iqMesh SPI (VISION SPI) Integration — Step-by-Step Debug Guide

0) Preconditions (do these once)

- **Build flags / optimization:** Start with no LTO and -O0/-Og, enable full debug symbols. (Debug mode)
- **Single caller:** Ensure iqMesh_task() is called from **one** context only (main loop or a dedicated RTOS task). No concurrent calls.
- **Scheduler cadence:** Call iqMesh_task() regularly (e.g., in main or every ms). Starvation will stall the state machine.
- **Pin mapping:** Double-check IRQ, CS, MOSI, MISO, SCK match your board schematic and voltage levels (no 5V on 3V3 parts).
- **SPI mode/speed:** Match CPOL/CPHA and max clock of the module. If unsure, start slow (e.g., 1 MHz) and mode 0 or the mode your example uses.

---

1) Library initialization

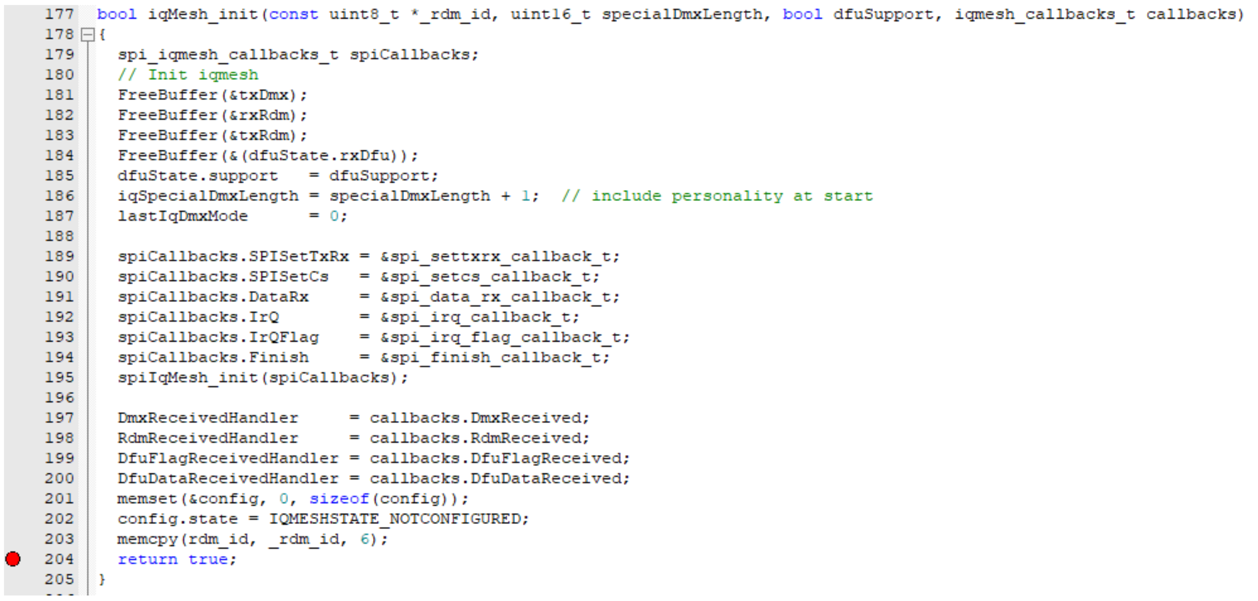
Break: libIqMeshSpiInterface.c:204 in iqMesh_init.

Expect:

- Function is called exactly once after boot.

If FAIL:

- Confirm you call iqMesh_init() before the first iqMesh_task().



```
177  bool iqMesh_init(const uint8_t *_rdm_id, uint16_t specialDmxLength, bool dfuSupport, iqmesh_callbacks_t callbacks)
178  {
179      spi_iqmesh_callbacks_t spiCallbacks;
180      // Init iqmesh
181      FreeBuffer(&txDmx);
182      FreeBuffer(&rxRdm);
183      FreeBuffer(&txRdm);
184      FreeBuffer(&(dfuState.rxDfu));
185      dfuState.support   = dfuSupport;
186      iqSpecialDmxLength = specialDmxLength + 1;  // include personality at start
187      lastIqDmxMode      = 0;
188
189      spiCallbacks.SPISetTxRx = &spi_settxrx_callback_t;
190      spiCallbacks.SPISetCs   = &spi_setcs_callback_t;
191      spiCallbacks.DataRx     = &spi_data_rx_callback_t;
192      spiCallbacks.IrQ        = &spi_irq_callback_t;
193      spiCallbacks.IrQFlag    = &spi_irq_flag_callback_t;
194      spiCallbacks.Finish     = &spi_finish_callback_t;
195      spiIqMesh_init(spiCallbacks);
196
197      DmxReceivedHandler     = callbacks.DmxReceived;
198      RdmReceivedHandler     = callbacks.RdmReceived;
199      DfuFlagReceivedHandler = callbacks.DfuFlagReceived;
200      DfuDataReceivedHandler = callbacks.DfuDataReceived;
201      memset(&config, 0, sizeof(config));
202      config.state = IQMESHSTATE_NOTCONFIGURED;
203      memcpy(rdm_id, _rdm_id, 6);
204      return true;
205  }
```

