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. This implies that the pin receives input, but generates interrupt events rather than input change events. 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.