**Next Gen Releasing Panel EMEA**

Software Design Specification

Author: M. Cole

Document ID FD-1611101-EXM851MB SDS

Revision 021

Date Thursday, 23 March 2023

Document Format Microsoft Office Professional Plus 2013

Document Type SDS

Confidentiality Notice

This document, and all information contained herein, is owned by Johnson Controls, and must be kept in confidence and use solely for the owner’s specified purpose. This document must not be copied or reproduced, nor divulged in part or whole, without the owner’s prior consent.

Table of Contents

[Revision History 4](#_Toc121326468)

[1. Introduction 5](#_Toc121326469)

[1.1 Scope 5](#_Toc121326470)

[1.2 SIL2 Considerations 5](#_Toc121326471)

[1.3 Safety Functions 5](#_Toc121326472)

[1.4 References 6](#_Toc121326473)

[1.5 Design Philosophy 7](#_Toc121326474)

[2. Processor and Development Environment 8](#_Toc121326475)

[3. Software Domains 9](#_Toc121326476)

[3.1 Application Software Layers 10](#_Toc121326477)

[4. Boot Loader and Configuration 12](#_Toc121326478)

[4.1 Memory Map 14](#_Toc121326479)

[5. Scheduler 16](#_Toc121326480)

[6.1 GPIO Module 17](#_Toc121326481)

[6.2 Multiplexer Module 18](#_Toc121326482)

[6.3 ADC Module 19](#_Toc121326483)

[6.4 Monitored Inputs Module 20](#_Toc121326484)

[6.5 Hardware Comms Module 21](#_Toc121326485)

[6.6 User Interface Module 22](#_Toc121326486)

[Switches and buttons 24](#_Toc121326487)

[Dip Switch Tests 25](#_Toc121326488)

[6.7 Upper User Interface 26](#_Toc121326489)

[6.5 SLU Interface Module 27](https://mysite.jci.com/personal/jmojoa3_jci_com/Documents/FD-1611101_EXM851MB_SDS_019.docx#_Toc121326490)

[6.8 Extinguishing Application 28](#_Toc121326491)

[6.9 Releasing State Machine 29](#_Toc121326492)

[6.10 Loop Driver 30](#_Toc121326493)

[ASIC with processor control 30](#_Toc121326494)

[ASIC only 30](#_Toc121326495)

[Software simulated interface 30](#_Toc121326496)

[EXM851 Signals currently to be supported by Panel 32](#_Toc121326497)

[(Reference: EXM851 Loop API Requirements Rev 0.2) 32](#_Toc121326498)

[Interface A 32](#_Toc121326499)

[Interface B 32](#_Toc121326500)

[Interface C 32](#_Toc121326501)

[Interface D 33](#_Toc121326502)

[6.11 MX Protocol 33](#_Toc121326503)

[6.12 Loop Interface 34](#_Toc121326504)

[Loop Processor 34](#_Toc121326505)

[Loop Interface ASIC 35](#_Toc121326506)

[Loop Protocol 35](#_Toc121326507)

[6.13 Serial Communications 37](#_Toc121326508)

[USART Driver State Machines 38](#_Toc121326509)

[USART Configuration 39](#_Toc121326510)

[GPIO Configuration 39](#_Toc121326511)

[Flexcom configuration 39](#_Toc121326512)

[Fractional Rate Generator 40](#_Toc121326513)

[USART FIFO 41](#_Toc121326514)

[Stepper motor UART 42](#_Toc121326515)

[USART5\_ParseRxBuffer 43](#_Toc121326516)

[6.14 I2C Driver 44](#_Toc121326517)

[Initialising the I2C port 46](#_Toc121326518)

[Configuring 46](#_Toc121326519)

[IDLE State 46](#_Toc121326520)

[Transmitting State 46](#_Toc121326521)

[Waiting for IDLE State 46](#_Toc121326522)

[6.15 Timers/Counters 46](#_Toc121326523)

[MAINCLKSELA 47](#_Toc121326524)

[MAINCLKSELB 47](#_Toc121326525)

[AHBCLKCTRL (0,1,2) 47](#_Toc121326526)

[Capture and Match 48](#_Toc121326527)

[Using Timers 49](#_Toc121326528)

[System Tick 50](#_Toc121326529)

[6.16 Stepper Motor Module 51](#_Toc121326530)

