MSP430 I2C Firmware

The firmware must provide the following features

- I/O Expander PCA9698-like API exposed on I2C (Stem)
- Event driven runtime using interrupts
- Reset/Init setting up the firmware runtime
- Docker build container Debian based that runs the build script
- I can modify and recompile the init/configuration with a common code editor
- Written in C, C++ or Rust

You will create a GitHub repo for building and flashing firmware on a MSP430FR2476 evaluation board. The build will output a binary image and be able to also flash the development board via USB. The build is made to run from a shell on Unix.

Init function can configure

- The USI pins used for I2C port to listen on
- The I2C address to respond to
- Initial register values
- Register callback for reading custom register
- Pins for Stem 1-Wire Messaging

The firmware will ultimately be written in Rust using msp430_rt, with the initial version potentially being written in C or C++.

- MSP430 timer sample code
- Voltage monitor sample code
- ADC wake and transmit on threshold
- Alternative I2C lib
- MSP430 I2C Concepts
- Decoding a Two-Wire SPI-esque Serial Protocol

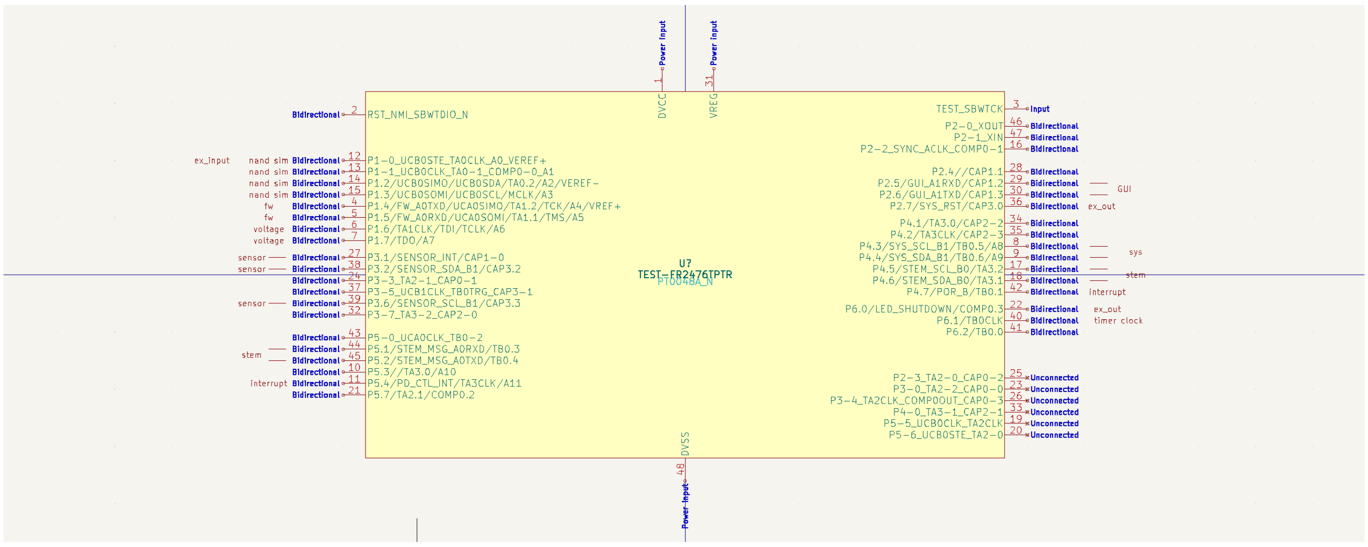
Test Hardware

MSP430FR2476 side

Use a FR2476 LaunchPad

Pinout on the launchpad reserves pins for

- Two voltage measurements
- Input pins that trigger an interrupt message
- Input pins that update an input register
- Output pins driven by an output register
- Sensor I2C that can be proxied to an Expander register
- Stem I2C + 1-Wire negotiation pins


