Avnet Visible Things UART Example

by

Michael C. Li
https://www.miketechuniverse.com/

4/5/2018

e2 Studio Version: 5.3.1.002

Project Summary

Board:	gw002_rev1_3
Device:	R7FS7G27H2A01CBD
Toolchain:	GCC ARM Embedded
Toolchain Version:	4.9.3.20150529
SSP Version:	1.2.0

PMOD Configuration for this example

Gateway PMOD	I2C I2C	Type 1 GPIO	Type 2 SPI	Type 2A Expanded SPI	Type 3 UART	Type 4 UART	Type 4A Expanded UART	Type 5 H-Bridge	Type 6 Dual H-Bridge
1	Y	Y	Y	Y	Y	Y	Y	N	N
2	Y	Y	Y	Y	Y	Y	Y	N	N
3	Y	Y	Y	Y	Y	Y	Y	N	N
4	Y	Y	Y	Y	Y	Y	Y	N	N
5	N	Y	Y	Y	N	Y	Y	Y	Y
6	Y	Y	N	N	N	N	N	N	N

Table 3 – Pmod Compatibility Chart

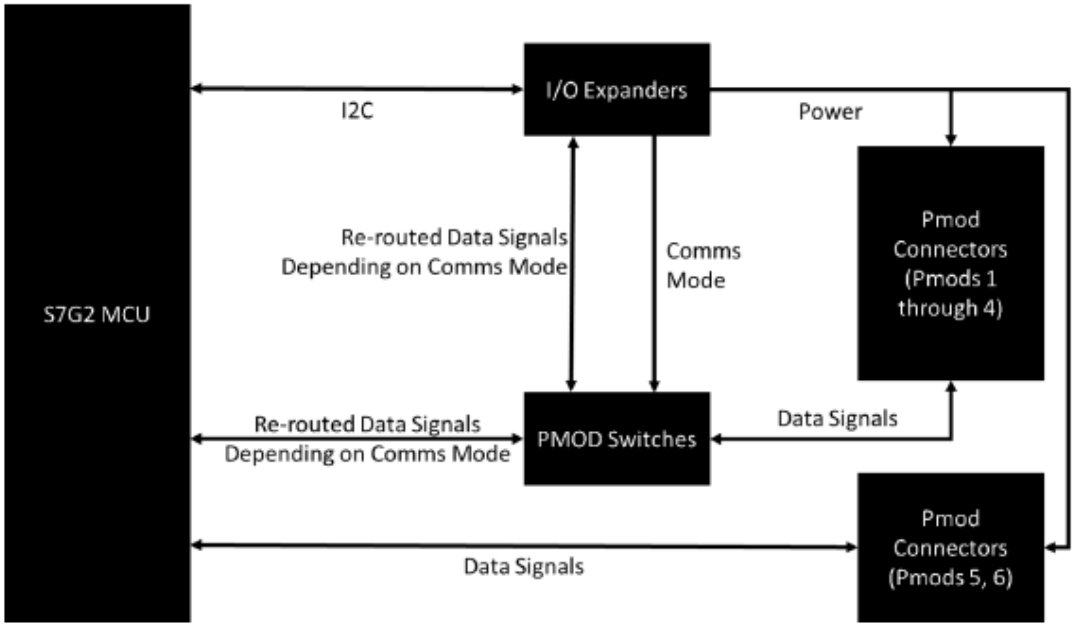


Figure 30 – Pmod, I/O Expander Interconnections Diagram

Hardware Setup

Open or close (either way is ok for SPI mode)

Closed
(SPI mode)

BMC150 Module

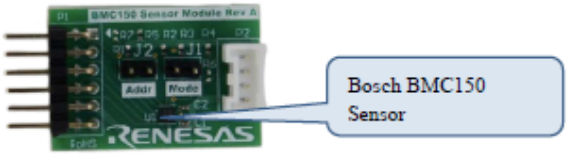


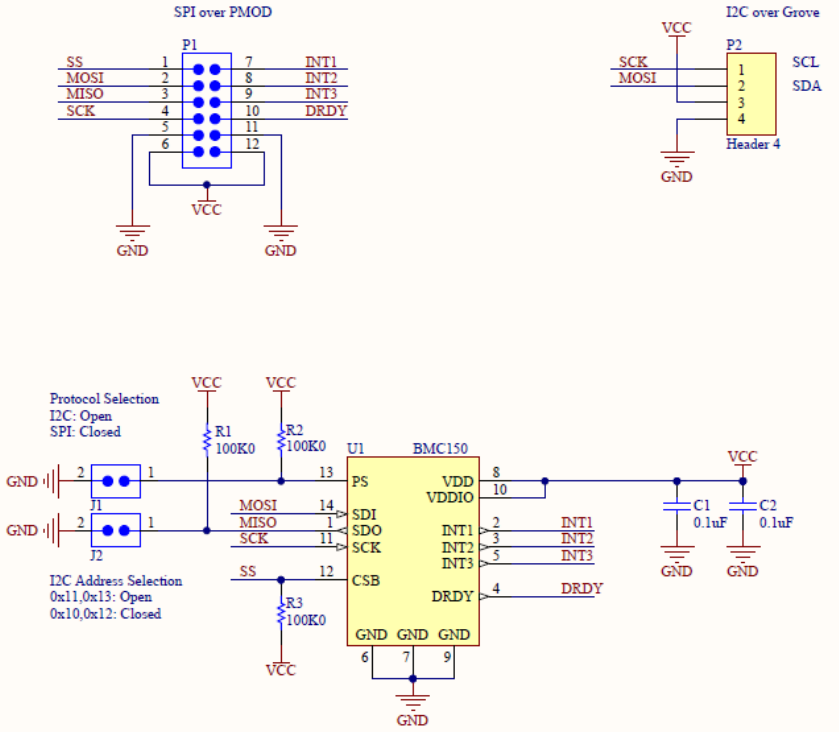
Figure 1.3 Bosch BMC150 sensor board

The active interface to be used is configured by jumper J1:

- Open = I²C (Grove)
- Closed = SPI (PMOD).

In case the I²C is selected, the device addresses are dependent on the state of jumper J2:

- Open = 0x11, 0x13
- Closed = 0x10, 0x12.