[6.17 RS485 Comms 53](#_Toc121326531)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev** | **Date** | **By** | **Description** |
| 001 | 05 March 2020 | M. Cole | Initial draft |
| 002 | 11 March 2020 | M. Cole | Review Meeting 1 |
| 003 | 07 April 2020 | M. Cole | Added Timers section |
| 004 | 08 April 2020 | M. Cole | Serial Comms, MX protocol |
| 005 | 09 April 2020 | M. Cole | SLU, watchdog and Boot, Configuring Stepper Motor |
| 006 | 17 April 2020 | M. Cole | ASIC interface investigation and comments. Serial protocol for loop interface.  Added timer API descriptions. |
| 008 | 24 November 2020 | M. Cole | I2C buffering lower UI, System tick functionality |
| 009 | 7 December 2020 | M. Cole | Added UART diagram |
| 010 | 11 December 2020 | M. Cole | Added UI and HW layer diagram |
| 011 | 15 December 2020 | M. Cole | Added DIP switch tests  Added scheduler diagram |
| 012 | 23 December 2020 | M. Cole | Edited and moved Application layer diagram |
| 013 | 09 March 2021 | M. Cole | Add Loop processor schematics  Add Stepper UART5 parser |
| 014 | 12 March 2021 | M. Cole | Add MISRA references |
| 015 | 16 March 2021 | G. Wentworth | Add SIL considerations |
| 016 | 30 March 2021 | M. Cole | - Added Releasing State Machine and state transition diagram  - Updated the software architecture layering diagram |
| 017 | 7 May 2021 | M. Cole | - Update the software architecture layering diagram |
| 018 | 17 May 2021 | M. Cole | - Update software architecture to include new UI objects and events. |
| 019 | 20 July 2021 | M. Cole | Added daughter board to software architecture. |
| 020 | 07 December 2022 | J. Monteiro | Updated ADC and Monitored Inputs sections  Added HW comms section |

# Introduction

# Scope

This document gives a high-level overview of the proposed software solution of the EXM851MB Next Generation Extinguishing Module. It covers the embedded solution to control the digital and software, electronic hardware platform and product requirements presented within the following documents:

*It should be noted that this is not a final version of the SDS and it is expected that it will be updated and maintained in a controlled manner throughout the duration of the project.*

# SIL2 Considerations

SIL2 has been considered for this design and the necessary processes are in place to comply with the SIL2 procedures and requirements defined by IEC 61508.

# Safety Functions

The EXM851 Extinguishing Module is used to receive state of devices and prepare responses with sounder and extinguishing functions within an IEC61508 safety capable system. The product only provides specific safety functions within the system as described in the [SRS]; other functions cannot be allocated to them.

The following functions detailed in this specification are considered to be safety functions:

* [sf-01] A remote Manual Trigger switch (supervised input) (OR)
* [sf-01] Standard Interface Trigger Signal (supervised input via VdS interface)
* [sf-02] Sounder output (supervised output)

The module startup including the processor and hardware initialisation are considered to be part of the Initialise module function.

The interrupt handler used to activate Sounder and hardware monitor data is also considered part of these safety functions.

All other functions are considered outside the scope of the IEC61508 assessment. However, they shall be designed so that they can co-exist with the safety functions without compromising the operation of the safety functions.

The safety functions are marked with the numbered tag ‘[sf-nn]’ (where nn is a number) in this document to provide traceability through the development process.

# 1.4 References

**Internal Document References**

1. EXM851\_HRS\_Issue 0A\_v\_06 Author: Alex Hiley
2. EXM851MB-i0A\_-1\_Interconnection\_sheet.SchDoc Author: Unknown
3. FD-1611101 Nex Gen Releasing DSRS-18\_06.docx Authors: G. Wentworth, M. Whaley
4. FD-1611101 Nex Gen Releasing and panel PF Author: P. Walsh
5. NGExMLP Design Technical Report Rev3 Author: Daniel Matthews
6. New Gen releasing solution iss1-3-2.00tx Authors: Lisa Paine, P. Walsh
7. Technical\_Proposal\_Extinguishing Module.pptx Author: Prakash Dhadage
8. \_cppstan (‘C’ Coding Standard) Author: Mike Mayhew

**Third Party Document References**

* 1. DDI0439B\_cortex\_m4\_r0p0\_trm Rev r0p0
  2. LPC546XX Data Sheet Rev 2.6
  3. NXP\_LPC546xx\_User\_Manual Rev 1.3
  4. NXP\_OM13092\_Board\_UM11035 Rev 2.1
  5. PCA9535\_PCA9535C I2C-bus GPIO Port Rev 6
  6. MISRA-C:2004 parts 1.2, 6.1, 6.2, 6.3, 6.4, 6.5, 9.1, 9.2, 9.3, 14.1, 14.2, 14.3, 14.4, 14.5, 14.6, 14.7, 14.8. 14.9, 14.10, 20.4, 21.1 inspired by 61508 part 3

