Keil, não usa gravador e não corre o risco de deixar o módulo mudo. Programar o núcleo OS se justifica quando o objetivo é eliminar o MCU externo, alcançar as periférias do módulo ou atender dois protocolos ao mesmo tempo.

2. AS PEÇAS DO AMBIENTE

São seis coisas, e só a primeira se compra:

PEÇA	O QUE É	OBRIGATÓRIA
Keil µVision + C51	O compilador. A cadeia 8051 da Keil, instalada junto com o µVision.	Sim
Projeto de partida	Projeto µVision já montado: header de SFR, startup, acesso ao espaço de variáveis, driver das UARTs, CRC.	Sim, na prática
Conversor de imagem	O utilitário srec_cat e o arquivo de lote que o chama, transformando o HEX do Keil no binário que o módulo aceita.	Sim
Ferramenta de download	Grava o código C51 no módulo pela serial, sem cartão.	Opcional – ver §9.1
Driver AGDI + driver USB	Faz o depurador do µVision falar com o ASIC pelo adaptador JTAG. Instala dentro da pasta do Keil.	Só para depurar
Placa de avaliação	Um módulo de bancada para errar à vontade, separado do produto.	Recomendada

Compilador certo. O que serve é o **C51**, a cadeia 8051 clássica da Keil. O MDK-ARM, que é o produto mais visível da Keil hoje, não compila nada disso. Na instalação, o C51 costuma vir como pacote separado; se o µVision abrir e não oferecer dispositivos 8051, o pacote não foi instalado.

3. POR QUE NÃO SE COMEÇA DO ZERO

A tentação natural é criar um projeto novo no µVision, escolher um 8051 genérico e começar a escrever. Isso não funciona, e falha de um jeito difícil de diagnosticar: compila limpo, grava sem erro e o módulo não faz nada.

Três coisas faltam num projeto vazio:

- **Os SFRs do ASIC.** O 8051 da DWIN tem registradores que nenhum 8051 de catálogo tem — as quatro UARTs, o controle de página de memória, o acesso ao espaço de variáveis compartilhado, o multiplexador de pinos. Nenhum arquivo de dispositivo do Keil os declara.
- **A assinatura do núcleo.** O módulo só reconhece a imagem como núcleo OS válido se ela carregar uma marca em posição fixa. Sem ela, a imagem é ignorada.
- **A inicialização de baixo nível.** Sete registradores precisam ser escritos em assembly, antes da primeira instrução gerada pelo compilador. Não dá para fazer isso em C.

O projeto de partida resolve os três. O trabalho começa sempre por uma cópia dele, com outro nome, e não por *Project* → *New*.

Projeto de partida disponível no site da AGT Displays. Na área de suporte, com o projeto µVision montado, o header de SFR, o startup correto, o driver de UART de referência e o script de conversão de imagem. É o mesmo projeto que os trechos de código da nossa documentação de núcleo OS presumem.

4. O QUE TEM DENTRO DO PROJETO

O pacote traz dois níveis. O **demo de partida** é deliberadamente mínimo e serve para você confirmar que a cadeia inteira funciona antes de encarar código de verdade. Ao lado dele vêm **exemplos completos**, construídos sobre a mesma base: um com o protocolo serial implementado nas portas disponíveis, outro com leitura de sensor de temperatura. Comece pelo mínimo; use os exemplos como referência, não como base.

O demo de partida são duas metades, e você precisa das duas. Um **firmware** para compilar no Keil e um **projeto de tela** para carregar pelo DGUS. O firmware pisca um LED a cada 500 ms, lê o estado de uma tecla física e, quando ele muda, escreve o novo estado no VP 0x1000; a tela traz um ícone de chave ligado a esse mesmo VP, que alterna entre vermelho e verde. Gravar só o firmware, ou só a tela, faz o demo parecer quebrado quando ele está correto.