Power input
DVCC
1
Power input
VREG
31
U?
TEST-FR2476TPTR
PT0048A_N
DVSS
48
Power input
RST_NMI_SBWTDIO_N 2 Bidirectional
P1-0_UCB0STE_TA0CLK_A0_VEREF+ 12 Bidirectional nand sim ex_input
P1-1_UCB0CLK_TA0-1_COMP0-0_A1 13 Bidirectional nand sim
P1.2/UCB0SIMO/UCB0SDA/TA0.2/A2/VEREF- 14 Bidirectional nand sim
P1.3/UCB0SOMI/UCB0SCL/MCLK/A3 15 Bidirectional nand sim
P1.4/FW_A0TXD/UCA0SIMO/TA1.2/TCK/A4/VREF+ 4 Bidirectional fw
P1.5/FW_A0RXD/UCA0SOMI/TA1.1/TMS/A5 5 Bidirectional fw
P1.6/TA1CLK/TDI/TCLK/A6 6 Bidirectional voltage
P1.7/TDO/A7 7 Bidirectional voltage
P3.1/SENSOR_INT/CAP1-0 27 Bidirectional sensor
P3.2/SENSOR_SDA_B1/CAP3.2 38 Bidirectional sensor
P3-3_TA2-1_CAP0-1 24 Bidirectional
P3-5_UCB1CLK_TB0TRG_CAP3-1 37 Bidirectional
P3.6/SENSOR_SCL_B1/CAP3.3 39 Bidirectional sensor
P3-7_TA3-2_CAP2-0 32 Bidirectional
P5-0_UCA0CLK_TB0-2 43 Bidirectional
P5.1/STEM_MSG_A0RXD/TB0.3 44 Bidirectional stem
P5.2/STEM_MSG_A0TXD/TB0.4 45 Bidirectional
P5.3//TA3.0/A10 10 Bidirectional
P5.4/PD_CTL_INT/TA3CLK/A11 11 Bidirectional interrupt
P5.7/TA2.1/COMP0.2 21 Bidirectional
TEST_SBWTCK 3 Input
P2-0_XOUT 46 Bidirectional
P2-1_XIN 47 Bidirectional
P2-2_SYNC_ACLK_COMP0-1 16 Bidirectional
P2.4//CAP1.1 28 Bidirectional
P2.5/GUI_A1RXD/CAP1.2 29 Bidirectional GUI
P2.6/GUI_A1TXD/CAP1.3 30 Bidirectional
P2.7/SYS_RST/CAP3.0 36 Bidirectional ex_out
P4.1/TA3.0/CAP2-2 34 Bidirectional
P4.2/TA3CLK/CAP2-3 35 Bidirectional
P4.3/SYS_SCL_B1/TB0.5/A8 8 Bidirectional sys
P4.4/SYS_SDA_B1/TB0.6/A9 9 Bidirectional
P4.5/STEM_SCL_B0/TA3.2 17 Bidirectional stem
P4.6/STEM_SDA_B0/TA3.1 18 Bidirectional
P4.7/POR_B/TB0.1 42 Bidirectional interrupt
P6.0/LED_SHUTDOWN/COMP0.3 22 Bidirectional ex_out
P6.1/TB0CLK 40 Bidirectional timer clock
P6.2/TB0.0 41 Bidirectional
P2-3_TA2-0_CAP0-2 25 Unconnected
P3-0_TA2-2_CAP0-0 23 Unconnected
P3-4_TA2CLK_COMP0OUT_CAP0-3 26 Unconnected
P4-0_TA3-1_CAP2-1 33 Unconnected
P5-5_UCB0CLK_TA2CLK 19 Unconnected
P5-6_UCB0STE_TA2-0 20 Unconnected

Drive it with 3V3 from Raspberry Pi.

A motion sensor MC6470 is connected to SENSOR I2C. Additionally an Ambient light meter APDS-9960 as well.

Connect a pair of UART P5.1 + P5.2 pins combined to STEM_MSG.

Connect P1.6 to the LiPo battery(on nRF52 side) for level measuring.

Connect P1.7 to VBUS(on nRF52 side) for level measuring.

Raspberry Pi setup

Use Python 3 to run PyTests

Connect the MSP430 to pins 3, 5, 21, 24, 26, 29, 31 to command it. As test GPIO pins use RPi pins 7, 11, 27, 28, 36. Connect RPi VCC_FULL to CHARGE pin. Connect a LiPo battery to VSOM pin. Connect pins 21 and 24 together.