---

2) Task loop is running

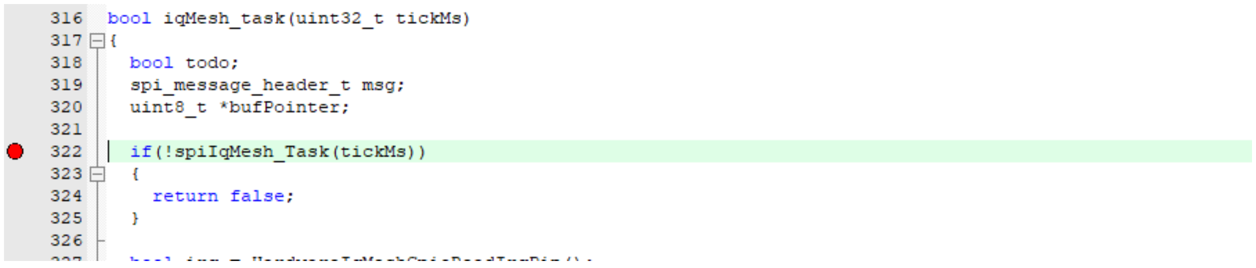
Break: libIqMeshSpiInterface.c:322 in iqMesh_task.

Expect:

- Breakpoint hits repeatedly at your chosen cadence.
- No long gaps (watch a counter or timestamp to detect stalls).

If FAIL:

- Wire iqMesh_task() into main loop/RTOS task with adequate priority and stack.
- Make sure no blocking calls (e.g., long delays) run at higher priority and starve this task.



```
316  bool iqMesh_task(uint32_t tickMs)
317  {
318    bool todo;
319    spi_message_header_t msg;
320    uint8_t *bufPointer;
321
322    if(!spiIqMesh_Task(tickMs))
323    {
324      return false;
325    }
326
327    bool irq = HardwareIqMeshGpioReadIrqPin();
```

---

3) IRQ pin read function

Function: Check Function HardwareIqMeshGpioReadIrqPin() (in your main.c example).

Expect:

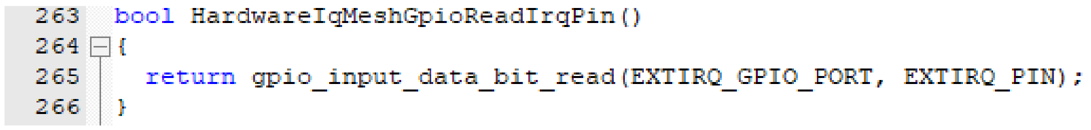
- Reads the **actual** IRQ line (correct port/pin).
- Correct logic polarity (active level matches hardware).
- Input mode and pull (pull-up/down) are configured correctly.

Probe:

- **Code:** Set a breakpoint inside HardwareIqMeshGpioReadIrqPin() and watch the returned value change when traffic occurs.
- **HW:** Logic analyzer/oscilloscope on the IRQ pin. You should see edges when iqMesh needs attention.

If FAIL:

- Fix the GPIO init (mode=Input, pull as required).
- Invert the read value if your board inverts the signal.
- Check level shifting / voltage domain compatibility.



```
bool HardwareIqMeshGpioReadIrqPin()
{
  return gpio_input_data_bit_read(EXTIRQ_GPIO_PORT, EXTIRQ_PIN);
}
```