A estrutura do firmware de partida é pequena e vale abrir antes de escrever qualquer linha:

ARQUIVO	PAPEL	VOCÊ MEXE?
T5LOS8051.h	Declaração de todos os SFRs do ASIC. É a folha de dados em forma de header.	Nunca
STARTUP_M5.A51	Código de boot em assembly: assinatura, registradores de página, pilha.	Raramente – ver §5
sys.c / sys.h	Inicialização do núcleo, base de tempo de 1 ms, atrasos, macros de watchdog e acesso ao espaço de variáveis.	Só para acrescentar
main.c	Inicializações em sequência e o laço principal.	Sempre

O `main.c` do demo é curto de propósito, e já mostra o essencial: configurar pinos, temporizar pelo atraso do sistema e escrever no espaço de variáveis quando algo muda.

```

sbit LED1 = P1^0;
sbit KEY1 = P1^1;

void io_init()
{
    PORTDRV   = 0x01;    /* corrente de acionamento de 8 mA    */
    P1MDOUT   |= 0x01;    /* P1.0 como saída, aciona o LED    */
    P1MDOUT   &= 0xFD;    /* P1.1 como entrada, lê a tecla    */
}

void main(void)
{
    u16 cnt_1ms, key1_sta;

    sys_init();          /* núcleo, registradores, base de 1 ms */
    io_init();

    cnt_1ms = 0;
    key1_sta = KEY1;
    while (1) {
        cnt_1ms++;
        sys_delay_ms(1);
        if (cnt_1ms == 500) {          /* pisca a cada 500 ms */
            LED1 = !LED1;
            cnt_1ms = 0;
        }
        if (key1_sta != KEY1) {        /* mudou: avisa a tela */
            key1_sta = KEY1;
            sys_write_vp(0x1000, (u8*)&key1_sta, 1);
        }
    }
}

```

Duas coisas merecem nota. As variáveis locais recebem valor **depois** de `sys_init()`, não na declaração — antes disso a inicialização do núcleo ainda não ocorreu e o valor não sobrevive. E a direção de cada pino se define em `PxMDOUT`, bit a bit: `|=` para saída, `&=` para entrada.

Os exemplos completos acrescentam a essa base o que o demo não precisa: `uart.c` com os buffers, interrupções e a máquina de estados de quadro das portas, `crc16.c` para a verificação opcional, e `Parameter_Config.h`, que concentra a configuração — é o assunto da seção seguinte.

4.1. UMA NOTA SOBRE A GRAFIA DOS IDENTIFICADORES

O código da DWIN grafa algumas constantes de forma não convencional — `Uart2_Rx_Lenth` em vez de `Length`, por exemplo. Está assim em todo o ecossistema, inclusive nos ports de terceiros. **Não corrija.** É o nome real do símbolo; trocá-lo faz o código deixar de compilar contra o header.

5. O STARTUP – A PARTE QUE NÃO SE IMPROVISA

O arquivo de startup faz duas coisas que nada mais no projeto faz, e as duas são condição para o módulo rodar o seu código.

5.1. A ASSINATURA EM 0X00F8

No endereço fixo **0x00F8** do espaço de código, a imagem carrega dois bytes de flag e a cadeia DWINT5:

```
CSEG    AT    0F8H
DB      0FFH,0FFH      ; normalmente 00H 00H
DB      'DWINT5'
```

É por essa marca que o módulo identifica a imagem como núcleo OS legítimo. Uma imagem sem ela é recusada em silêncio – a gravação termina sem erro e o programa nunca executa.

5.2. OS SETE REGISTRADORES ANTES DO PRIMEIRO MOVX

Antes que o código C possa tocar em memória externa, o mapa de memória precisa existir. Esses valores são escritos no startup, em assembly:

```
MOV  DAGESEL,  #01H    ; espaço de código de 64 KB
MOV  D_PAGESEL,#02H    ; xdata de 32 KB em 0x8000-0xFFFF
MOV  CKCON,    #00H    ; CPU em modo 1T
MOV  DPC,      #00H    ; modo padrão de ponteiro de dados
MOV  T2CON,    #70H    ; Timer 2 em recarga automática
MOV  MUX_SEL,  #60H    ; watchdog off; UART2 em P0.4/P0.5, UART3 em P0.6/P0.7
MOV  RAMMODE,  #00H    ; libera a RAM de aplicação
```

REGISTRADOR	O QUE DECIDE	SE ESTIVER ERRADO
DAGESEL	Tamanho do espaço de código visível	Código além do limite não executa
D_PAGESEL	Qual página de 32 KB responde como xdata	Variáveis em endereço errado, corrupção silenciosa
CKCON	Ciclo de máquina (1T)	Tudo roda mais devagar; baud errado
DPC	Comportamento do ponteiro de dados	Acesso a vetores falha de forma errática
T2CON	Modo do Timer 2	Base de tempo e watchdog param
MUX_SEL	Pinos das UART2/3 e o watchdog	Serial sai no pino errado ou reset em laço
RAMMODE	Liberação da RAM de aplicação	Nenhum acesso a memória externa funciona

A ordem importa mais que os valores. Esse bloco tem de vir **antes da primeira instrução MOVX do startup** — inclusive antes do laço que zera a xdata que o próprio Keil gera quando XDATALEN é diferente de zero. Deslocar o bloco para depois, ou repeti-lo só em C, produz um módulo que às vezes funciona.

