C++ Digital Control Framework

Digital Control Template Guide & Carrier Interrupt Timing

An architectural guide explaining simulator engine execution, precise carrier waveform timing for interrupts 0 to 3, multi-PWM array access, and step-by-step customization for power converter topologies.

Arief Noor Rahman
Power Control Design

Template File Architecture

buck.cpp

Engine & Scheduler: Allocates `sDGTL_CTRL_BLK`, manages `TSAMPLING` scheduler, dispatches ISRs, and calculates solver timestep limits (`MaxExtStepSize`).

pwm_edge_handler.c/.h

PWM Timing Engine: Generates symmetric center-aligned carriers, tracks switching edges (`t_on`, `t_off`), and enforces deadtime (`dt`).

interrupt_handler.c/.h

Control Algorithms: Holds ISR functions (`interrupt0..3`). Receives `struct sPWM pwm[]` pointer to update duty compare registers directly.

io_def.inc & uData.h

Pin Binding Layer: Maps simulator analog/digital channels (`uData`) to clean reference aliases (`vout`, `hi`, `lo`, `iref`).

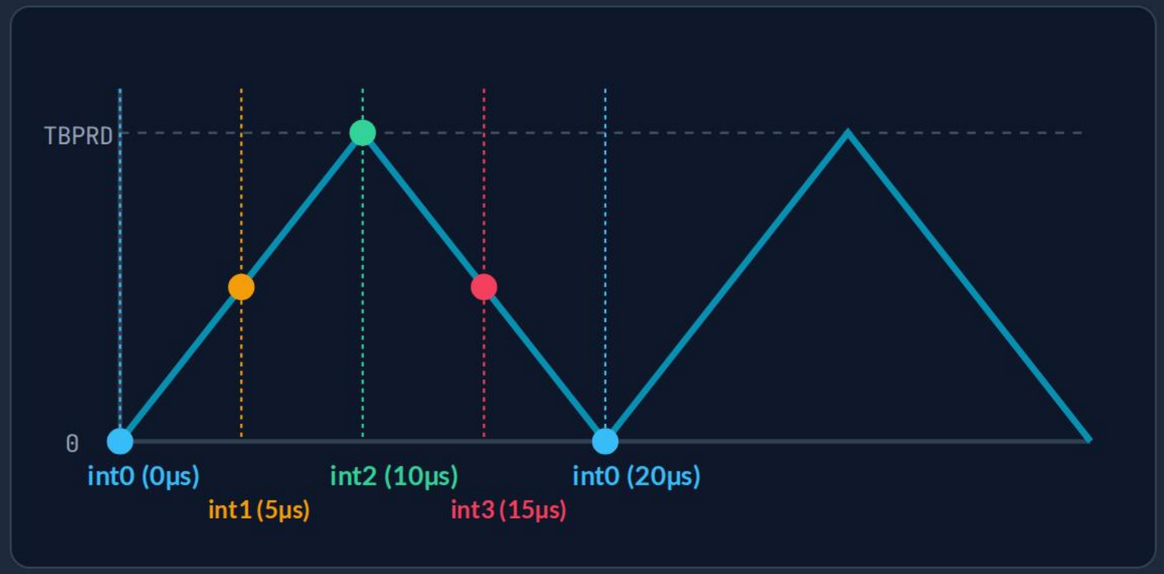
Carrier Waveform & Sampling Period Mechanics

PWM Carrier Period Parameters

The triangular carrier period T_{pwm} is defined by counter limit `TBPRD` and clock period `DGTL_CLK` (10ns):

```
#define TBPRD 1000 // 1000 clock counts #define DGTL_CLK 10E-9 // 10ns clock resolution // Half period (valley to peak): t_prd = 1000 * 10ns = 10us // Full period
```

Interrupt Timing Location on Carrier Waveform



ISR Function	Carrier Phase	Timestamp (t)	Carrier State
`interrupt0`	0° / 360°	$0\,\mu\text{s}, 20\,\mu\text{s}\dots$	**Carrier Valley** (Counter = 0)
`interrupt1`	90°	$5\,\mu\text{s}, 25\,\mu\text{s}\dots$	**Rising Slope** (50% Up)
`interrupt2`	180°	$10\,\mu\text{s}, 30\,\mu\text{s}\dots$	**Carrier Peak** (Counter = TBPRD)
`interrupt3`	270°	$15\,\mu\text{s}, 35\,\mu\text{s}\dots$	**Falling Slope** (50% Down)

Control Application for Each Interrupt Event

interrupt0 (Valley)

Valley ADC Sampling: In Buck converters, inductor current ripple reaches minimum at carrier valley. Sampling here provides noise-free average current sensing without switching spikes.

interrupt1 (Upslope)

Mid-carrier up/downslope useful for implementing 4x oversampling feedback measurement.

interrupt2 (Peak)

Peak Sampling / Double Update: Ideal for Boost converters (current peak sensing) or implementing double-update PWM (updating duty cycle twice per carrier period at valley and peak).

interrupt3 (Downslope)

Mid-carrier up/downslope useful for implementing 4x oversampling feedback measurement.