IC Chip Pin Description (not Module Pin)

Table 52: Pin description

Pin	Name	I/O Type	Sensor	Description	Connect to		
					SPI4W	SPI3W	I²C
1	SDO	Out	Mag+Acc	SPI: Data out	SDO / MISO	DNC (float)	GND for default address
2	INT1	Out	Acc	Interrupt output #1	INT 1 input or DNC if unused		
3	INT2	Out	Acc	Interrupt output #2	INT2 input or DNC if unused		
4	DRDY	Out	Mag	Data ready	DRDY input or DNC if unused		
5	INT3	Out	Mag	Interrupt output #3	INT3 input or DNC if unused		
6	GND	Supply	Mag+Acc	Ground	GND		
7	GND	Supply	Mag+Acc	Ground	GND		
8	VDD	Supply	Mag+Acc	Supply voltage	V_{DD}		
9	GND	Supply	Mag+Acc	Ground	GND		
10	VDDIO	Supply	Mag+Acc	I/O voltage	V_{DDIO}		
11	SCK	In	Mag+Acc	Serial clock	SCK	SCK	SCL
12	CSB	In	Mag+Acc	Chip Select	CSB	CSB	DNC (float) or V_{DDIO}
13	PS	In	Mag+Acc	Protocol select	GND	GND	V_{DDIO}
14	SDI	In/Out	Mag+Acc	SPI: Data in, I²C: Data	SDI / MOSI	SDA	SDA

PMOD 1 Port

GPIO Type 1 Pmods

If clearance is not an issue, two GPIO Type 1 standard Pmods can be used together when the connector is configured in UART/SPI Comms Mode. **Do not turn a second Type 1 Pmod upside down to get it to fit! Damage may occur as a result of inserting a Pmod upside down!**

	Pmod		S7G2	
	Pin	Pin Function	Port Pin	Pin Function
Upper Row	1	GPIO	PA05	GPIO
	2	GPIO	PA02	GPIO
	3	GPIO	PA03	GPIO
	4	GPIO	PA04	GPIO
	5	GND	GND	GND
	6	VCC (3.3V)	VCC (3.3V)	VCC (3.3V)
Lower Row	1	GPIO	P400	GPIO
	2	GPIO	I/O Expander	GPIO
	3	GPIO	I/O Expander	GPIO
	4	GPIO	I/O Expander	GPIO
	5	GND	GND	GND
	6	VCC (3.3V)	VCC (3.3V)	VCC (3.3V)

Table 7 – Pmod1 GPIO Type 1 Pin Connections (UART/SPI Mode)

PMOD 2 Port

UART/SPI Comms Mode

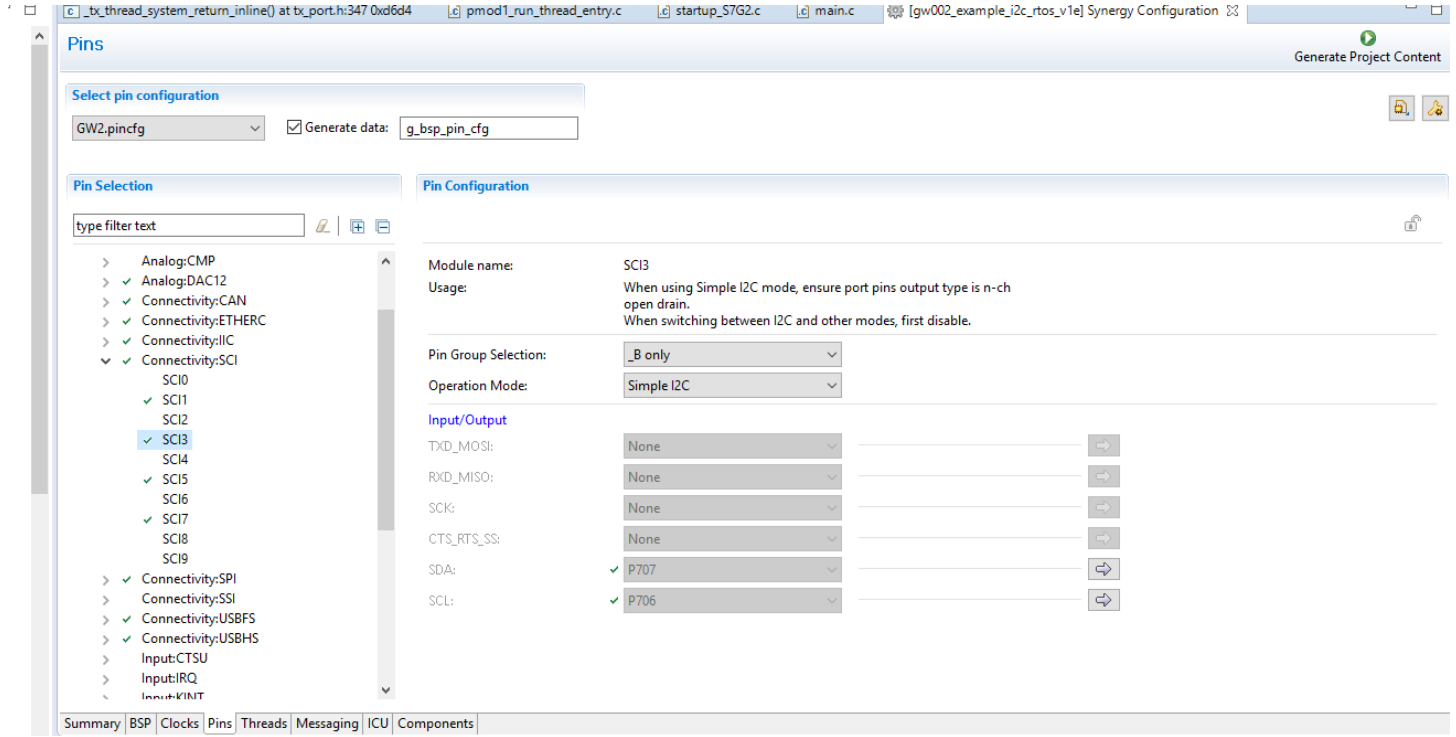
UART Type 4 and SPI Type 2 Pmods

UART Type 4 Pmods and SPI Type 2 Pmods only have 6 pins instead of the 12 provided by the Gateway's connector. Insert these Pmods into the upper row of the connector (connector pins 1 through 6).