Left side	Function	Pin	Pin	Function	Right side
When VSOM fully connected	LIVE_3V3	1	2	VCC_FULL	When VSOM fully connected
I2C3 SDA / STEM_SDA	SDA	3	4	VCC_FULL	When VSOM fully connected
I2C3 SCL / STEM_SCL	SCL	5	6	GND	
		7	8	TxD	UART2 TxD
	GND	9	10	RxD	UART2 RxD
		11	12	SPI1	PD CTL SPI CS
SDIO DAT3 / GPIO2_IO18	SDIO	13	14	SWD	(RPi GND)
SDIO CLK / GPIO2_IO13	SDIO	15	16	SDIO	SDIO CMD / GPIO2_IO14
When any VSOM connected	SLEEP_3V3	17	18	SDIO	SDIO DAT0 / GPIO2_IO15
		19	20	GND	

Left side	Function	Pin	Pin	Function	Right side
STEM MSG RXD4	MSG	21	22	SDIO	SDIO DAT1 / GPIO2_IO16
		23	24	MSG	STEM MSG TXD4
	GND	25	26	SCL	SENSOR SCL
SYS I2C	SYS SDA	27	28	SCL	SYS I2C
SENSOR INT	INT	29	30	(GND)	
SENSOR SDA	SDA	31	32	TxD	UART4 TX
UART4 RX	RxD	33	34		(RPi GND)
PD CTL SPI MISO	SPI1	35	36		
SDIO DAT2 / GPIO2_IO17	SDIO	37	38	SPI1	PD CTL SPI MOSI
SYS_PRG (GND on RPi)		39	40	SPI1	PD CTL SPI CLK

Future revisions will add SWD/SBWT programming pins switched in over the SDIO pins.

nRF52 side (later test setup)

Use an nRF52 dev board(such as Adafruit Feather nRF52 Bluefruit LE, pinout) with USB for easy programming and battery input connector. Pick nRF52832/33/40 for dual I2C support. PSEL allows mapping of GPIO pins to I2C/UART freely. It represents a low power application processor of type M0 / M4 / M7.

Connect to Sensor I2C and Stem I2C as master. This allows both processors to access the sensors and the nRF52 to read and write I2C registers on the MSP430. It also allows nRF52 to simulate sensors in slave mode.

Connect a pair of UART RXD + TXD pins to STEM_MSG.

Connect a LiPo battery

Implement testing software based on Zephyr RTOS. Use device tree to configure I/O Expander communication compatible with PCA9555.

Document how to connect to the nRF52 for programming and console log.

GPIO pins configuration

- TWI: STEM SCL, STEM SDA
- TWI: SENSOR SCL, SENSOR SDA
- UART: STEM MSG (TX), STEM MSG (RX)
-
- Zephyr uart peripheral driver

The Firmware

The firmware image should fit into 8K storage and need no more than 1Kb RAM.

The components of the firmware are,

- I/O Expander
- Message out queue
- Stem Devices tracker
- Voltage meter and monitor
- Stem Message sending
- Advanced interrupt routing
- Persisted default register state
- Persisted init function

Main function / Init

The main function is used to configure effective firmware setup and register RST handler to properly reinstate the setup. It should be possible to run the main multiple times to support runtime updates(replacement) to the main function. The main should be linked at the end of the firmware to enable swapping in a new main function by a simple binary replacement of the firmware image tail. The alternative approach is to store the init function in the Information FRAM and call the stored init from main. It should also be possible to persist the I2C address used in Information FRAM which is then used instead of the default compiled in the firmware.

Most API functions are made to be called during init.

stemI2C(scl_pin, sda_pin, i2c_address)

This starts to listen for incoming I2C requests on the pins with the address. It will provide reading and writing access to defined registers according to the I2C API.

stemPortInterrupt(port_no, int_pin)

It will trigger interrupt pin when input pins change. port_no=1 equals P1. If this has not been defined for a port, no interrupt output is triggered.

stemMSG(rxd_pin[, txd_pin])

This starts to listen for incoming 1-Wire coordination messages other Stem participants. It configures messaging pins. See STEM-MSG.md.

stemSendMSG(message_len, message_bytes)

Messages are queued for sending.

stemInterrupt(port_no, pin_no[], interrupt_id])

Registers an input pin that is used for interrupts. When raised it will generate a message over STEM_MSG. Events 0x10 or 0x20 will be used depending on if an interrupt number is given. Maintain 8 bytes of flag bytes for interrupt pins and a list of pin ID + interrupt IDs.

