STM32 BLDC Motor Library Manual

This library has been developed and tested on an STM32G431C6 (B-G431B-ESC1 board) driving a 42BLDC-24 motor (24 V, 4000 RPM, with three Hall sensors). Due to a broken PCB connection, one Hall sensor signal was unavailable. Rather than replacing the ESC, the library was extended to tolerate a single Hall sensor fault, allowing uninterrupted operation.
With this fault-tolerant design, you can drive your BLDC motor using either:

- Six-step hard switching
- Six-step soft switching

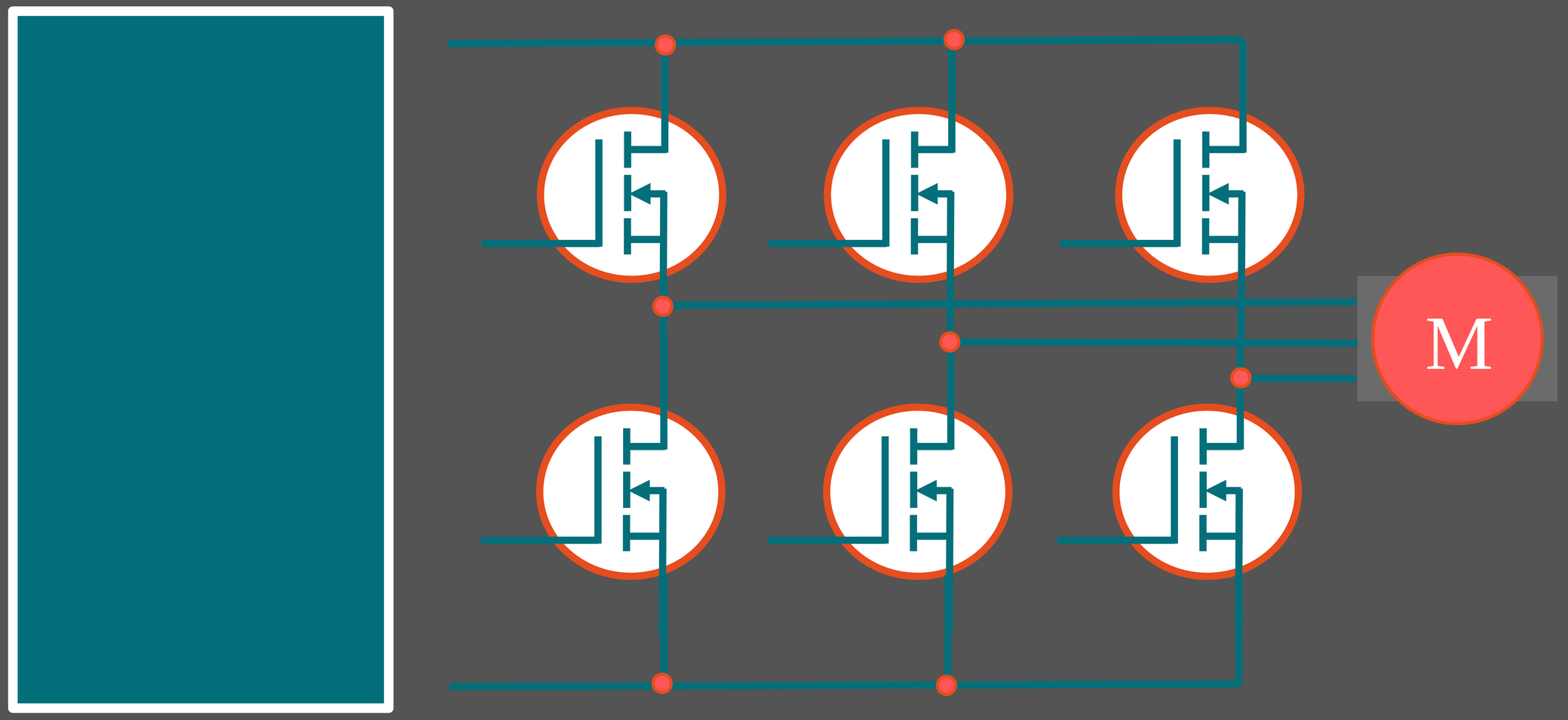
Both modes employ the same fundamental six-step sequence but differ in PWM gating during on/off intervals, trading complexity for efficiency.

Library Features

- Hard switching six-step commutation
- Soft switching six-step commutation with
- Fault tolerance for up to one missing Hall sensor signal
- Control with PID
- Timeout and noise rejection to prevent stalled or jittery operation