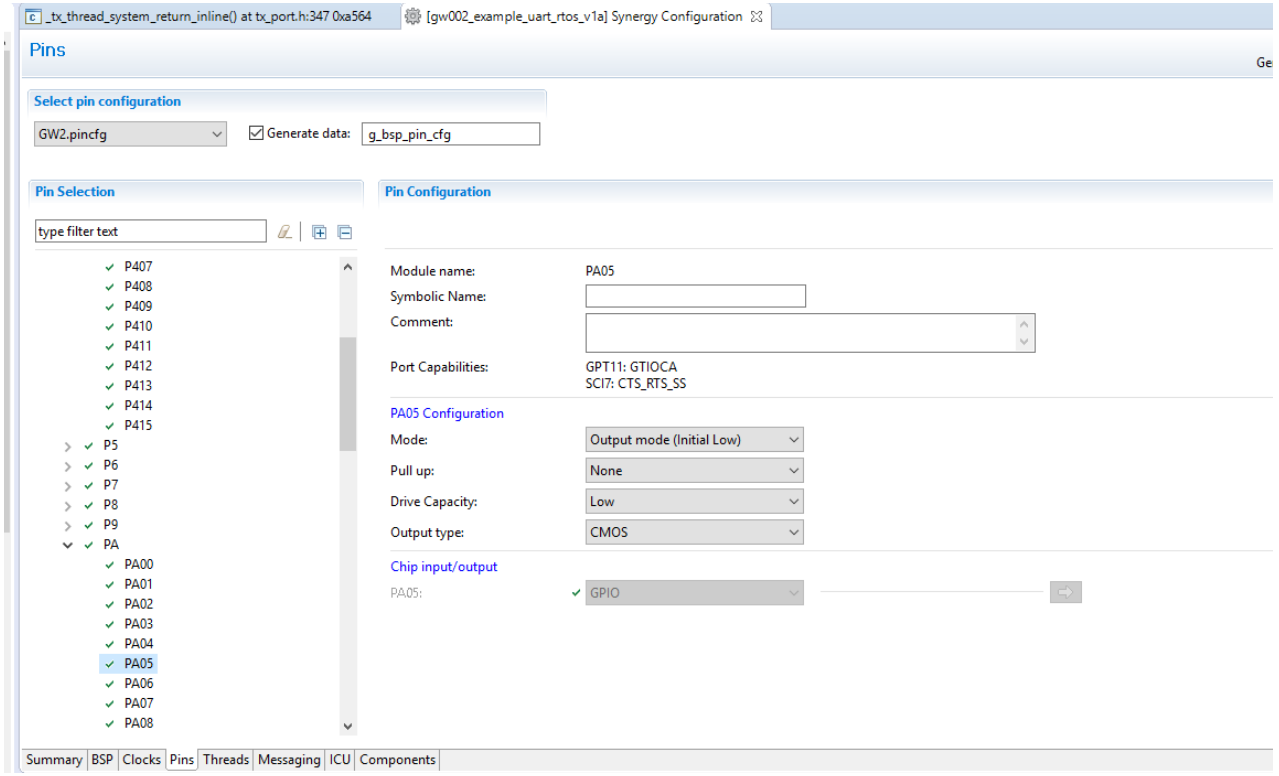
Pmod			S7G2		
Pin	Pin Function by Type		Port Pin	Pin Function	
	UART (4, 4A)	SPI (2, 2A)		UART	SPI
1	RTS	CS	PB02	CTS	CTS_RTS_SS
2	RX	MISO	PB04	TXD_MOSI	TXD_MOSI
3	TX	MOSI	PB05	RXD_MISO	RXD_MISO
4	CTS	SCK	PB03	RTS	SCK
5	GND	GND	GND	GND	GND
6	VCC (3.3V)	VCC (3.3V)	VCC (3.3V)	VCC (3.3V)	VCC (3.3V)