PWM Timing Engine & Hardware Emulation Customization

Edge Transition Logic (`pwm_edge_handler.c`)

At carrier valley (`t_zero`), turn-on (`t_on`) and turn-off (`t_off`) timestamps are calculated from compare register `cmpa` :

```
// Edge Timestamp Calculation at Valley a->t_on = *CKTtime + DGTL_CLK * (2 * a->prd - a->cmpa); a->t_off = *CKTtime + DGTL_CLK * a->cmpa; // Gate Drives with Deadtime
```

Multi-PWM Access via Array Pointer Signature

Passing `inst→pwm` Array Pointer

The dispatcher in `buck.cpp` passes the base array address, giving ISR handlers access to any PWM channel (`pwm[0]`, `pwm[1]`...):

```
// Dispatcher Call Site in buck.cpp switch(inst→counter) { case 0: interrupt0(&inst→data, inst→pwm, data); break; case 1: interrupt1(&inst→data, inst→pwm, dat
```

Signal Pin Mapping via io_def.inc

Pin Index (`data[N]`)	C++ Reference Alias	Signal Direction	Data Type	Typical Circuit Function
`data[0]`	`vout`	Simulator Input	`double`	Output Voltage Sensing (Feedback ADC)
`data[1]`	`iout`	Simulator Input	`double`	Inductor Current Sensing (Feedback ADC)
`data[2]`	`hi`	DLL Output	`bool &`	High-Side MOSFET Gate Drive Signal (`outa`)
`data[3]`	`lo`	DLL Output	`bool &`	Low-Side Synchronous MOSFET Gate Signal (`outb`)
`data[5]`	`iref`	DLL Output	`double &`	Duty / Reference Telemetry Scope Variable
`data[6..8]`	`dbg0, dbg1, dbg2`	DLL Output	`double &`	Debugging telemetry registers for simulation plot

Adaptation Guide: Workflow Steps

Step 1: Set Constants

Adjust `PWM_CH`, carrier period `TBPRD`, deadtime `DTIME`, and sampling interval `TSAMPLING`.

Step 2: Map IO Pins

Expand `io_def.inc` with new sensor inputs and gate driver output aliases.

Step 3: Output Assignment

Bind PWM gate outputs (`outa`, `outb`) to mapped pin aliases inside `buck.cpp`.

Step 4: Control Logic

Implement PID, 2P2Z, 3P3Z, or state feedback equations inside `interrupt0..3`.

Step 1 & 2: Multi-PWM & Initialization

Step 1: Set Top Constants (`buck.cpp`)

Define multi-channel count and period settings at the top of `buck.cpp`:

```
#define TSAMPLING 2.5E-6 // 400kHz sampling (4x oversampling) #define PWM_CH 2 // Dual Channel PWM #define TBPRD 1000 // Carrier Period = 2000 counts #define DTIME
```

Multi-Topology Configuration Matrix

Topology	`PWM_CH`	Preferred ISR Sampling Point & Modulation Strategy
Synchronous Buck	1	`interrupt0` (Valley: low current ripple noise sampling)
Synchronous Boost	1	`interrupt2` (Peak: clean current sample)
Dual-Phase Interleaved	2	`interrupt0` (Ph1 Valley) & `interrupt2` (Ph2 Valley / 180° Shift)
Phase-Shifted Full Bridge	2	`interrupt0` & `interrupt2` for double-edge duty calculation
3-Phase Inverter (SVPWM)	3	`interrupt0` (Valley) or `interrupt2` (Peak) for symmetrical vector updates

Build Guidelines & Naming Consistency Rules

⚠ CRITICAL: Strict Naming Consistency Requirement

To ensure QSPICE circuit simulator correctly links and loads your compiled C-block DLL, the following **4 elements MUST match name strings identically**:

- ✓ **1. QSPICE Schematic C-Block Symbol Name:** e.g., `buck`
- ✓ **2. C++ Source Filename:** e.g., `buck.cpp`
- ✓ **3. Exported DLL Function Signature in buck.cpp:** `extern "C" __declspec(dllexport) void buck(...)`
- ✓ **4. VS Code tasks.json Build Rule:** Compiler args referencing source `"buck.cpp"` and output `"buck.dll"`

Compiler Execution Command

Build DLL using Digital Mars C++ or MSVC command line:

```
// Build rule inside VS Code tasks.json: dmc -mn -WD -o buck.cpp kernel32.lib
```

Verification Checklist

If renaming your controller to e.g. `boost` or `full_bridge`, update all 4 locations simultaneously before rebuilding!