Masking IP/OP

Since pins may be reserved for other purposes a mask in stored for each bank to filter out changes to pins that are not supposed to be accessed as I/O Expander pins.

It should also be possible to define the default output value set at bootup time.

Voltage Measurement

`vsomVoltage(pin_no, threshold)` `chargeVoltage(pin_no, threshold)`

The firmware must be able to monitor when voltage on a registered pin drops below a specific level. The voltage is LiPo charging compatible, so between 2.8V - 5.0V.

Voltage register for immediate reading of the pin.

Future enhancements

Soft I2C

Some MSP430 chips only have one I2C port. To support additional I2C pins an option must be added to run over GPIO.

sysSoftI2C(scl_pin, sda_pin, i2c_address)

sensorSoftMonitor(scl_pin, sda_pin, int_pin, i2c_address)

exCustomRegister(reg_no, callback/async fetching)

Register callback for reading custom register

You can register a callback that will be triggered when a specific register is read over I2C. The register would be uniquely reserved for this functionality. The callback would then fetch the information and send it as a response.

sensorMonitor(scl_pin, sda_pin, int_pin, i2c_address)

Monitor I2C sensor

Respond to interrupt and read registers on sensor. Do computation of result and update local register. If passing threshold on local register trigger an interrupt or write I2C destination.

voltageThresholdPin(pin)

If set voltage threshold passing will raise this pin.

I2C Expander API

The firmware supports the emulation of a PCA9698 expanded to support 8 pin ports. In principle this supports 64 GPIO pins, but due to some being assigned to other functions the maximum with be ~52 pins. In addition the API will also support Voltage metering and I2C sensor proxying.

Special pins needed at a minimum are

- Stem SCL
- Stem SDA
- Stem MSG
- min. 1 interrupt output pins
- MSP RST
- MSP TEST

The MSP430 P1 - P8 ports are mapped directly to I/O Expander I2C ports. For each port a mask must also be set to screen out pins that should be left alone.

Registers Input, Output and Config must directly drive off the native pin registers. The Polarity & Mask Interrupt must be maintained by the firmware as there is no native support.

The open-drain interrupt output is activated when one of the port pins changes state and the pin is configured as an input. The interrupt is deactivated when the input returns to its previous state or the Input Port register is read (see Figure 12). A pin configured as an output cannot cause an interrupt. Since each 8-bit port is read independently, the interrupt caused by Port 0 will not be cleared by a read of Port 1 or the other way around.

The I/O Expander I2C API is enabled by calling `stemI2C(scl_pin, sda_pin, i2c_address)`.

From the PCA9698 Datasheet

7.3 Command Register / Part

Following the successful acknowledgement of the slave address + R/W bit, the bus master will send a byte to the PCA9698, which will be stored in the Command register.

The lowest 6 bits are used as a pointer to determine which register will be accessed. Registers are divided into 2 categories: 5-bank register category, and 1-bank register category.