Table 12 – Pmod2 UART Type 4 and SPI Type 2 Pin Connections

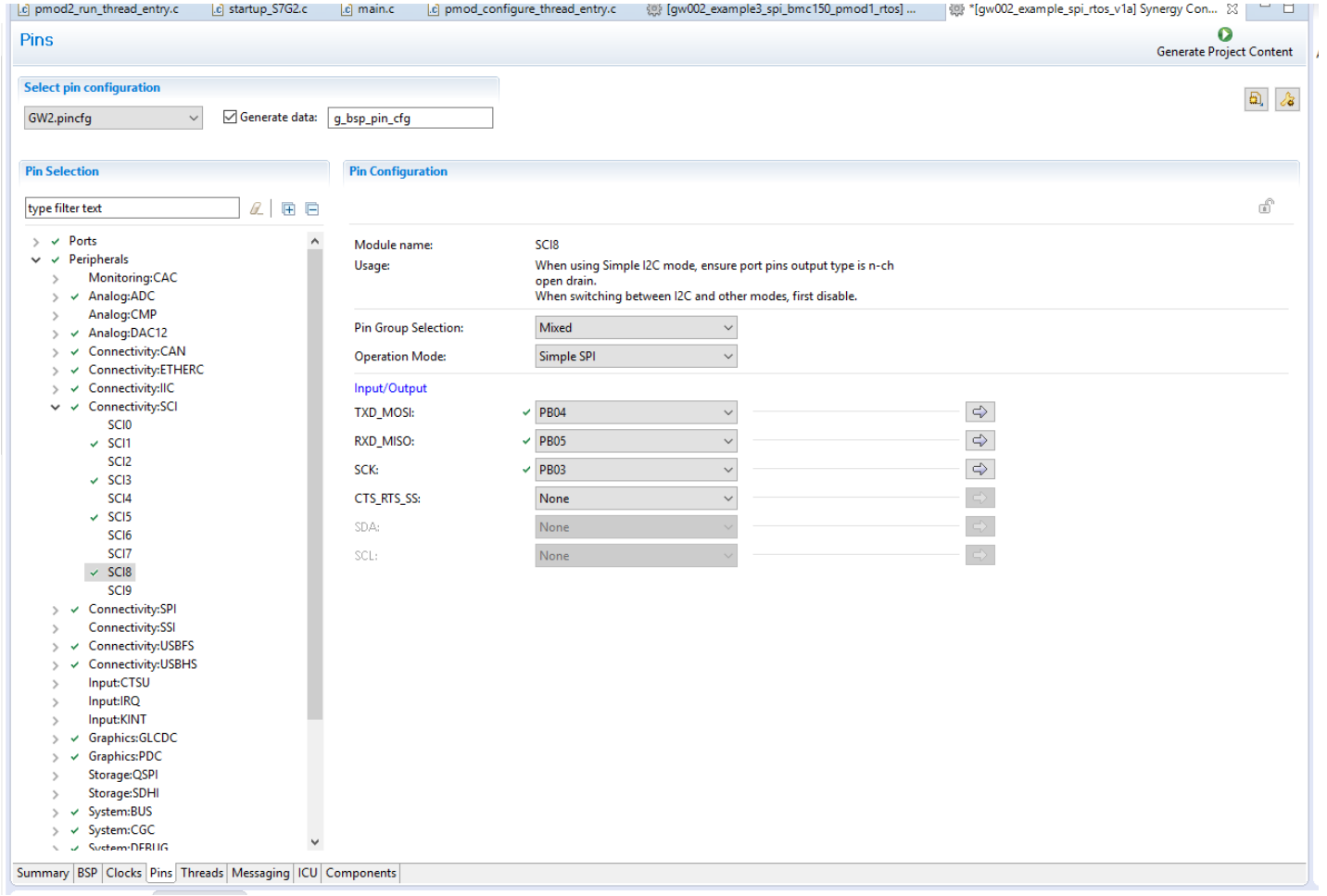
IO Expanders U18/U19 I2C port



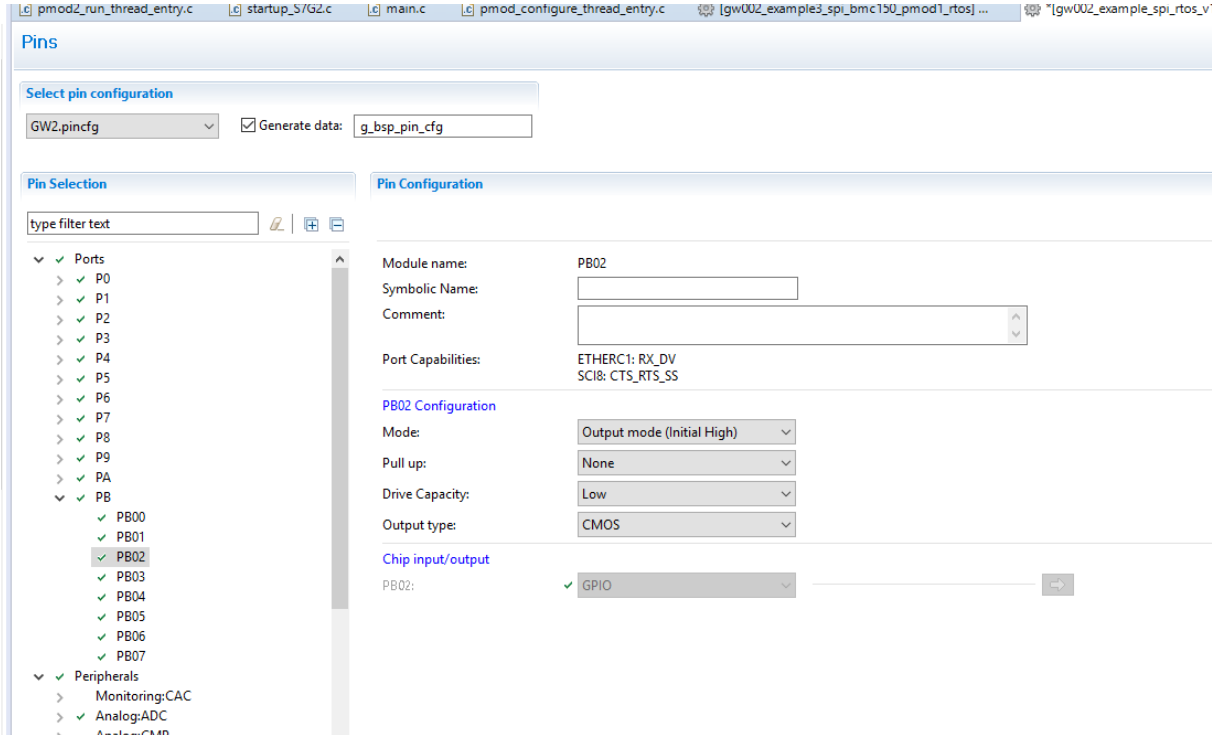
PMOD 1 : PA02/3/4/5 Output Mode (P400 output only)



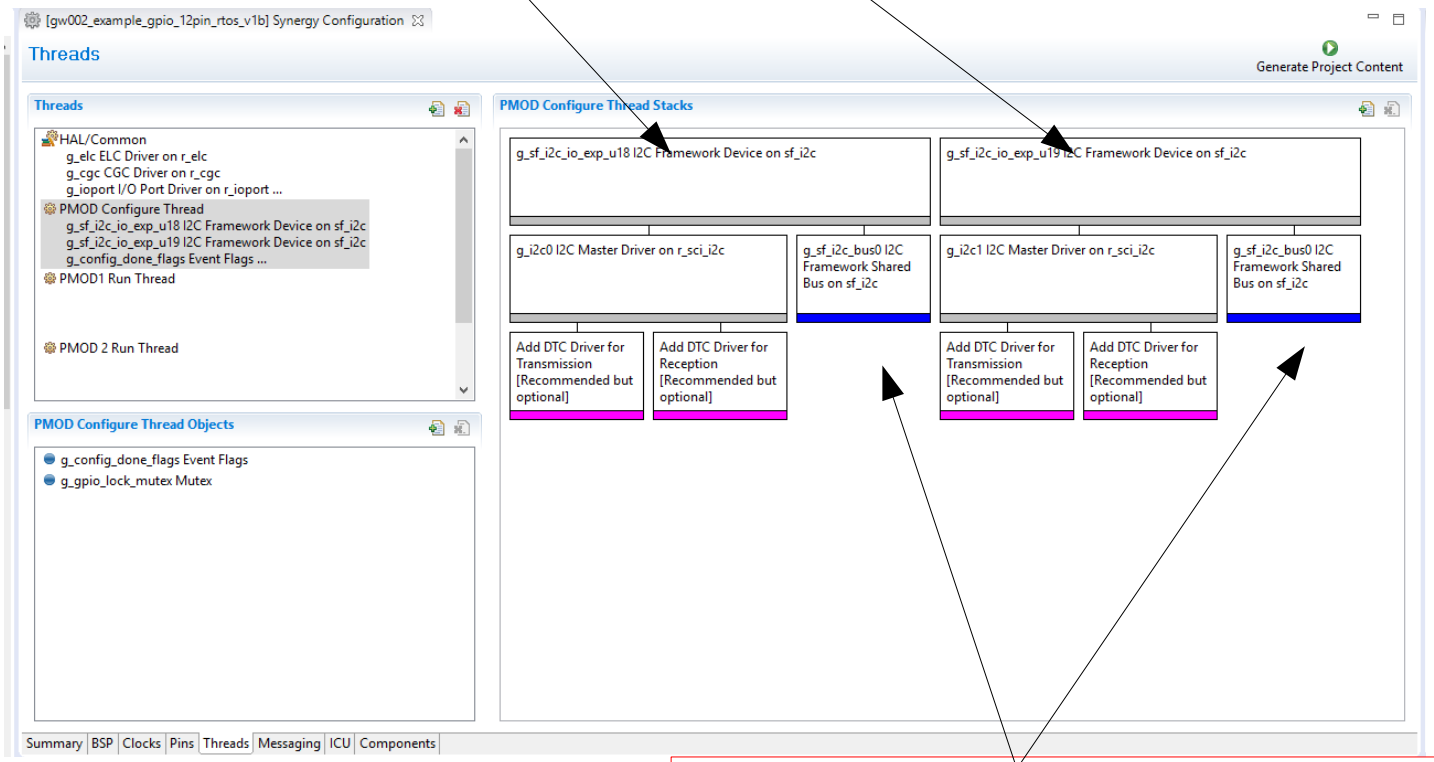
Configure PMOD 2 (Simple SPI)