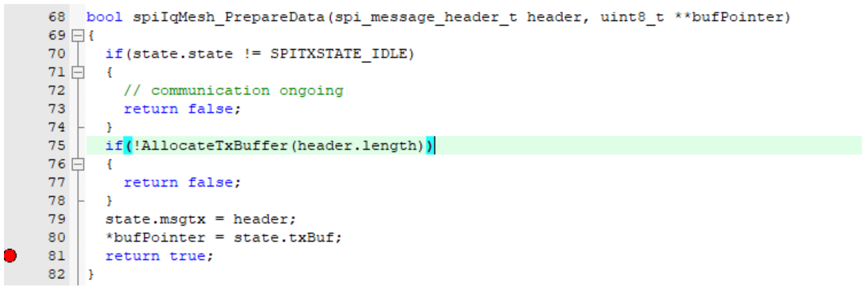
---

4) TX buffer & packet allocation
Break: spi_iqmesh.c:81 in spiIqMesh_PrepareData().
Expect:

- Function is hit whenever a packet is prepared.
- AllocateTxBuffer() returns **true**.

If FAIL (not called):

- Step back to where AllocateTxBuffer() is used and inspect why it returns false.
- **Heap check:** Increase heap or switch to static buffers. Watch for malloc failure or fragmentation.



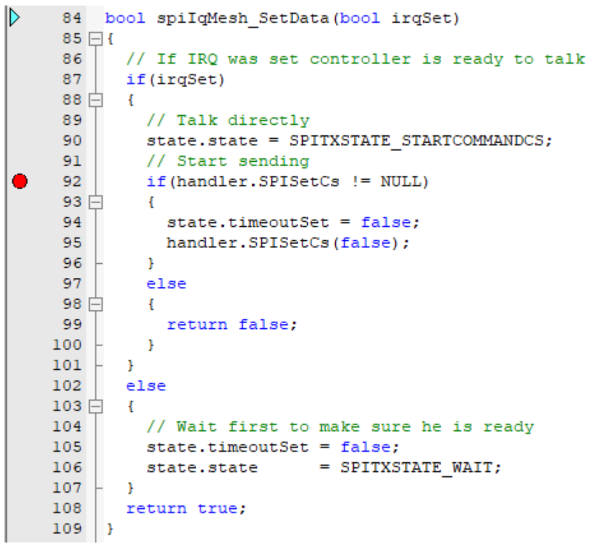
```
68  bool spiIqMesh_PrepareData(spi_message_header_t header, uint8_t **bufPointer)
69  {
70    if(state.state != SPITXSTATE_IDLE)
71    {
72      // communication ongoing
73      return false;
74    }
75    if(!AllocateTxBuffer(header.length))
76    {
77      return false;
78    }
79    state.msgtx = header;
80    *bufPointer = state.txBuf;
81    return true;
82  }
```

5) CS set via handler
Break: libIqMeshSpiInterface.c:92 in spiIqMesh_SetData() — confirm handler.SPISetCs(...) is invoked.
Expect:

- Called prior to each SPI transaction.
- Correct CS polarity (usually active-low).

If FAIL:

- Recheck the IRQ handling path (IRQ drives the need to transact).
- Return to step 3 (bad IRQ prevents transactions).
- Verify your handler struct is correctly assigned during init.



```
84   bool spiIqMesh_SetData(bool irqSet)
85   {
86     // If IRQ was set controller is ready to talk
87     if(irqSet)
88     {
89       // Talk directly
90       state.state = SPITXSTATE_STARTCOMMANDCS;
91       // Start sending
92       if(handler.SPISetCs != NULL)
93       {
94         state.timeoutSet = false;
95         handler.SPISetCs(false);
96       }
97       else
98       {
99         return false;
100      }
101    }
102    else
103    {
104      // Wait first to make sure he is ready
105      state.timeoutSet = false;
106      state.state       = SPITXSTATE_WAIT;
107    }
108    return true;
109  }
```

6) Hardware CS toggling (board function)

Function: HardwareIqMeshSpiSetCsPin(bool active) in your main.c. (Example)

Break: HardwareIqMeshSpiSetCsPin and check if it gets called.

Expect:

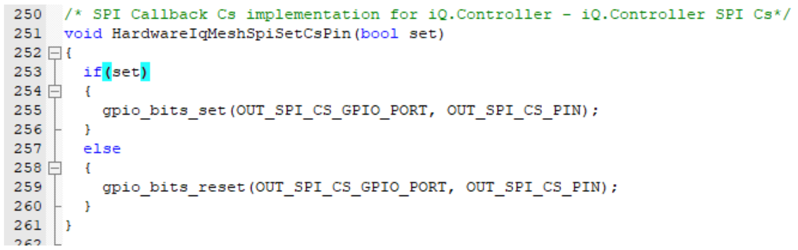
- CS pin toggles (active during the full SPI frame).