7.4 Register definitions

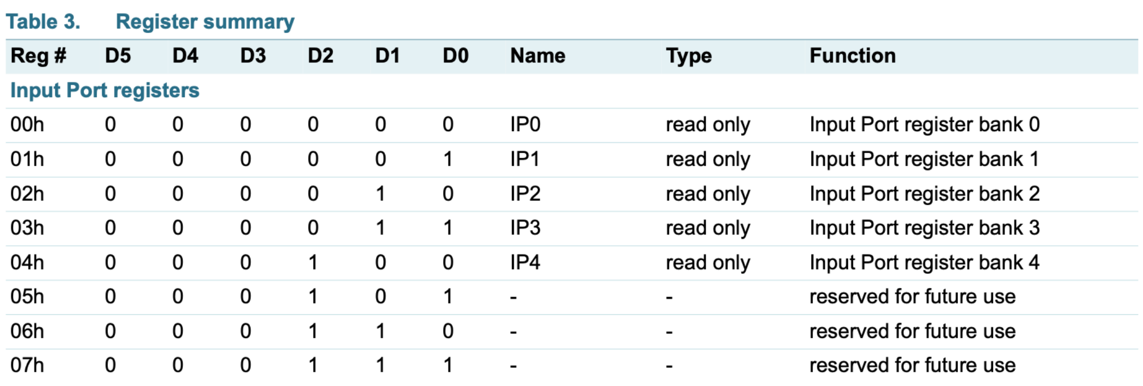
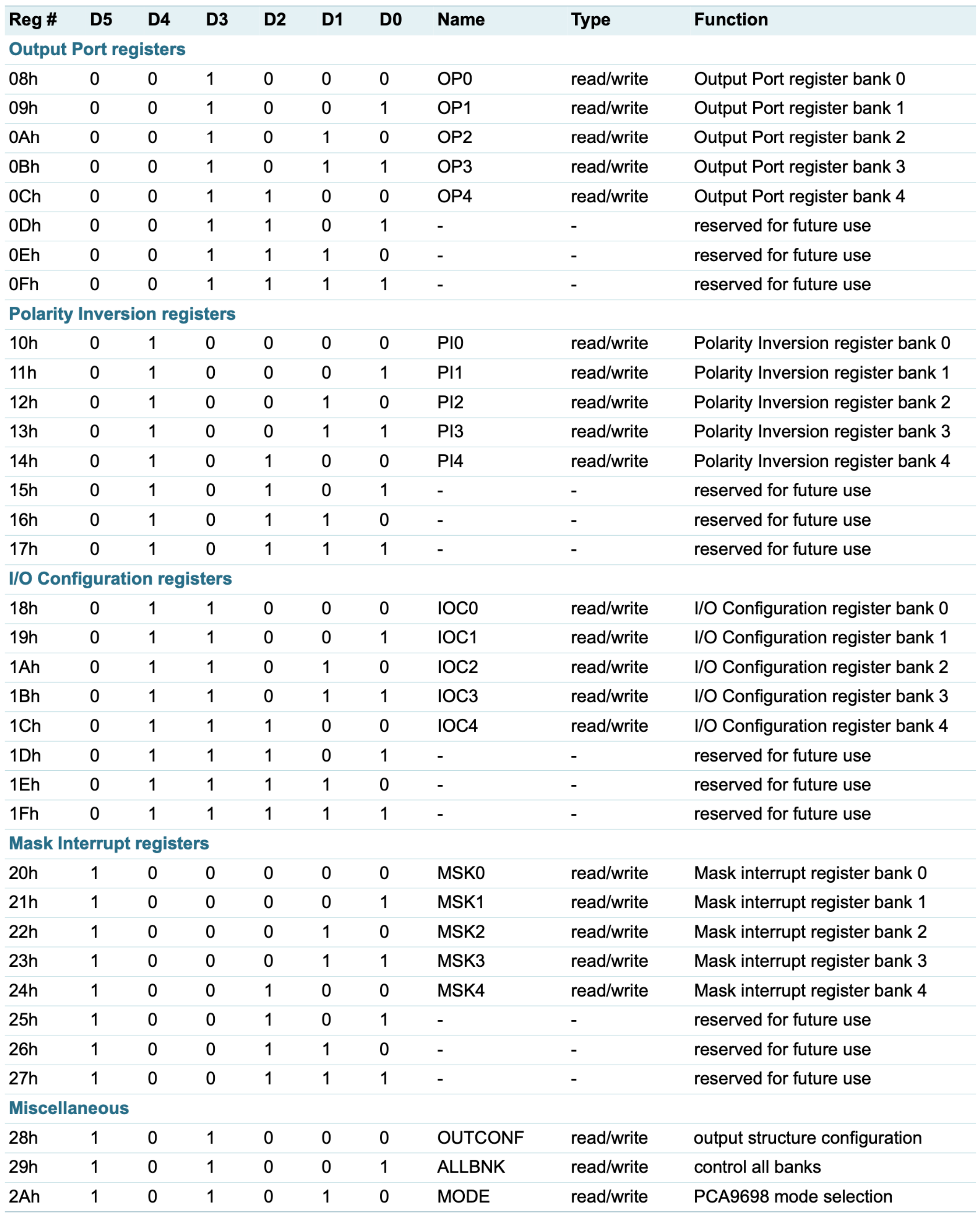