PMOD2 : SS pin (GPIO out mode)

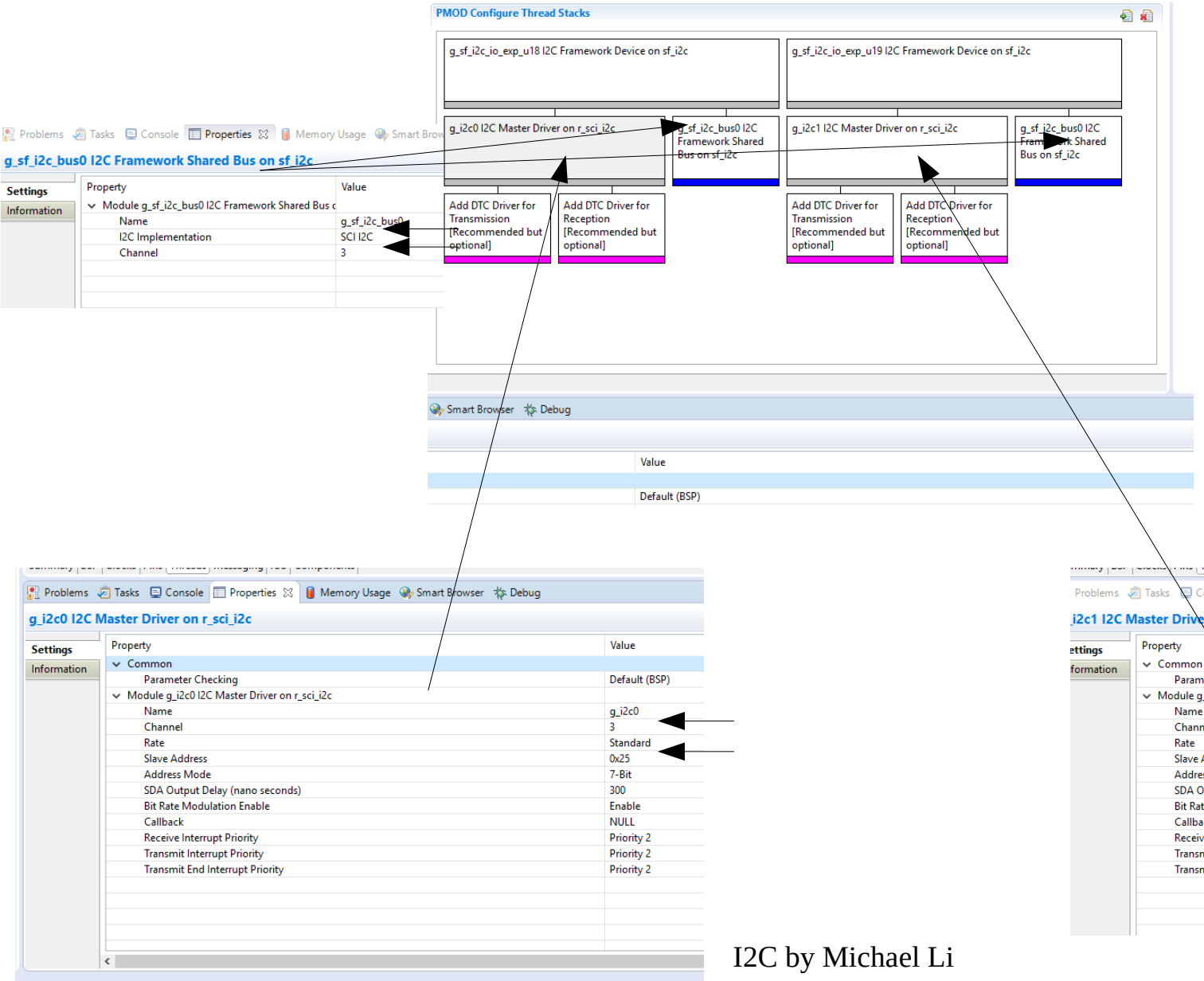


PMOD Configure Thread
(Create U18/U19 I2C Framework Drivers)