Bridging Concept: Half-Bridge Simplification

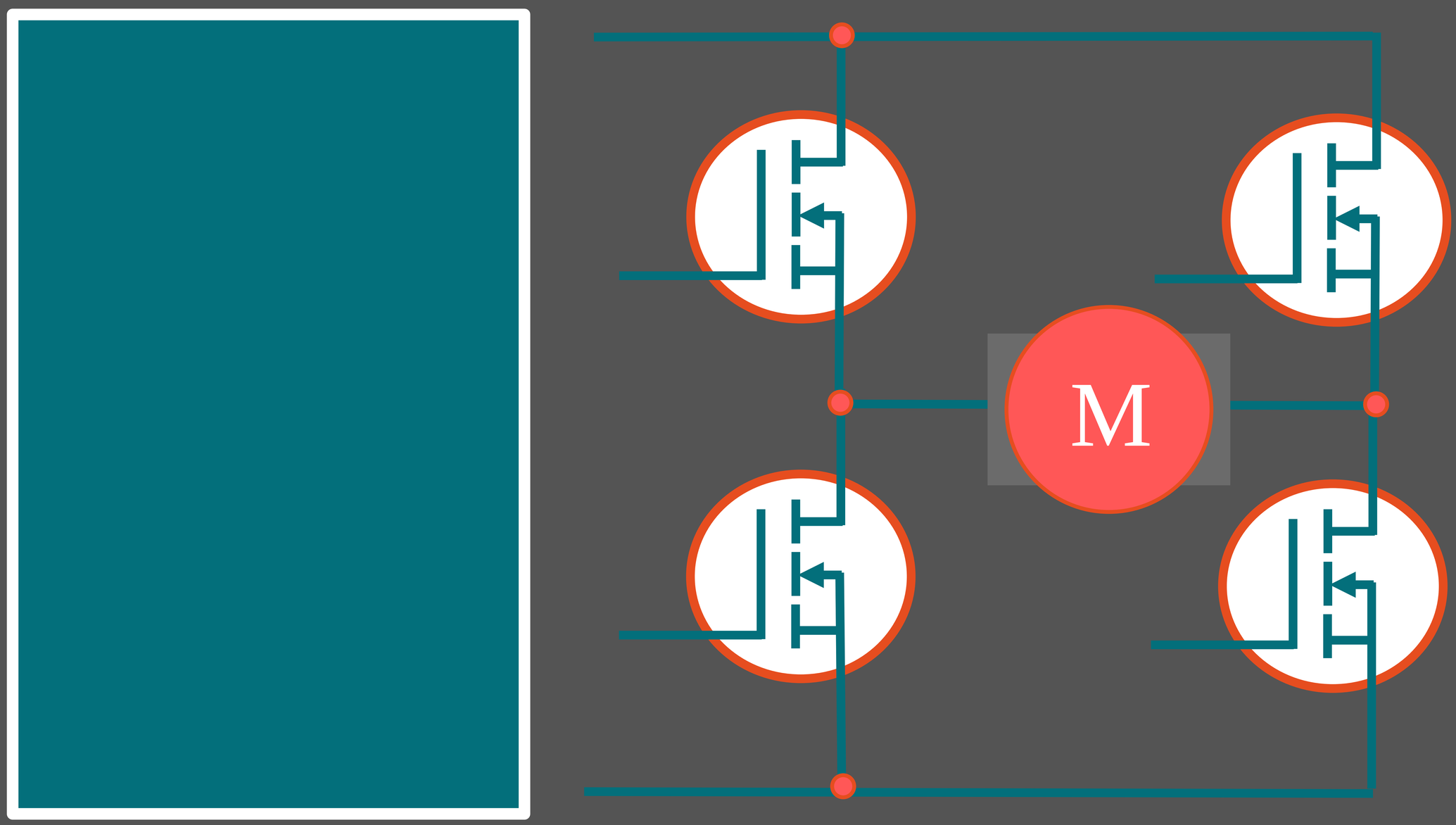
To clarify current paths, we collapse each full three-phase bridge into its active half-bridge pair.



At any given step, only two transistors conduct:

- High-side transistor of Phase X
- Low-side transistor of Phase Y

All other devices remain off, so a two-device model suffices for understanding switching losses and conduction paths.



Hard Switching

- Drive: PWM on High-side of Phase X and PWM on Low-side of Phase Y (same duty)
- On-time path: High-side X → motor winding → Low-side Y
- Off-time path: Freewheel via Low-side Y's flyback diode and High-side X's flyback diode

Losses:

- Two diode drops during each off interval
- Switching losses on both transistors

Soft Switching

- Drive: PWM on High-side of Phase X; constant on (logic 1) on Low-side of Phase Y
- On-time path: High-side X → motor winding → Low-side Y
- Off-time path: Freewheel through Low-side Y conduction and one diode drop in High-side X or low-side diodes

Losses:

- Single diode drop during off interval → higher efficiency
- Switching losses on high side transistor

Note!!!

All channels assume active-high gating.

Timer Configurations

- TIM1 (or TIM8): generates complementary PWM on three channels for six-step gating.
- TIM2 (or any): drives the commutation step rate and measures elapsed microseconds.

The appropriate timer (TIM2) you could set in timer.c.

PID Controller Setup

PID setpoint tied to TIM2 (or any) reload value for step interval. Change the values as needed for appropriate values.

The idea of hall sensor fault tolerance lie in the fact that it does not matter which presciese sensor is broken, the only thing that you need is to detect the {1, 1, 1} and regardless of the which sensor is broken the third and the sixth step (if we will consider {1, 1, 1} step as first) will not be seen by the MCU, that is why we emulate the end of the step by simply breaking the step after the same time will elapse which previous step is been taken to elapse.

Of course to avoid cases uder rapid acceleration, when the each next step is much much shorter then the previous one, we add the break by the hall sensor along with the break by time during the steps which I called "duplicated" steps.

Usage Example

```c
#include "SIXSTEPHeader/MAL_declaration.h"

int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM1_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */

  HAL_TIM_Base_Start(&htim2);


  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
   MAL_Run(HardSwitching, &htim1, function); // Executes six step

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```