Table 3. Register summary

Reg #	D5	D4	D3	D2	D1	D0	Name	Type	Function
Input Port registers									
00h	0	0	0	0	0	0	IP0	read only	Input Port register bank 0
01h	0	0	0	0	0	1	IP1	read only	Input Port register bank 1
02h	0	0	0	0	1	0	IP2	read only	Input Port register bank 2
03h	0	0	0	0	1	1	IP3	read only	Input Port register bank 3
04h	0	0	0	1	0	0	IP4	read only	Input Port register bank 4
05h	0	0	0	1	0	1	-	-	reserved for future use
06h	0	0	0	1	1	0	-	-	reserved for future use
07h	0	0	0	1	1	1	-	-	reserved for future use

Reg #	D5	D4	D3	D2	D1	D0	Name	Type	Function
Output Port registers									
08h	0	0	1	0	0	0	OP0	read/write	Output Port register bank 0
09h	0	0	1	0	0	1	OP1	read/write	Output Port register bank 1
0Ah	0	0	1	0	1	0	OP2	read/write	Output Port register bank 2
0Bh	0	0	1	0	1	1	OP3	read/write	Output Port register bank 3
0Ch	0	0	1	1	0	0	OP4	read/write	Output Port register bank 4
0Dh	0	0	1	1	0	1	-	-	reserved for future use
0Eh	0	0	1	1	1	0	-	-	reserved for future use
0Fh	0	0	1	1	1	1	-	-	reserved for future use
Polarity Inversion registers									
10h	0	1	0	0	0	0	PI0	read/write	Polarity Inversion register bank 0
11h	0	1	0	0	0	1	PI1	read/write	Polarity Inversion register bank 1
12h	0	1	0	0	1	0	PI2	read/write	Polarity Inversion register bank 2
13h	0	1	0	0	1	1	PI3	read/write	Polarity Inversion register bank 3
14h	0	1	0	1	0	0	PI4	read/write	Polarity Inversion register bank 4
15h	0	1	0	1	0	1	-	-	reserved for future use
16h	0	1	0	1	1	0	-	-	reserved for future use
17h	0	1	0	1	1	1	-	-	reserved for future use
I/O Configuration registers									
18h	0	1	1	0	0	0	IOC0	read/write	I/O Configuration register bank 0
19h	0	1	1	0	0	1	IOC1	read/write	I/O Configuration register bank 1
1Ah	0	1	1	0	1	0	IOC2	read/write	I/O Configuration register bank 2
1Bh	0	1	1	0	1	1	IOC3	read/write	I/O Configuration register bank 3
1Ch	0	1	1	1	0	0	IOC4	read/write	I/O Configuration register bank 4
1Dh	0	1	1	1	0	1	-	-	reserved for future use
1Eh	0	1	1	1	1	0	-	-	reserved for future use
1Fh	0	1	1	1	1	1	-	-	reserved for future use
Mask Interrupt registers									
20h	1	0	0	0	0	0	MSK0	read/write	Mask interrupt register bank 0
21h	1	0	0	0	0	1	MSK1	read/write	Mask interrupt register bank 1
22h	1	0	0	0	1	0	MSK2	read/write	Mask interrupt register bank 2
23h	1	0	0	0	1	1	MSK3	read/write	Mask interrupt register bank 3
24h	1	0	0	1	0	0	MSK4	read/write	Mask interrupt register bank 4
25h	1	0	0	1	0	1	-	-	reserved for future use
26h	1	0	0	1	1	0	-	-	reserved for future use
27h	1	0	0	1	1	1	-	-	reserved for future use
Miscellaneous									
28h	1	0	1	0	0	0	OUTCONF	read/write	output structure configuration
29h	1	0	1	0	0	1	ALLBNK	read/write	control all banks
2Ah	1	0	1	0	1	0	MODE	read/write	PCA9698 mode selection

The standard register allocation has reserved entries for bank 5-8. The firmware will support these.