Probe (HW):

- LA/scope on CS line: one clean low (or high, per polarity) spanning the entire transfer.

If FAIL:

- Fix pin mapping, polarity, and GPIO speed.



```c
/* SPI Callback Cs implementation for iQ.Controller - iQ.Controller SPI Cs*/
void HardwareIqMeshSpiSetCsPin(bool set)
{
  if(set)
  {
    gpio_bits_set(OUT_SPI_CS_GPIO_PORT, OUT_SPI_CS_PIN);
  }
  else
  {
    gpio_bits_reset(OUT_SPI_CS_GPIO_PORT, OUT_SPI_CS_PIN);
  }
}
```

7) TX/RX data path selection

Function: HardwareIqMeshSpiSetTxRx(...) in your main.c. (Example)

Break: HardwareIqMeshSpiSetCsPin and check if it gets called.

Expect:

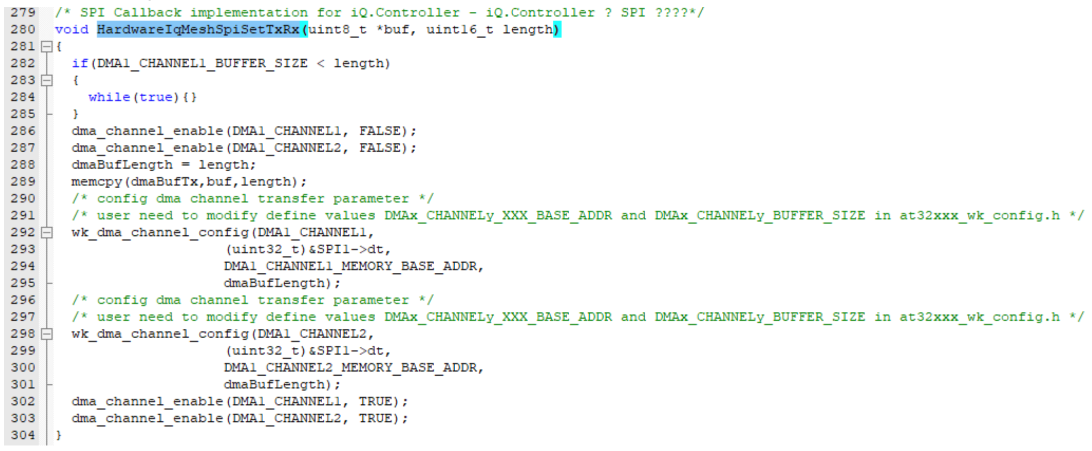
- MOSI shows activity during transfer.
- SCK present and clean; matches configured SPI mode.
- MISO shows non-tristated data while CS is active.

Probe (HW):

- LA/scope on MOSI/MISO/SCK. Sample around mid-bit per your CPHA.

If FAIL:

- Lower SPI speed.
- Verify CPOL/CPHA and bit order.



```c
/* SPI Callback implementation for iQ.Controller - iQ.Controller ? SPI ????*/
void HardwareIqMeshSpiSetTxRx(uint8_t *buf, uint16_t length)
{
  if(DMA1_CHANNEL1_BUFFER_SIZE < length)
  {
    while(true){}
  }
  dma_channel_enable(DMA1_CHANNEL1, FALSE);
  dma_channel_enable(DMA1_CHANNEL2, FALSE);
  dmaBufLength = length;
  memcpy(dmaBufTx,buf,length);
  /* config dma channel transfer parameter */
  /* user need to modify define values DMAx_CHANNELy_XXX_BASE_ADDR and DMAx_CHANNELy_BUFFER_SIZE in at32xxx_wk_config.h */
  wk_dma_channel_config(DMA1_CHANNEL1,
                        (uint32_t)&SPI1->dt,
                        DMA1_CHANNEL1_MEMORY_BASE_ADDR,
                        dmaBufLength);
  /* config dma channel transfer parameter */
  /* user need to modify define values DMAx_CHANNELy_XXX_BASE_ADDR and DMAx_CHANNELy_BUFFER_SIZE in at32xxx_wk_config.h */
  wk_dma_channel_config(DMA1_CHANNEL2,
                        (uint32_t)&SPI1->dt,
                        DMA1_CHANNEL2_MEMORY_BASE_ADDR,
                        dmaBufLength);
  dma_channel_enable(DMA1_CHANNEL1, TRUE);
  dma_channel_enable(DMA1_CHANNEL2, TRUE);
}
```

8) DMA RX completion path

Break: main.c in your _SPI_DMA_RxTransferCompleteCallback.