O MUX_SEL merece atenção redobrada: ele carrega o watchdog nos bits **.1** (liga) e **.0** (alimenta), junto com a seleção de pinos. Em C, sempre MUX_SEL |= ..., nunca =, ou uma escrita de seleção de pino desliga o watchdog sem querer — ou o religa.

5.3. PILHA E MEMÓRIA INTERNA

O startup reserva **30 bytes de pilha** em IDATA e zera 0x80 bytes de memória interna na partida. Trinta bytes dão conta do driver de referência com folga pequena: aninhamento profundo de funções ou uma interrupção que chame muita coisa estoura a pilha, e o sintoma é reset aparentemente aleatório. Ao acrescentar camadas ao programa, aumentar DS 30 é barato e vale como primeira medida diagnóstica.

6. PARAMETER_CONFIG.H – A CONFIGURAÇÃO INTEIRA NUM ARQUIVO

Esse arquivo é do **exemplo de comunicação serial**, não do demo de partida – que não tem porta serial nenhuma para configurar. Vale conhecê-lo cedo, porque é para lá que você vai no momento em que o projeto passar a conversar pela serial, e porque ler os `#define` é a forma mais rápida de entender o que o firmware de referência faz.

DEFINE	VALORES	EFEITO
SELECT_RTC_TYPE	0 / 1 / 2	Sem RTC, RX8130 ou SD2058. Tem de bater com a placa.
UART2..5_ENABLE	0 / 1	Compila ou não o driver de cada porta
BAUD_UART2..5	ex. 115200	Baud por porta; o divisor é calculado na compilação
RESPONSE_UART2..5	0 / 1	Devolve ou não a confirmação 4F 4B na escrita
DATA_UPLOAD_UART2..5	0 / 1	Habilita o envio espontâneo disparado pela tela
CRC_CHECK_UART2..5	0 / 1	Exige CRC-16 no quadro daquela porta
DTHD1 / DTHD2	0x5A / 0xA5	Os dois bytes de cabeçalho de quadro

O RTC errado quebra o relógio, não o boot. SELECT_RTC_TYPE tem de corresponder ao chip que está na placa. Com o valor errado o módulo sobe normalmente, a tela funciona e só a hora está errada ou congelada – o que costuma ser descoberto tarde. Confirme o modelo antes de compilar.

As opções por porta são independentes de propósito. É comum deixar CRC ligado na porta que fala com o MCU do produto e desligado na porta de bancada, onde se digitam quadros à mão num terminal.

7. MEMÓRIA – O QUE CABE

O orçamento é apertado para os padrões de hoje e generoso para o que uma IHM precisa fazer:

ESPAÇO	TAMANHO	USO
Código	64 KB	Seu programa inteiro, driver incluso
xdata	32 KB em 0x8000–0xFFFF	Variáveis grandes, buffers de recepção
idata	256 bytes	Pilha e variáveis de acesso rápido
data / bit	128 bytes / 16 bytes	Contadores quentes e flags

Buffer de UART é o que consome xdata: quatro portas com recepção e transmissão somam rápido. Declare buffers grandes explicitamente em xdata e deixe em data só o que é lido a cada volta do laço – num 8051 a diferença de velocidade entre os dois é grande.

O projeto tem de dizer isso ao Keil. O projeto de partida já vem configurado; num projeto novo, na aba Target das opções do alvo, escolha **Memory Model: Large (variables in XDATA), Code ROM Size: Large (64K program)** e declare a RAM externa em **Off-chip Xdata memory** com início 0x8000 e tamanho 0x8000. Sem isso o ligador distribui as variáveis num espaço que o chip não tem.

Não há proteção de estouro. O 8051 não tem MMU nem checagem de faixa. Um vetor escrito além do fim sobrescreve a variável seguinte e o programa continua rodando com dado errado. Arrays intermediários do driver de referência têm limites justos – o de leitura de bloco, por exemplo, não aceita mais que algumas dezenas de words. Confira o tamanho declarado antes de aumentar um pedido de leitura.

