MCP2515 Standalone CAN controller with SPI Interface Driver Manual

Content :

Important notes for this driver

This drivers assume certains conditions which are necessary to be met for its proper functioning.

- This driver is written using the microprocessor’s HAL library which includes using several inter-processor components like SPI, GPIO etc, thus one need to include the header files such as HAL_SPI.h, HAL_GPIO.h etc in the ‘/Inc/CAN_mcp2515.h’.
- Since the driver uses HAL library, for SPI communication, a SPI structure is create by HAL library, in this driver, the SPI structure called as ‘hspi’, is used by the name of ‘hspi1’, thus its must to initialize the SPI structure by the name ‘hspi1’.
- All the HAL library sub-sections like HAL_GPIO, HAL_SPI, HAL_NVIC etc must be initialized by the the programmer prior to using the driver using the appropriate HAL routines.
- This library is written for little endian architecture based MCUs.

This driver can be grouped into two parts :
1.) CAN IO API
2.) CAN Config API

where CAN Config API is further divided into CAN Config mode config API and CAN Normal mode config API.

API Summary

CAN IO API :

CAN_WriteRegister()	Writes to a specific register
CAN_ReadRegister()	Reads from a specific register
CAN_BitModify()	Writes to specific bit of a register without fluctuating others
CAN_WriteFrame()	Writes the entire CAN frame
CAN_ReadFrame()	Reads the entire CAN frame
CAN_Transmit()	Transmits the loaded frame
CAN_MarkRead()	Mark the received frame as read
CAN_tx()	Transmits a CAN frame loaded in a TXB
CAN_rx()	Copies the received CAN frame from RXB to specific memory

CAN Config mode config API :

CAN_GetFilter()	Gets specific filter
CAN_SetFilter()	Sets specific filter
CAN_GetMask()	Gets specific mask

CAN_SetMask()	Sets specific mask
CAN_ConfigureSJW()	Configures SJW
CAN_ConfigureBRP()	Configures BRPs
CAN_ConfigurePropSeg()	Configures Prop. segm.
CAN_ConfigurePS1()	Configures PS1 segment
CAN_ConfigurePS2()	Configures PS2 segment
CAN_GetBaudRate()	Gets the current baudrate
CAN_SetBaudRate()	Sets a new baudrate
CAN_ConfigTXnRTS()	Configure TxnRTS pins
CAN_ConfigureTripleSapling()	Enables/disables triple sampling of CAN bits
CAN_ConfigureBTLMODE()	Configures BTL mode
CAN_ConfigureWakeUp()	Configures wake up mode
CAN_ConfigureSOF_CLKOUT()	Configures SOF mode for CLKOUT pin
CAN_HardReset	Performs hard reset
CAN_SoftReset()	Performs soft reset
CAN_Init()	Initializes the device with this API

CAN Normal mode config API

CAN_SetInterrupt()	Sets specific interrupt
CAN_GetInterrupt()	Gets specific interrupt

CAN_EnableInterrupt()	Enables specific interrupt
CAN_DisableInterrupt()	Disables specific interrupt
CAN_TriggerRTS_SPI()	Trigger RTS via SPI
CAN_TriggerRTS_PIN()	Trigger RTS via TxnRTS pins
CAN_AbortTX()	Aborts specific TXBn transmission
CAN_AbortAllTX()	Aborts all pending transmissions
CAN_UnAbortAllTX()	Marks the TXB as empty
CAN_GetClkOut()	Gets current CLKOUT configuration
CAN_SetClkOut()	Sets specific CLKOUT configuration
CAN_SetClkOutFreq()	Sets specific frequency for CLKOUT
CAN_EnableOSM()	Enables One-Shot Mode (OSM)
CAN_DisableOSM()	Disables OSM
CAN_ChangeTXPriority()	Change transmit priority of specific TXB
CAN_SetRXBMode()	Sets filter and no-filter mode for specific RXB
CAN_ConfigRXnBF()	Configures RXnBF pins
CAN_GetRTR()	Gets RTR bit of specific TXB/RXB

CAN_SetRTR()	Sets RTR bit for specific TXB
CAN_GetFrameType()	Gets the frame type of specific TXB/RXB
CAN_SetFrameType()	Sets the frame type for specific TXB
CAN_RollOver2RX1()	Configure RX0 rollover to RX1
CAN_GetEFLG()	Gets the EFLG value
CAN_SetEFLG()	Sets the EFLG value
CAN_Get_TEC_REC()	Gets the value of TEC/REC register
CAN_SwitchMode()	Switches the device's operation mode

MCP2515 bootup with this driver

MCP2515 by default bootup into configuration mode, this driver initializes the device with a specific configuration by setting appropriate values into the configuration and buffer registers.
MCP2515 by default may be booted up with some different default configuration, but when this driver is used to initialize the device, it does it into a specifc manner.