**Standards**

1. BS EN12094-1

# Design Philosophy

The following considerations should be made when designing the software for the EXM815MB:

**Reuse**- Where possible, existing proven software solutions should be re-used from other Johnson Controls products in order to speed up development

**Testing** – This software design must make consideration for testing via third parties. If possible, a test interface should be provided to allow external automated test.

**Portability and maintainability**– As much of this software as possible should be platform independent. This includes anticipating modifications to the hardware and digital platforms as well as microprocessor itself. This can take into consideration hardware design changes, product requirements and product obsolescence issues that may arise during the development or in the future. Only the lower software layers should be affected when platform change occurs. Only the application layers should change when product requirements changes occur.

**Object Oriented** – In order to meet the portability/maintainability requirement above, as far as possible, a clear domain/class analysis and solution should be provided. Coupling and dependencies between modules should be as loose as possible. This may incur a cost in program space and memory but the pay-off will be in understandable code structure and maintainability.

**Software layering** – In order to meet the portability/maintainability requirement above the software will be layered in accordance with the OSI 7-layer model as per <https://en.wikipedia.org/wiki/OSI_model>

**Reliability** – The software will adhere to MSRA standards as per <https://en.wikipedia.org/wiki/MISRA_C>

**Unit Testing** – A unit test platform will be provided to pre-empt bugs and issues that occur throughout the development.

**Regular demonstrable deliverables** – An iterative approach will be taken that provides demonstrable pieces of functionality fast and often. These may not necessarily constitute customer requirements but nevertheless should illustrate project progress. This will assist in the testing process, planning process and help with rolling back functions and features should rework be required.

**Ease of implementation –** The software will be designed in such a way that engineers can work in isolation on specific features/functions with minimum interdependency. This relates to the ‘Portability ‘and ‘Object Oriented’ criteria above.

**Clearly defined interfaces** – inter-module and inter-domain communication will be defined up-front in order to facilitate the ‘Ease of implementation’ criteria above.

**Proprietary**– To avoid copyright problems, it is important that only software designed and written in-house will be used. This means third party examples and application solutions should not be used.

# 2. Processor and Development Environment

**Processor:** LPC54606J256 ARM Cortex M4 (100 pin)

Relevant specifications:

* 256 KB on-chip flash
* 128 KB SRAM
* 2 x 32 pin configurable GPIO ports (256 pin has 6 ports)
* 12 x ADC converters (12 bit)
* 10 x Configurable Flexcomm register banks regions for mapping GPIO to I2C, USART, SPI, I2S, ADC ports
* DMA Controller with 30 channels
* 5 x 32-bit timers/counters supporting input capture, output compare and PWM
* Multi-rate 24-bit timer
* Windowed watchdog timer
* Repetitive interrupt timer (RIT)
* 12MHz internal oscillator (provides 12,48,96 MHz outputs
* External input clock for 25 MHz
* Crystal oscillator up to 25 MHz
* Ultra-low power mode

**IDE:** MCUXpresso v11.0.1 IDE

**Debugger:** Segger J-Link plus JTAG Emulator

# 3. Software Domains

The following diagram is an overview of the software domains that need to be considered when performing the design. It is not exhaustive and the relationships are preliminary. Full breakdowns of each domain will follow in subsequent sections of this document.

**MX Loop Protocol interface**

- communicate with NOSEX device for transmission and reception of protocol messages to the panel.

**Timer subsystem**

- Timer interrupt processing and call-backs (coarse)

Generate subsystem events for:

- Valve timers

- Bell timers

- Release timers

- UI blocked timer

- Hold-off timer

- State machine transition guard timers

- Watchdog

- Low priority system status updates (outside of main loop)

**UI subsystem**

Interface to CIE and ECD UI as well as on-board switches and optical outputs

- Driven by system state changes

**Monitored I/O**

Provides interface to fault manager and Core engine

**RS485 interface 1**

- Device addressing

- Packet TX/RX control

- Buffer management

**Panel communications subsystem**

- Panel command processing

- Status reporting

**Core extinguishing engine**

- Process UI inputs & IO inputs

- generate faults (timing, transition errors, and invalid events for current state)