8. COMPILAR E CONVERTER

O Keil produz um arquivo HEX. O módulo não aceita HEX — ele quer um binário puro, com a imagem posicionada desde o endereço zero e a assinatura no lugar. A conversão é um passo próprio.

A ferramenta de download serial (seção 9.1) aceita o HEX diretamente — nesse caminho a conversão é dispensável. Ela continua obrigatória para gravar por cartão, que só lê o binário.

Opções de saída que o projeto de partida já traz e que valem para um projeto novo: ligador estendido LX51 marcado na aba Device; diretiva REMOVEUNUSED em LX51 Misc, que descarta funções não chamadas em vez de só avisar; saída na pasta OBJ em formato HEX-80, onde a conversão procura o arquivo; e otimização de nível 3 com ênfase em tamanho. Níveis mais altos rendem código mais rápido, mas o material do fabricante recomenda não subir em projetos grandes.

No projeto de partida esse passo já está escrito, num arquivo de lote que chama o utilitário `srec_cat`. A linha que importa é uma só:

```
srec_cat.exe template.hex -Intel -o T5L51.bin -Binary
copy "T5L51.bin" E:\DWIN_SET
copy "T5L51.bin" ..\
```

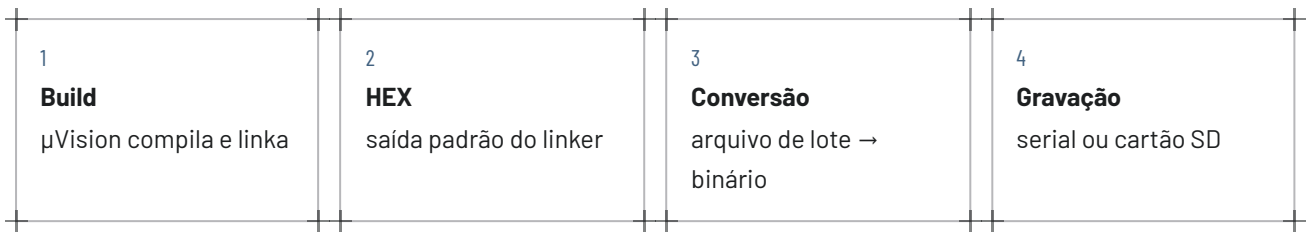
Três coisas para saber antes de rodar:

- **Roda de dentro da pasta OBJ** que o Keil cria no build — é lá que o HEX está. Rodar de fora não acha o arquivo de entrada e o lote segue adiante sem reclamar.
- **O nome do HEX de entrada é o nome do alvo do projeto.** Se você renomear o alvo no µVision, o lote passa a apontar para um arquivo que não existe. É o primeiro lugar a conferir quando "a mudança não fez efeito".
- **A letra do drive é fixa no arquivo.** Edite `E:\DWIN_SET` para a do seu cartão, ou apague a linha e copie à mão.

O nome de saída também não é livre. Dentro do `DWIN_SET`, cada tipo de arquivo é reconhecido pelo **prefixo** do nome, e o do programa 8051 do núcleo OS é `T5L51` — `T5L51*.BIN`, onde o `*` é um sufixo descritivo opcional. O lote gera `T5L51.bin`, que atende a regra. Um nome com outro prefixo é simplesmente ignorado na gravação.

Nome e assinatura são duas coisas. O **prefixo do nome** diz ao gravador *onde* o arquivo vai — é o que decide que aquele binário é programa do núcleo OS e não configuração de tela. A **assinatura em 0x00F8** (§5.1) é o que faz a imagem já gravada ser aceita como válida na partida. Errar o nome é gravação que não acontece; errar a assinatura é gravação que acontece e não roda.

O caminho completo, do fonte ao módulo:



Duas conferências valem o tempo, sempre:

- **O mapa do linker.** O µVision informa o consumo de código e de cada classe de RAM ao fim do build. Acompanhar esse número a cada versão evita a descoberta tardia de que não cabe mais nada.
- **A data do binário.** Se a conversão falhar, o arquivo antigo continua na pasta e é ele que vai para o módulo. Gravar a versão anterior e concluir que "a mudança não fez efeito" é o erro de diagnóstico mais comum de toda essa cadeia.

9. GRAVAR NO MÓDULO

Há dois caminhos, e eles servem a momentos diferentes.

