# **ISO1228 SPI Flowchart & Algorithm**

# **Control Pins set by Jumpers on board: *Burst\_EN = GND (Jumper set on Board)SYNC = VCC1 (Jumper set on Board)***

***F1 =******VCC1 (Jumper set on Board)***

***F0 = Float***

***COMM\_SEL = controlling by MCU***

***OUT\_EN = controlling by MCU***

***nRST = controlling by MCU***

**File:** main.c (ISO1228 serial-only integrated)

**What this doc contains**

* High-level flowchart (mermaid) of the program
* Detailed step-by-step algorithm (initialization, SPI read/write, pin transactions)
* Pin/timing transaction sequences for serial start/stop and register R/W
* Pseudocode / state-machine for the main loop and iso\_quick\_test
* SPI byte-level examples and explanation of returned bytes
* Quick suggestions for instrumentation and improvements

## **Legend / Definitions**

* ISO\_REG\_\* — register addresses defined in code (0x00..0x06)
* CS — chip select (active LOW) = ISO\_CS\_PIN (SP3I\_CS)
* COMM\_SEL — COMM\_SEL pin (selects serial vs parallel) = ISO\_COMM\_SEL\_PIN (PA8)
* nRST — active-low reset pin = ISO\_nRST\_PIN (PB10)
* OUT\_EN — outputs enable (1 = outputs enabled) = ISO\_OUT\_EN\_PIN (PB4)
* HAL\_SPI\_TransmitReceive(&hspi3, tx, rx, 2, ...) — 2‑byte full-duplex SPI transfer
* Timing notes: delay\_us(x) uses DWT; code often uses 1–5 µs and HAL\_Delay(ms)

## **High-level flowchart (Mermaid)**

flowchart TD

 Start[Start / Reset MCU]

 InitPeripherals[MX\_GPIO\_Init, MX\_USART2\_UART\_Init, MX\_SPI3\_Init]

 DWTInit[DWT\_Delay\_Init]

 SetCSHigh[iso\_cs\_set(HIGH)]

 BootPinState[Set initial MCU pin states: nRST=LOW, OUT\_EN=LOW, COMM\_SEL=LOW]

 PreSerial[Prepare for serial: COMM\_SEL=HIGH, OUT\_EN=LOW, nRST=LOW]

 SerialStart[iso\_serial\_start\_sequence() -> COMM\_SEL=H, release reset, wait, OUT\_EN=H]

 QuickTest[iso\_quick\_test()]

 ClearFilters[Write 0x00 to FILTER03..06]

 Readback[Readback regs 03,00,01,02]

 MainLoop[Main loop: periodic reads; handle fault changes]

 End

 Start --> InitPeripherals --> DWTInit --> SetCSHigh --> BootPinState --> PreSerial --> SerialStart --> QuickTest --> ClearFilters --> Readback --> MainLoop --> End

## **Algorithm (high-level steps)**

1. **MCU & peripheral init**
	* HAL\_Init()
	* SystemClock\_Config()
	* MX\_GPIO\_Init()
	* MX\_USART2\_UART\_Init()
	* MX\_SPI3\_Init()
	* HAL\_Delay(10) to let UART stabilise
	* DWT\_Delay\_Init() for microsecond delays
2. **Initial pin state setup**
	* iso\_cs\_set(HIGH) to make CS inactive
	* Drive MCU outputs to mimic board boot wiring:

		+ iso\_set\_nrst(LOW) (hold ISO1228 in reset)
		+ iso\_set\_out\_en(LOW) (tri-state outputs)
		+ iso\_set\_comm\_sel(LOW) (default)
	* HAL\_Delay(5) to settle
3. **Switch to serial for SPI comms**
	* iso\_set\_comm\_sel(HIGH) (enable serial mode on ISO1228)
	* Ensure iso\_set\_out\_en(LOW) and iso\_set\_nrst(LOW) before starting
	* delay\_us(5) to settle
	* Call iso\_serial\_start\_sequence() — sequence below
4. **iso\_serial\_start\_sequence()**
	* iso\_set\_comm\_sel(HIGH) (ensures serial)
	* iso\_set\_nrst(LOW) then hold (1 µs) (Recommended min 35 nSec)
	* iso\_set\_nrst(HIGH) (release reset)
	* delay\_us(2) (allow SPI logic init) (Recommended min 200 nSec)
	* iso\_set\_out\_en(HIGH) (enable outputs)
	* small delay\_us(2) then ready for SPI transfers
5. **SPI register access primitives**
	* iso\_spi\_write\_reg(reg, data, &r0, &r1)