CAN_Init() is used to initialize the device using this driver.

Following are the values for the registers set by this driver :

1. Configuration registers :
   - CNF1 : 0b00000000
   - CNF2 : 0b10011000
   - CNF3 : 0b01000001

Since the baudrate is calculated as :
baudrate =
F_OSC/(2*(1+BV_brp)*(4+BV_propseg+BV_ps1
+BV_ps2))

and assuming the oscillator frequency be 8MHz,
baudrate=500kbps
and sample point location is 75% bit-time.

Now the filters and mask registers, all are set to 0x00 to allow reception of any successful.

- RXFn : 0b00000000
- RXMn : 0b00000000

Enabling all TXRTS pins to allow frame transmission trigger via GPIO.

- TXRTSCTRL : 0b00000111

Now, mode is changed to normal mode but before that, interrupts are disabled, although interrupts are not available in configuration mode but still for the smooth configuration and initialization of the device, interrupts are turned off.

- CANINTE : 0b00000000
- CANINTF : 0b00000000

Once the configuration registers are set and interrupts are disabled and flags are cleared, the mode of the device is then changed by writing to the CANCTRL register to Normal mode.

- CANCTRL : 0b000xxxxx

last 5 bits are not set during this operation, only mde bits are modified.

2. Other register :

- BFPCTRL : 0b00001111 (enabling both RXnBF pins.)
- TEC : 0b00000000
- REC : 0b00000000

- CANCTRL : 0bxxx00100 (CLKOUT freq = OSC freq)
- EFLG : 0b00000000 (lower 6 bits are read only, setting 0 won’t do anything, they will be low)

Now setting all the TXBn/RXBn registers (except TXBnCTRL and RXBnCTRL register) to 0x00.

- TXBn : 0b00000000
- RXBn : 0b00000000

TXBn/RXBn includes mXnSIDL, mXnSIDH, mXnEID0, mXnEID0, mXnDLC, mXnDp [m{T, R}, n{0-2 for m=T, 0-1 for m=R}, p{0-7}]

- TXBnCTRL : 0b00000000 (Transmit buffer priority is set to lowes for all TX buffers.)
- RXBnCTRL : 0b01100000 (RXBn is allowed to accepts any of the incoming frame.)

Modes of operation, Configuration mode, CAN Bit timing and CAN Config API

MCP2515 has 5 mode of operation as listed below :

- Configuration mode
- Normal mode
- Sleep mode
- Listen-only mode
- Loopback mode

The device can be intentionally put into any of the mode by writing to CANCTRL register.
In Configuration mode, all registers are accessible and device can be configured to operate under several constraints and dependencies (RX and TX is not allowed in this mode).

In Normal mode, configuration registers are not accessible which includes CNFn(s), TXRTSCTRL, masks and filters. This is the default operation mode which allows TX and RX.

In Sleep mode, the device enters a power saving state from which it can be awaken by some external activity on CAN bus or by SPI instruction, MCU must reset the mode bits in CANCTRL register for transitioning back to the normal mode.

In Listen-only mode, the device don’t actively participates into the communication insead it just passively listens the

CAN traffic, thus generally used or debugging purpose.

In Loopback mode, there is no real transmission or reception, the device enters a loopback state where it can send frames to itself and read them as well, thus wll suited for device configuratio and testing purpose.

Since the device boots up into the configuration mode by default and thus needs to be switched to the normal mode by the programmer, it thus gives fine grain control to the configuration registers of the device to the programmer, The configuration registers includes various registers as discussed in previous module, among them, CNFn(s) are used for controlling the baud rate and synchronization behaviour of the device, you can look at the device data sheet to configure the bit timing and synchronization or just use this driver to do the same.

The baud rate is calculated as follow :
baudrate(F_OSC, BV_brp, BV_propseg, BV_ps1, BV_ps2) =
F_OSC/(2*(1+BV_brp)*(4+BV_propseg+BV_ps1+BV_ps2))
[Mathematical relation]

thus using the above relation, device can be configured to operate at required baudrate and with required synchronization by modifying the SJW field using CAN_ConfigureSJW() routine.

The CAN config API is divided into two parts :
1.) CAN Config mode config API
2.) CAN Normal mode config API

Config mode config API includes those routines which performs the configuration in configuration mode and Normal mode config API does the same in Normal operation mode of MCP2515.

In this module, we’ll be covering the CAN config mode config API.