9.1. PELA SERIAL, DURANTE O DESENVOLVIMENTO

A ferramenta de download acompanha o ambiente: ela abre a porta serial do PC, conversa com o módulo e escreve a imagem do núcleo OS. É o caminho de trabalho, porque o ciclo editar-compile-gravar-testar fica em segundos e não exige mexer no cartão.

Precisa de um conversor USB-serial ligado à porta de programação do módulo, e a taxa usada pela ferramenta tem de ser a que o módulo está usando – se o firmware anterior mudou o baud da porta, é preciso acompanhar.

A gravação por serial depende do firmware que já está no módulo. Quem atende o download é o próprio código C51 em execução – não é recurso do silício. O firmware de referência só passou a aceitar isso a partir de uma revisão de 2023, e **com a confirmação 4F 4B habilitada** naquela porta (RESPONSE_UARTx em Parameter_Config.h). Consequências práticas: a **primeira** gravação de um módulo de fábrica é sempre por cartão, e uma imagem que desabilite essa confirmação fecha a porta atrás de si – a partir dela, só cartão.

9.2. POR CARTÃO SD, NA PRODUÇÃO

O binário do núcleo OS entra na pasta DWIN_SET junto com o resto do projeto DGUS e é gravado no mesmo procedimento de sempre. As regras do cartão valem inteiras – **SD ou SDHC, FAT32, cluster de 4 KB**, formatado pelo prompt de comando:

```
format /q g: /fs:fat32 /a:4096
```

E a sequência sem troca a quente: desligar, inserir o cartão, ligar, esperar a tela azul terminar, desligar, remover, ligar. Esse é o caminho para gravar um lote e para entregar atualização a campo.

Queda de energia no meio da gravação danifica o módulo. Vale para os dois caminhos. Em bancada, use fonte firme e não desligue no impulso. Incluir o arquivo de verificação CRC no DWIN_SET faz o módulo conferir o que gravou, em vez de deixar passar uma gravação parcial.

9.3. QUANDO O MÓDULO PARA DE RESPONDER

Um código C51 que trava o laço principal com o watchdog desligado deixa o módulo aparentemente morto – a tela pode até continuar desenhando, porque o núcleo GUI é outro, mas nada responde. A saída é regravar por cartão, que não depende do programa em execução. Se nem o cartão resolver, o próprio núcleo foi afetado e o caminho é a interface de regravação de kernel do módulo.

A prevenção é barata: **deixe o watchdog ligado** desde a primeira versão, alimentado pela base de 1 ms do Timer 2. Um travamento vira um reset, e um reset é depurável.

10. DEPURAR NO HARDWARE

O µVision pode depurar o código rodando dentro do display – ponto de parada, passo a passo, inspeção de variável e de SFR. Para isso ele precisa de um driver de depuração específico do ASIC, instalado dentro da pasta do Keil, mais o driver USB do adaptador.

A instalação mexe em dois lugares do µVision, e os dois precisam estar certos:

ONDE	O QUE SELECIONAR
<i>Options for Target</i> → <i>Device</i>	Em Database , escolher DWIN Device ; dentro dela, vendor DWIN , dispositivo T5 , toolset C51 . A base só aparece depois de instalar o driver.
<i>Options for Target</i> → <i>Debug</i>	Marcar a coluna da direita (depurador de hardware) e escolher o emulador 8051 do adaptador na lista <i>Use:</i> – não <i>Use Simulator</i> , à esquerda.

Se a base **DWIN Device** não estiver na lista, ou se o emulador não aparecer em *Use:*, a instalação foi na pasta errada do Keil. É a falha típica, e ela não dá mensagem – o µVision simplesmente segue oferecendo só o simulador.

Vale conferir também o *include path*: os projetos de referência esperam a pasta de headers da DWIN dentro da árvore de Inc do C51. Se o build reclamar de header não encontrado, é esse caminho.

Vale dizer o que a depuração **não** alcança: o núcleo GUI. Parar o núcleo OS num ponto de parada não para a tela; os controles continuam sendo varridos, o toque continua respondendo e o que estava no espaço de variáveis continua sendo exibido. Isso ajuda – dá para inspecionar uma variável e ver a tela ao mesmo tempo – mas confunde quem espera que o sistema todo congele.