If the Auto-Increment flag is set (AI = 1), the 3 least significant bits are automatically incremented after a read or write. This allows the user to program and/or read the 8 register banks sequentially.

5-bank register category

- IP – Input registers
- OP – Output registers
- PI – Polarity Inversion registers
- IOC – I/O Configuration registers
- MSK – Mask interrupt registers

1-bank register category

- OUTCONF - Output Structure Configuration register
- ALLBNK - All Bank Control register
- MODE - Mode Selection register

If more than 1 byte of data is written or read, previous data in the same register is overwritten independently of the value of AI.

7.4.4 IOC0 to IOC4 - I/O Configuration registers

These registers configure the direction of the I/O pins.

```
Cx[y] = 0: The corresponding port pin is an output.
Cx[y] = 1: The corresponding port pin is an input.
```

Where ‘x’ refers to the bank number (0 to 4); ‘y’ refers to the bit number (0 to 7).

Pins are configured as input by default. Note that this is the inverse bit values of the MSP430 configuration registers.

7.4.5 MSK0 to MSK4 - Mask interrupt registers

These registers mask the interrupt due to a change in the I/O pins configured as inputs. ‘x’ refers to the bank number (0 to 4); ‘y’ refers to the bit number (0 to 7).

```
Mx[y] = 0: A level change at the I/O will generate an interrupt if IOx_y
defined as input (Cx[y] in IOC register = 1).
Mx[y] = 1: A level change in the input port will not generate an interrupt
if IOx_y defined as input (Cx[y] in IOC register = 1).
```

Voltage and more as Misc. Register

Additional registers are added within the PCA9698 command framework.

2Bh 101011 - Init Code Binary

Allows setting the Init function called on startup/reset. The code will be persisted.

2Ch 101100 - VSOM voltage (read)

Reading this register must return a value reflecting the volate at the P1.6 pin(configurable).

2Dh 101101 - CHARGE voltage (read)

Reading this register must return a value reflecting the volate at the P1.7 pin(configurable).

2Eh 101110 - VSOM voltage threshold (16 bit write)

A message is sent if the threshold is passed. Up to 3 x 16 bit thresholds can be set. LSB/MSB to be defined.

2Fh 101111 - free

It could offer a way to set baud rate from master

Extended Registers

30h 110000 -

3Fh 111111 -

Custom Registers

- Custom registers

Linux Device Driver Support

The firmware will be compatible with PCA9698 40-bit I/O Expander. This should make it simple to configure a Linux Host processor by adjusting the Device Tree bindings and using standard device drivers.

- Linux GPIO driver: PCA95[357]x, PCA9698, TCA64xx, and MAX7310 I/O ports
- PCA9698 IO expander - Interrupt support with pca953x linux driver
- U-Boot Driver for NXP's pca9698 40 bit I2C gpio expander
- RPi overlay comment

1-wire messaging (STEM INT/MSG)

An UART RXD capable pin from each MCU is used message between MCUs on the Stem. The pins are all connected together for low bandwidth message exchange. An UART TXD capable pin can be added to get hardware support for transmitting. If no TXD pin with hardware support is specified, the RXD pin is used to transmit by using GPIO mode.

Each MCU is required to listen in when they are not sending to track who is the master, and other participants. Any MCU can send a request to be a master identifying itself using its I2C address. MCUs can also send messages to notify master of register changes.

- Hackaday 1-Wire
- onewire-over-uart
- Maxim 1-wire

- CCL & USART & 1-WIRE & MCU communication
- UART Intro
- Common 1-Wire review
- Issue in "Address-Bit Multiprocessor Format" UART

Messages are null-terminated byte strings.

Sharing of Message Bus

The main purpose of the bus is for I2C slaves to notify the master when they change state. The secondary purpose is switching the master MCU. The primary coordination needs to be between multiple slave MCUs wanting to send notifications.

If an established mechanism can be used great, otherwise one will have to be devised. The MSP430 supports Idle-Line Multiprocessor Format, perhaps that can be used or extended. It has 10+ bits idle stop bit between blocks and first character is an address. The Stem I2C address for the sending device would be used.