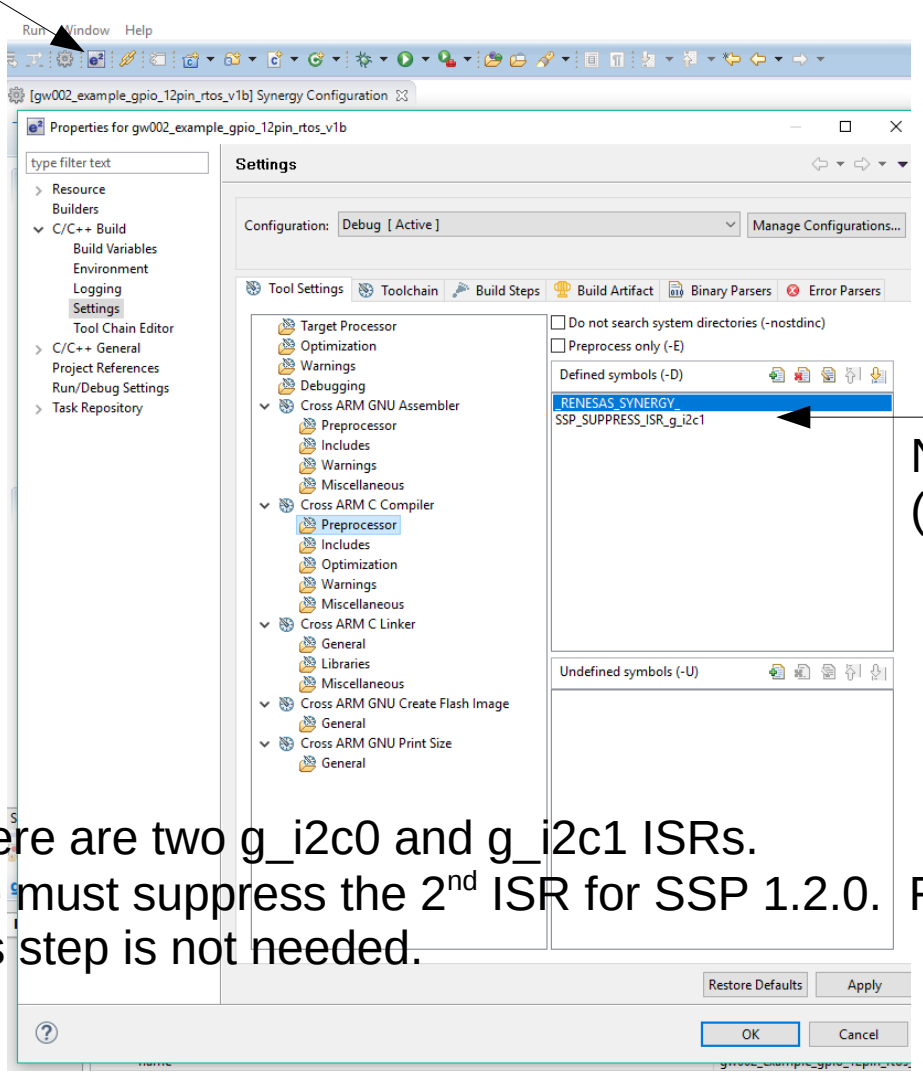
Both drivers share the same g_sf_i2c_bus0 bus.

Properties


PMOD Configure Thread Stacks
g_sf_i2c_io_exp_u18 I2C Framework Device on sf_i2c
g_sf_i2c_io_exp_u19 I2C Framework Device on sf_i2c
g_i2c0 I2C Master Driver on r_sci_i2c
g_sf_i2c_bus0 I2C Framework Shared Bus on sf_i2c
g_i2c1 I2C Master Driver on r_sci_i2c
Add DTC Driver for Transmission [Recommended but optional]
Add DTC Driver for Reception [Recommended but optional]
Problems Tasks Console Properties Memory Usage Smart Browser Debug
g_sf_i2c_bus0 I2C Framework Shared Bus on sf_i2c
Settings
Information
Property
Value
Module g_sf_i2c_bus0 I2C Framework Shared Bus
Name
g_sf_i2c_bus0
I2C Implementation
SCI I2C
Channel
3
Default (BSP)
g_i2c0 I2C Master Driver on r_sci_i2c
Common
Parameter Checking
Default (BSP)
Module g_i2c0 I2C Master Driver on r_sci_i2c
Name
g_i2c0
Channel
3
Rate
Standard
Slave Address
0x25
Address Mode
7-Bit
SDA Output Delay (nano seconds)
300
Bit Rate Modulation Enable
Enable
Callback
NULL
Receive Interrupt Priority
Priority 2
Transmit Interrupt Priority
Priority 2
Transmit End Interrupt Priority
Priority 2
i2c1 I2C Master Driver on r_sci_i2c
Module g_i2c1 I2C Master Driver on r_sci_i2c
Name
g_i2c1
Slave Address
0x27

Compiler Pre-processor Requirement

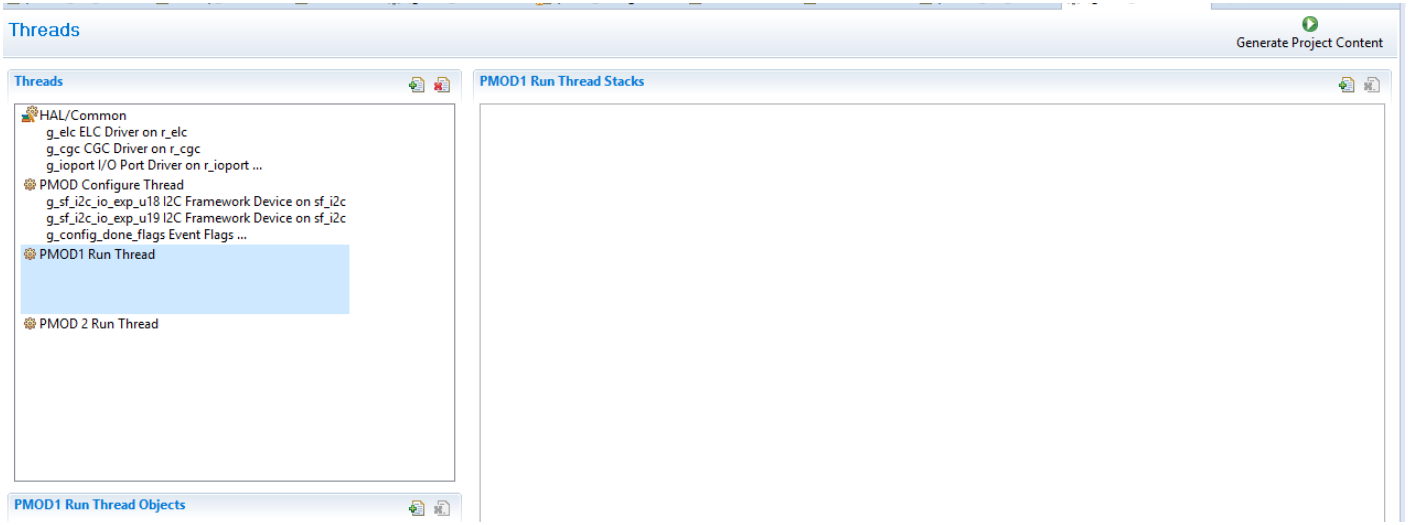
Click this icon!