- manage sequence timing

- drive solenoids and outputs

- inform local and remote UI

- send panel state messages

Figure 1. Domain Overview

**ADC interface**

Analogue interrupt processing APIs for client polling, averaging, thresholds

**UART interface**

- RX/TX interrupt processing

- Data handling

- Buffering and access APIs

**Serial Communications**

- Buffers and buffer management

- Buffer access APIs

-Timed interface polling

**Test mode subsystem**

- Command and response processing (test mode protocol command)

- Raw serial output (run-time debug)

- Provides interface to all subsystems generating mock events

- allows automated testing

**SMC96 (Stepper Motor) subsystem**

- Programming

- PIC processor interface

**RS485 interface 2**

- Device addressing

- Packet TX/RX control

- Buffer management

**I2C interface**

PZ8X, Status LEDs, Dip switches

**Fault manager**

Monitor I/O subsystem for threshold breaches relay timeouts fault inputs. Inform UI and extinguishing engine send messages to panel

**Fault relay**

- monitored for sc/oc

- trigger when de-energized (fault)

**Solenoids**

Driven by extinguishing engine

- monitored by fault manager for o/c and s/c

**GPIO interface**

(On board LEDs and switches) Also drives solenoids and auxiliary outputs

**Supervised inputs**

- Manual Release

- Alarm pressure

- Hold off

- Valve suppressed

- Preaction

- monitored by fault manager for o/c and s/c

**Auxiliary outputs**

- Sounders & beacons

- monitored by fault manager for o/c and s/c

# 3.1 Application Software Layers

**Isolates Manager**

This module contains the state of all the isolates in the system. It latches them and provides interlocking to the Releasing engine. It reports them to the panel via the loop. It calls the UI to display the locally or via SLU

**Fault Manager**

This module contains the state of all the faults in the system. It latches them and provides interlocking to the Releasing engine. It reports them to the panel via the loop. It calls the UI to display the locally or via SLU

**Releasing Engine**

This module contains the state machine which controls the releasing process. It calls the MON (monitored inputs) domain for information about the valves, sounders, relays and faults. It will access the sounders, relays and valves via moninputs to open, close, turn on and off these peripherals. It also access the UI by calling the status module to turn on and off the buzzer and drive the LEDs. In addition to this it monitors button presses from the UI to trigger release, hold and abort and reset functionality.

**DRV drvcomms.c**

**(Monitors all comms)**

**LPC**

**lpccomms.c**

**SLU**

**slucomms.c**

**MB-UI**

**mbuicomms.c**

**DB-UI**

**dbuicomms.c**

**MB-HW**

**mbhwcomms.c**

**DB-HW**

**dbhwcomms.c**

**STP**

**stpcomms.c**

**COMMS Domain**

**MON**

**dbmonitor.c**

**IND**

**OUT**

**dbocoutputs.c**

**PWR**

**VLV**

**SDR**

**RLY**

**DB HW Domain**

(Daughter board)

**MON**

**monitor.c**

**OUT**

**ocoutputs.c**

**MB HW Domain**

(Motherboard)

Common

**IND**

**PWR**

**VLV**

**SDR**

**RLY**

Common

**MBADC**

APIs for reading ADC channels

**DBADC**

APIs for reading ADC channels

**MBMX**

Selects inputs via mux devices

**DBMX**

Selects inputs via mux devices

**GPIO**

APIs to read and write to pins

**DRIVER Domain**

**UI Events**

**UI Objects**

BtnEv

SwtEv

LedEv

BzrEv

InfEv

**SWT**

**LED**

**BTN**

**INF**

**BZR**

**UI Domain**

**UI Manager**

**UART3**

Stepper Motor

**UART2**

Status Lamps Unit

Common

**UART1**

Loop Comms

**I2C 4**

APIs for reading and writing to the UI I2C chips

Common

**I2C 1**

APIs for reading and writing to the MB I2C chips

**power.c,h**

This module is called by **moniputs.c**.

It operates in the same way as valves.c,h. It contains information about the 24v and 26v power rails

**hwcomms.c, hwcomms.h (refer uicomms.c,h)**

This module is called by the scheduler.

It triggers ADC reads of all monitored inputs.

It holds those values in raw format.

It provides functions for the higher layers to access them.

For example, PWR (power.c) will periodically poll its respective

ADC read. Likewise relay.c, sounder.c etc. Also, it provides APIs to drive the hardware. This can either be directly (in the case of sounders and relays) or indirectly (in the case of uart driver talking to the stepper motor valve controller)

**relays.c,h**

This module is called by **moniputs.c**.