Watch:

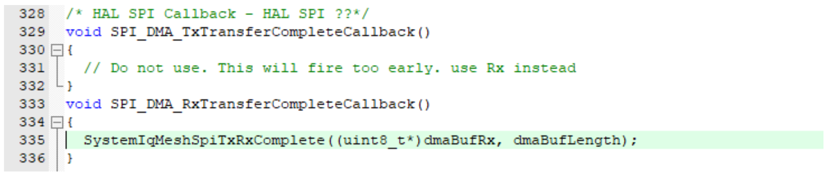
- dmaBufLength > 0.
- dmaBufRx[0] **not** 0x00 and **not** 0xFF when dmaBufLength == 1.

Expect:

- Callback fires per transaction.
- Buffer contents change between transactions.

If FAIL:

- Ensure DMA is properly linked to the SPI RX stream.
- Verify CS remains asserted until the **last** RX byte completes.



```
/* HAL SPI Callback - HAL SPI ??*/
void SPI_DMA_TxTransferCompleteCallback()
{
  // Do not use. This will fire too early. use Rx instead
}
void SPI_DMA_RxTransferCompleteCallback()
{
  SystemIqMeshSpiTxRxComplete((uint8_t*)dmaBufRx, dmaBufLength);
}
```

---

9) External IRQ line (EXTI) callback chain

Breaks & Flow:

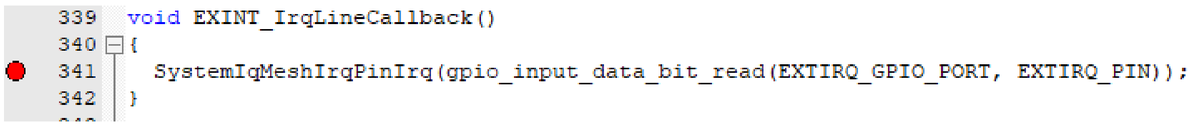
- Break inside your EXTI callback (e.g., EXINT_IrqLineCallback in main.c).
- That callback should call SystemIqMeshIrqPinIrq(irqState).
- Which in turn must cause iqMesh_extIrqInterupt() to run.
- **Break:** libIqMeshSpiInterface.c:208 in iqMesh_extIrqInterupt.

Expect:

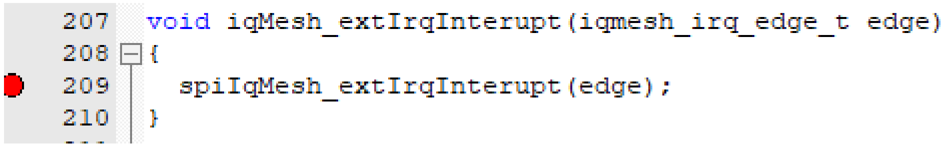
- All three points are hit on edges.
- The irqState matches what HardwareIqMeshGpioReadIrqPin() reads.

If FAIL:

- Configure EXTI for the **correct edge(s)** (rising and falling).
- Clear EXTI pending bits.
- Check NVIC priority (EXTI must preempt long-running code).
  Best highest IRQ priority. IRQ is short.



```
void EXINT_IrqLineCallback()
{
  SystemIqMeshIrqPinIrq(gpio_input_data_bit_read(EXTIRQ_GPIO_PORT, EXTIRQ_PIN));
}
```



```
207  void iqMesh_extIrqInterupt(iqmesh_irq_edge_t edge)
208  {
209    spiIqMesh_extIrqInterupt(edge);
210  }
```

---

10) State machine verification
Primary Break: libIqMeshSpiInterface.c:331 inside iqMesh_task to inspect config.state.
Expect:
During normal operation, config.state cycles among:
IQMESHSTATE_CHECKIRQ, IQMESHSTATE_RUN, IQMESHSTATE_RUNSET, IQMESHSTATE_UPDATEIRQ (as your note indicates).
If state looks wrong / stuck:

- Set another **Break:** libIqMeshSpiInterface.c:357 and inspect config.module_rdm_id vs your rdm_id.
    - **If different:** SPI **read** likely failing (bad MISO/CPOL/CPHA/CS timing/DMA RX).
    - Re-verify steps 6–8, especially MISO and the RX callback path.