To keep multiple senders to clash an MCU will wait for `n` millis before starting transmission based on its I2C address. This could be `addr % 16 * 2`. During this time it will check to see if the line goes high or if there is transmission going on. If not it will switch to transmitting which will drive the line high.

The master could be given the role of sending a special message when the bus will be idle and a new MCU can send. It would have to save the slaves work in monitoring the bus.

numbers

For port input registers the lower 3 bits reflect the port number. port_no=1 equals P1.

Port number is 3 bits.

Pin index is 3 bits.

Interrupt ID is 7 bits.

Voltage threshold index can be 0 to 2 (2 bits).

Message Types

Idle Hint Message

`| address | 0x1 | 0x0 |`

New Master Message

`| address | 0x2 | 0x0 |`

Input Port Changed Message

`| address | 0xA0 + port number | port value | 0x0 |`

Register Changed Message (reserved, not implemented)

`| address | 0xA0 + register | register value | 0x0 |`

Interrupt Received Message

`| address | 0x10 + port number | pin value & pin index | 0x0 |`

Interrupt by ID Message

`| address | 0x20 + port number | interrupt ID | 0x0 |`

Sensor Threshold Message (reserved)

`| address | 0x4 | sensor id or address | byte array value | 0x0 |`

VSOM Voltage Threshold Message

`| address | 0x8 + threshold index | 16 bit voltage level | 0x0 |`

CHARGE Voltage Threshold Message

`| address | 0xA + threshold index | 16 bit voltage level | 0x0 |`

stemMSG(baud_rate, rxd_pin[, txd_pin])

This starts to listen for incoming 1-Wire coordination messages other Stem participants

Message

- from address
- event
- param

0x40. VSOM threshold 0 - 2 0x44. CHARGE threshold 0 - 2 0x80. Input pin changed state pin = 4 - 56, 0b10xxxxxx.

Upwork task

MSP430 PCA9698 compatible I/O Expander, Voltage meter and I2C Proxy

You should already have existing MSP430 LaunchPad and Raspberry Pi boards to work with. I can send you specific boards, but I will need them back at the end of the project. I expect a robust initial implementation with room for further development.

Later enhancements will include voltage monitoring, multi-MCU coordination, event messaging, slave sensors and rewriting in Rust.

Tests are run on the Raspberry Pi. Python would be a good choice. They could be written as Python unit tests that logs progress to the console.

Show past work

- Show past work on tweaking UART logic in MCU firmware
- Show past experience with Device Drivers and Apps on Zephyr RTOS
- Show past experience with MSP430 firmware

- Show past work with implementing I2C slave firmware
- Show past work measuring voltage
- Show past work of low power, async, event/timer driven MCU firmware
- Show past experience with Rust on MCUs

Milestone 1

Implement firmware that supports input and output registers and pins along the lines of the PCA9698 API. Support Device ID I2C call. No support for SMBAlert, GPIO All Call.

Verify the implementation in hardware by running test scripts on Raspberry Pi. How many changes can it handle? Provide firmware source/binary in Git repo. Provide test script and setup in Git repo. Provide build setup in Git repo.

Milestone 2

Support reading voltage levels as VSOM and CHANGE registers.

Trigger interrupt pin when port input changes like PCA9698. Queue port change event/message when input changes. Update test script to saturate interrupt triggering. How many interrupts can it handle?

Milestone 3

STEM MSG is monitored to determine when sending is allowed. Implement sending STEM MSG from the event queue when STEM MSG is free.

Implement Voltage threshold monitor.

Test it with two MSP430 attached. They should compete to send notifications to the master. The Raspberry Pi is the master. How many interrupt notifications can it handle?

Milestone 4

Implement Zephyr RTOS demo application using standard Device Drivers to I/O with the MSP430 using the Stem I2C API.

Implement test script to run on Zephyr nRF52832 or 52840 which saturates the I2C API with read/write calls. Trigger interrupts, input and output GPIO pins on the MSP430.

Implement GitHub Action to run tests on a Raspberry Pi build slave.

Milestone 5

Support the GPIO All Call functionality

Support SMBAlert command.

Support persisted address & register states

Support update the init code using an I2C command