Depuração sem depurador. Na prática, boa parte do trabalho se resolve escrevendo no espaço de variáveis: um Data Variables num canto da tela, alimentado por `Write_Dgus()`, é um *printf* que não custa nada e não muda a temporização. Contadores de quadro recebido, de erro de CRC e de volta de laço, cada um no seu VP, resolvem mais defeitos de comunicação que um ponto de parada.

11. PLACA DE BANCADA

Vale ter um módulo separado do produto para desenvolver o núcleo OS — e se for uma placa de avaliação, melhor, porque os sinais estão acessíveis.

A razão é concreta: o ciclo de aprendizado do núcleo OS inclui necessariamente algumas imagens que travam o módulo, e recuperar cada uma custa tempo. Numa placa de bancada isso é rotina; num produto montado, com o cartão atrás da tampa e a serial já ocupada pelo MCU, cada recuperação é uma desmontagem.

O segundo motivo é o RTC. Placas diferentes trazem chips de relógio diferentes, e `SELECT_RTC_TYPE` tem de acompanhar. Desenvolver numa placa e produzir em outra sem conferir esse define é um erro que só aparece no campo.

12. ARMADILHAS DO AMBIENTE

Todas já custaram tempo a alguém. Nenhuma dá mensagem de erro.

SINTOMA	CAUSA PROVÁVEL
Compila e grava, módulo não faz nada	Assinatura ausente – projeto criado do zero, sem o startup certo
Funciona às vezes; variáveis com lixo	Bloco de registradores de página depois do primeiro MOVX
Reset em laço logo após ligar	Watchdog ligado e não alimentado; ou MUX_SEL escrito com =
Serial muda não sai em nenhum pino	Seleção de pino do MUX_SEL, ou porta desabilitada em Parameter_Config.h
Baud errado em todas as portas	CKCON fora do modo 1T – o clock inteiro mudou
Mudança no código sem efeito nenhum	Conversão falhou e o binário antigo foi gravado: lote rodado fora da pasta OBJ, ou alvo do projeto renomeado sem ajustar o lote
Imagem no cartão ignorada	Binário renomeado fora do prefixo T5L51
Gravação por serial não responde	Firmware em execução não implementa o download serial, ou a confirmação 4F 4B está desabilitada naquela porta
µVision só oferece o simulador	Base de dispositivos da DWIN ausente – driver de depuração instalado fora da pasta do Keil
Relógio errado, resto normal	SELECT_RTC_TYPE não corresponde ao chip da placa
Reset aparentemente aleatório	Pilha de 30 bytes estourada por aninhamento ou interrupção
Um VP muda sozinho quando outro é escrito	A leitura de contexto dentro da escrita de word foi removida
Driver de depuração não aparece no µVision	Instalado fora da pasta do Keil

13. PRIMEIRA SUBIDA – ROTEIRO

Faça esta sequência antes de escrever uma linha de lógica. Ela valida a cadeia inteira com o mínimo de variáveis em jogo.

1. Instale o µVision **com o pacote C51** e confirme que o compilador 8051 está presente.
2. Descompacte o projeto de partida numa pasta **sem espaço nem acento no caminho** – a cadeia da Keil ainda tropeça nisso.
3. Instale os drivers do pacote antes de tentar gravar ou depurar: o USB do adaptador de depuração e o do conversor USB-serial.
4. Abra o projeto de partida, compile sem mudar nada e confirme que o build termina limpo. Anote o consumo de código e de RAM como linha de base.
5. Converta o HEX e grave o firmware. **Confira a data do binário** antes de gravar.
6. Carregue pelo DGUS o **projeto de tela que vem com o demo** – é a outra metade, e sem ela não há o que observar.
7. Ligue e pressione a tecla. O LED piscando mostra que o firmware roda; o ícone na tela alternando de cor a cada toque mostra que o canal entre os dois núcleos está aberto.

Esses dois sinais, juntos, validam a cadeia inteira: compilador, startup, mapa de memória, conversão, gravação e acesso ao espaço de variáveis. Se algum deles faltar, o problema está em um desses passos e não na sua lógica – que ainda não existe. Depurar essa sequência mínima é muito mais rápido do que descobrir o mesmo defeito no meio de um firmware pronto.

Só depois disso vale trocar o conteúdo do laço pelo seu, um pedaço por vez, sempre com o demo funcionando como referência de que a base está boa.

A partir daqui o assunto é o programa em si: portas seriais, interrupções, protocolo e acesso ao espaço de variáveis.