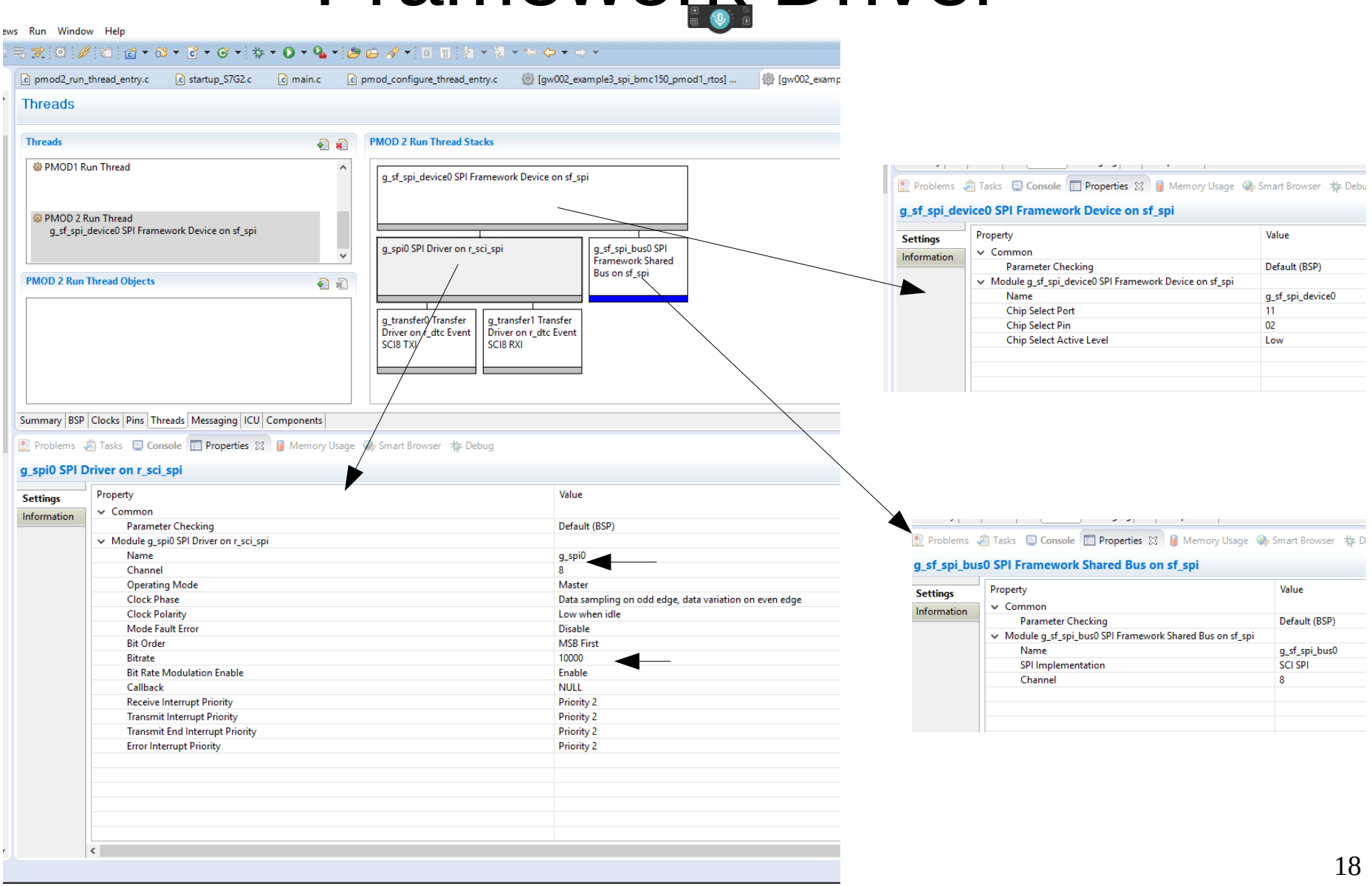
Need to Suppress 2nd i2c ISR (needed for SSP 1.2.0 only)

There are two g_i2c0 and g_i2c1 ISRs.
We must suppress the 2nd ISR for SSP 1.2.0. For later SSP versions, this step is not needed.

PMOD 1 Thread has no drivers needed because the port is configured as a GPIO port.



PMOD 2 Run Thread : SPI Framework Driver



Configure PMOD 1 and PMOD 2

```
 *
 *  input: None
 *  Return : None
 *
 */
void setup_ioexp_registers(void) {


    ////////////////////////////////////////////////
    ///  U18 setup for PMOD1 : GPIO Type 1 12 pin configuration
    ///                        Set IO1-5 output pins (S7G2 MCU pins)
    ///                        Set IO6/7/8 output pins
    ///                        Set IO6/7/8 =001
    ////////////////////////////////////////////////
    pmod_bus_type_cfg[PMOD1_PORT] = GPIO_TYPE1_12PINS_COML;

    u18port0outreg.bit.pmod1_comms = set_pmod_com_bit(pmod_bus_type_cfg[PMOD1_PORT]);

    u18port0cfg.bit.pmod1_reset_io6 = SET_CFG_PIN_OUTPUT;
    u18port1cfg.bit.pmod1_io7       = SET_CFG_PIN_OUTPUT;
    u18port1cfg.bit.pmod1_io8       = SET_CFG_PIN_OUTPUT;

    u18port0outreg.bit.pmod1_reset_io6  = 1;  // make 0010 0000 initially
    u18port1outreg.bit.pmod1_io7        = 0;
    u18port1outreg.bit.pmod1_io8        = 0;

    ////////////////////////////////////////////////
    ///  U19 setup for PMOD1 : Power enabled
    ////////////////////////////////////////////////
    u19port0outreg.bit.pmod1_power = 1;  // pmod1 power enabled.

    ////////////////////////////////////////////////
    ///  U18 setup for PMOD2 : UART 4 6 pin configuration
    ////////////////////////////////////////////////
    pmod_bus_type_cfg[PMOD2_PORT] = SPI_TYPE2_6PINS_COML;


    u18port0outreg.bit.pmod2_comms = set_pmod_com_bit(pmod_bus_type_cfg[PMOD2_PORT]);

    ////////////////////////////////////////////////
    ///  U19 setup for PMOD2 : Power enabled
    ////////////////////////////////////////////////
    u19port0outreg.bit.pmod2_power = 1;  // pmod2 power enabled.

}
```

PMOD 1 Thread Entry

rt_rtos_v1c] Synergy Configuration | pmod_configure_thread_entry.c | pmod1_run_thread_entry.c | pmod2_run_thread_entry.c

```c
#include "pmod1_run_thread.h"
#include "pmod_configure_thread_entry.h"   // event flag
#include <pca9535/pca9535.h>

extern PMOD_BUS_TYPE_t          pmod_bus_type_cfg[PMOD_PORT_NUM];

/* PMOD1 Run Thread entry function */
void pmod1_run_thread_entry(void)
{
    ULONG event_flags;
    uint8_t writedata[8] = {0x33, 0x4A, 0x33, 0xF1, 0x8F, 0x41, 0xCB, 0x99}; // write data

    tx_event_flags_get(&g_config_done_flags, IOEXP_DONE_EVENT_FLAG, TX_AND, &event_flags, TX_WAIT_FOREVER);  // Don't clear it. Leave it


     while (true)
     {
         for (int i=0; i<8; i++) {
             tx_mutex_get(&g_gpio_lock_mutex, TX_WAIT_FOREVER);
             write_pmode_gpio_typeI_byte_port (1, writedata[i],pmod_bus_type_cfg[PMOD1_PORT]);
             tx_mutex_put(&g_gpio_lock_mutex);
             tx_thread_sleep(10);
         }
     }

}
```