		+ Build tx[0] = 0x80 | (reg & 0x7F) (MSB=1 indicates WRITE)
		+ tx[1] = data
		+ iso\_cs\_set(LOW); delay\_us(1); HAL\_SPI\_TransmitReceive(...,2); delay\_us(1); iso\_cs\_set(HIGH)
		+ Device returns rx[0], rx[1] — function returns them via rcv0/rcv1
	* iso\_spi\_read\_reg(reg, &pdata, &r0, &r1)

		+ tx[0] = reg & 0x7F (MSB=0 indicates READ)
		+ tx[1] = 0x00
		+ Same CS toggling and transfer as write
		+ pdata is set to rx[1] (per debug log: second response byte is register value)
6. **iso\_quick\_test()**
	* Print START
	* Perform three cycles of raw reads of registers INPUT\_DATA (0x00), WIRE\_BREAK (0x01), FAULT (0x02); log rx0, rx1, rx1 as value
	* Re-init sequence: set COMM\_SEL=H, toggle nRST (low then high), set OUT\_EN=H and read INPUT\_DATA
	* Toggle OUT\_EN low briefly and re-read
	* Change SPI prescaler (fast -> prescaler=16) one read; restore prescaler=2
	* Test changing CPHA from 1EDGE to 2EDGE for one read; restore
	* Print inverted-bit sample for debugging
7. **Filter init and readback**
	* Write 0xBB to FILTER03 and print burst-style debug
	* Write zero to FILTER03..FILTER06
	* Read-back registers FILTER03 (0x03), INPUT\_DATA (0x00), WIRE\_BREAK (0x01), FAULT (0x02) and print
8. **Main loop**
	* Repeatedly read registers: INPUT\_DATA, WIRE\_BREAK, FAULT
	* Print read values (or "SPI read error" if any HAL call fails)
	* If FAULT changed from prev\_fault:

		+ update prev\_fault
		+ If fault != 0: set COMM\_SEL=H, delay\_us(2), call iso\_serial\_start\_sequence() to ensure SPI ready, re-read WIRE\_BREAK and FAULT and print
	* HAL\_Delay(250) (4 Hz sampling)
9. **Error handling**
	* Error\_Handler() disables IRQs and spins in while(1)

## **Pin & SPI transaction timing sequences (detailed)**

### **Serial START sequence (timing used in code)**

1. COMM\_SEL = HIGH (ensures serial mode)
2. nRST = LOW (hold reset)

	* code uses delay\_us(1) (>=35 ns required by spec)
3. nRST = HIGH (release reset)
4. delay\_us(2) (>=200 ns for SPI logic to power up)
5. OUT\_EN = HIGH (enable outputs)
6. delay\_us(2) before starting SPI

**CS behavior for each SPI transfer**

* CS asserted by iso\_cs\_set(LOW) before transfer
* delay\_us(1) then HAL\_SPI\_TransmitReceive(..., 2)
* delay\_us(1) then iso\_cs\_set(HIGH) to release

### **SPI READ (reg 0x00 example)**

* Master transmits: [reg\_byte, 0x00] where reg\_byte = reg & 0x7F (MSB=0)
* Slave returns: [rx0, rx1] where rx1 is documented as register value
* Example: master sends [0x00, 0x00] → receive [0x00, 0xAB] → register value = 0xAB

### **SPI WRITE (reg 0x03 example)**

* Master transmits: [0x80 | reg, data] (MSB=1 indicates WRITE)
* Slave returns two bytes (status or previous data) captured as rx[0], rx[1]
* Example: to write 0xBB to reg 0x03: master sends [0x83, 0xBB]

## **Byte-level map & example logs from code**

* Read INPUT\_DATA (0x00): send [0x00 0x00] rcv [rx0 rx1] → value rx1
* Write FILTER03 (0x03) with 0xBB: send [0x83 0xBB] rcv [rx0 rx1]
* Code prints lines like:

	+ RAW IN: rx0=0x00 rx1=0x00 val=0x00
	+ BURST W: send[83 BB] rcv[00 00]

These example logs match how the iso\_spi\_\* helpers capture and print rx[0] and rx[1] returned by the device.

## **State machine / Pseudocode (condensed)**

// Initialization

HAL\_Init(); SystemClock\_Config(); MX\_GPIO\_Init(); MX\_USART2\_UART\_Init(); MX\_SPI3\_Init();

DWT\_Delay\_Init(); iso\_cs\_set(HIGH);

iso\_set\_nrst(LOW); iso\_set\_out\_en(LOW); iso\_set\_comm\_sel(LOW); HAL\_Delay(5);

// Prepare serial

iso\_set\_comm\_sel(HIGH); iso\_set\_out\_en(LOW); iso\_set\_nrst(LOW); delay\_us(5);

iso\_serial\_start\_sequence(); // releases reset, sets OUT\_EN HIGH

iso\_quick\_test();

// Clear filters

iso\_spi\_write\_reg(0x03, 0xBB, &r0,&r1); // prints

iso\_spi\_write\_reg(0x03,0x00,NULL,NULL); // and others

// Readback registers

iso\_spi\_read\_reg(0x03,&val03,&r0,&r1);

...

// Main loop

prev\_fault = 0xFF;

while(1) {

 if (iso\_spi\_read\_reg(0x00,&in) == OK && ... ) {

 print IN/WB/FAULT

 } else print error

 if (fault != prev\_fault) {

 prev\_fault = fault;

 if (fault != 0) {

 iso\_set\_comm\_sel(HIGH); delay\_us(2);

 iso\_serial\_start\_sequence();

 iso\_spi\_read\_reg(0x01,&wb2...);

 iso\_spi\_read\_reg(0x02,&fault2...);

 }

 }

 HAL\_Delay(250);

}

## **iso\_quick\_test() flow (more detailed)**

1. Print header
2. For i=0..2: read INPUT\_DATA, WIRE\_BREAK, FAULT (using iso\_spi\_read\_reg) and print rx0, rx1, value for each
3. Re-init: COMM\_SEL=H; nRST low->high pulse; OUT\_EN=H → delay
4. Read INPUT\_DATA and print
5. Toggle OUT\_EN low for 5 ms then high and read INPUT\_DATA
6. Temporarily change SPI prescaler to SPI\_BAUDRATEPRESCALER\_16 and do one read; restore prescaler
7. Temporarily change CPHA to SPI\_PHASE\_2EDGE and do one read; restore
8. Print inverted view sample and footer

## **Notes & suggestions**

* CS timing: the code asserts CS -> small delay -> SPI transfer -> small delay -> deassert CS. This is safe; keep delay\_us(1) as the device seems tolerant.
* When changing HAL SPI init parameters (prescaler/phase), the code disables SPI, changes fields, re-inits, and re-enables — this is correct.
* If you need guaranteed atomic sequences between multiple registers, consider using a burst mode (if device supports) or keep CS asserted across multiple HAL\_SPI\_TransmitReceive calls.
* Add timeout or error counters when repeated HAL\_SPI\_TransmitReceive calls fail to allow recovery (e.g. re-init SPI or toggle reset).
* Consider reporting rx0 semantics: it often appears zero in logs — check datasheet for meaning (status flags?).

## **What I included in this document (checklist)**

* Flowchart (mermaid)
* Full algorithm / step-by-step walkthrough
* Detailed pin & SPI transaction timings
* Pseudocode and state machine for main loop and quick test
* Byte-level examples and how the code interprets rx[0]/rx[1]
* Suggestions for robustness

If you want, I can:

* Convert the mermaid flowchart into an image and attach it
* Produce a printable PDF of this document
* Generate an annotated sequence diagram (per-pin timing) as an image
* Add comments inline in your source to match each step