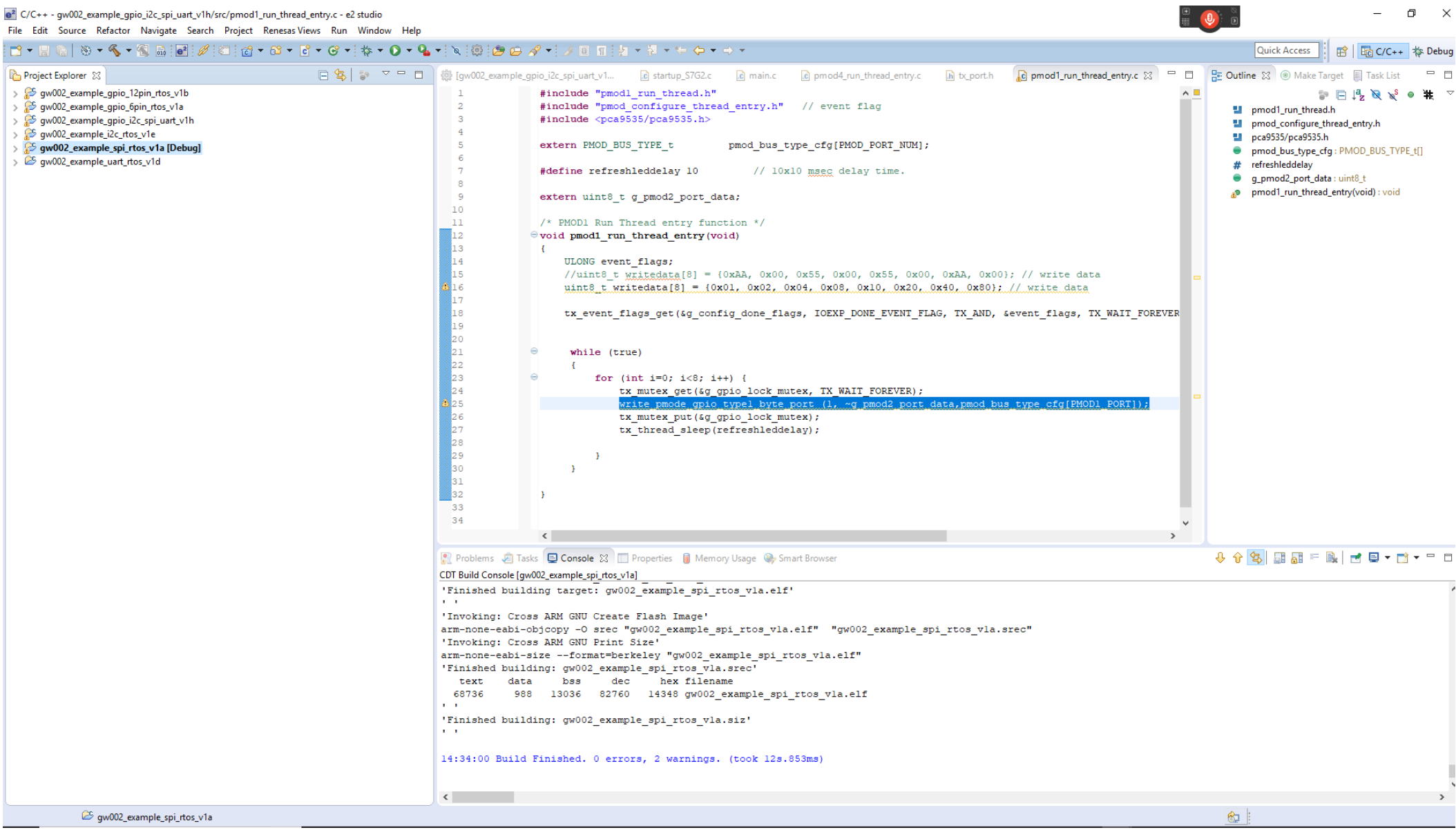
PMOD 2 Thread Entry

pmod2_run_th... | [gw002_exam... | [gw002_exam... | pmod_configu... | pmod1_run_th... | pmod2_run_th... | bmc150_thre... | startup_S7G2.c | main.c

```
7
8   int count = 0;
9
10  /* PMOD2 Run Thread entry function */
11  void pmod2_run_thread_entry(void)
12  {
13      ULONG event_flags;
14      char buf[20];
15      ssp_err_t err;
16
17
18      tx_event_flags_get(&g_config_done_flags, IOEXP_DONE_EVENT_FLAG, TX_AND, &event_flags, TX_WAIT_FOREVER);  // Don't clear it. Leave
19
20      //read acceleration
21      err = g_sf_spi_device0.p_api->open(g_sf_spi_device0.p_ctrl, g_sf_spi_device0.p_cfg);
22      if (err)
23          g_ioport.p_api->pinWrite(LEDREDPIN, true);
24
25
26
27       while (1)
28       {
29           // read xyz value
30           buf[0] = (char)(0x80 | 0x02);
31           //buf[0] = (char)(0x80 | 0x00);
32           err = g_sf_spi_device0.p_api->writeRead(g_sf_spi_device0.p_ctrl, buf, &buf[7], 7, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREVER);
33           if (err)
34               g_ioport.p_api->pinWrite(LEDREDPIN, true);
35
36           //read chip id
37           buf[0] = (char)(0x80 | 0x00);
38           err = g_sf_spi_device0.p_api->writeRead(g_sf_spi_device0.p_ctrl, buf, &buf[7], 2, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREVER);
39           if (err)
40               g_ioport.p_api->pinWrite(LEDREDPIN, true);
41
42           //read temperature
43           buf[0] = (char)(0x80 | 0x08);
44           err = g_sf_spi_device0.p_api->writeRead(g_sf_spi_device0.p_ctrl, buf, &buf[7], 2, SPI_BIT_WIDTH_8_BITS, TX_WAIT_FOREVER);
45           if (err)
46               g_ioport.p_api->pinWrite(LEDREDPIN, true);
47
48           tx_thread_sleep (1);
49           count++;
50       }
51
52  }
```

Get sensor's data and device ID and temperature reading

Successful Build



Correct Device ID read