It operates in the same way as valves.c,h

**valves.c,h**

This module is called by **moniputs.c**.

It will call hwcomms.c for the status of the valve held according to the last ADC read.It will have the followig states: open, closed, opening, closing, fault. It will condition the raw ADC values and compare them with calibrated values to determine these states. It will also provide APIs to drive the valves.

**moninputs.c,h**

This module is called by the scheduler. It calls power.c, relay.c, sounders.c, valves.c for information on each of their states. This information is made available for use by the RLS domain to drive the releasing state machine.

**sounders.c,h**

This module is called by **moniputs.c**.

It operates in the same way as valves.c,h

**led.c,h**

This module is called by status.c (via the scheduler). It provides an interface to switch on or off specific LEDs and stores their current state. It does this using uicomms.c

**button.c,h**

This module is called by status.c (via the scheduler). It provides an interface to read button states from the uicomms.c. It stores these and provides APIs to the RLS domain for use in changing release state (abour, release, hold etc.)

**uicomms.c,h**

This module is called by the scheduler.

It triggers I2C reads of all the UI incputs. It includes LEDs, switches and buttons. It holds those values and provides functions for the higher layers to access them.

For example, LED (led.c) will drive specific LEDs on or off depending on status.c.

Likewise button.c will periodically check the last polled button state for buttons like hold, reset, abort, release etc.

**buzzer.c,h**

This module is called by status.c (via the scheduler). It provides an interface to drive the buzzer, read the buzzer state and keep local information about the state of the buzzer.

**status.c,h**

This module is called by the scheduler. It updates the status of the UI via buzzer, buttons, leds and dip switches. It provides APIs for release.c to read and drive these. It accesses this information via buzzer.c, button.c, led.c, dipsw.c.

# 4. Boot Loader and Configuration

On power up the following DIP switches will be read. There will be three boot modes.

* Firmware upgrade
* Configuration <We cannot pass configuration from the loop to the processor>
* Normal Operation
* Engineering mode

Each mode is mutually exclusive:

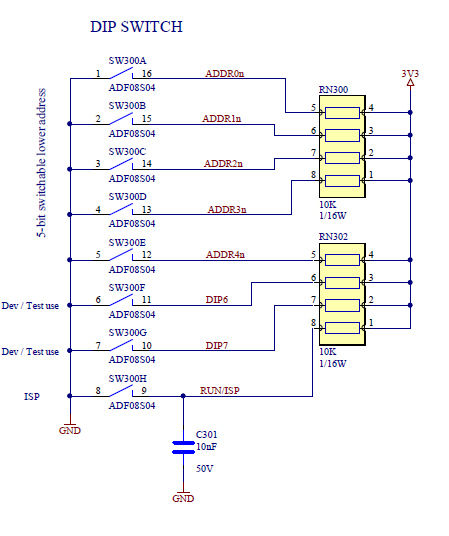
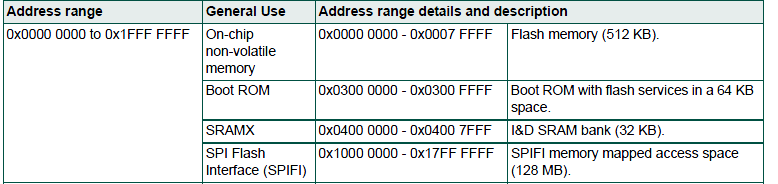


Figure 2. Dip switches

The following Non-volatile memory is available for storing Boot-loader and configuration:



A Flash Driver module will be created to read write access to flash memory

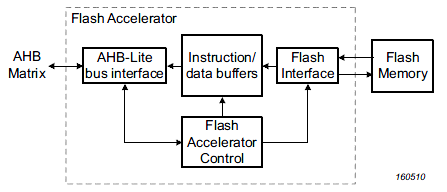
Flash

Fault history logging

Boot loader module

Configuration module

Flash driver module



On the assertion of any reset source (ARM software reset, POR, BOD reset, External

reset, and Watchdog reset), the following processes are initiated:

1. The FRO is enabled or starts up if not running.

2. The flash wake-up starts. This takes approximately 250 uS or less.

3. The boot code in the ROM starts. The boot code performs the boot tasks and may

jump to the flash.

When the internal Reset is removed, the processor begins executing at address 0, which

is initially the Reset vector mapped from the boot block.

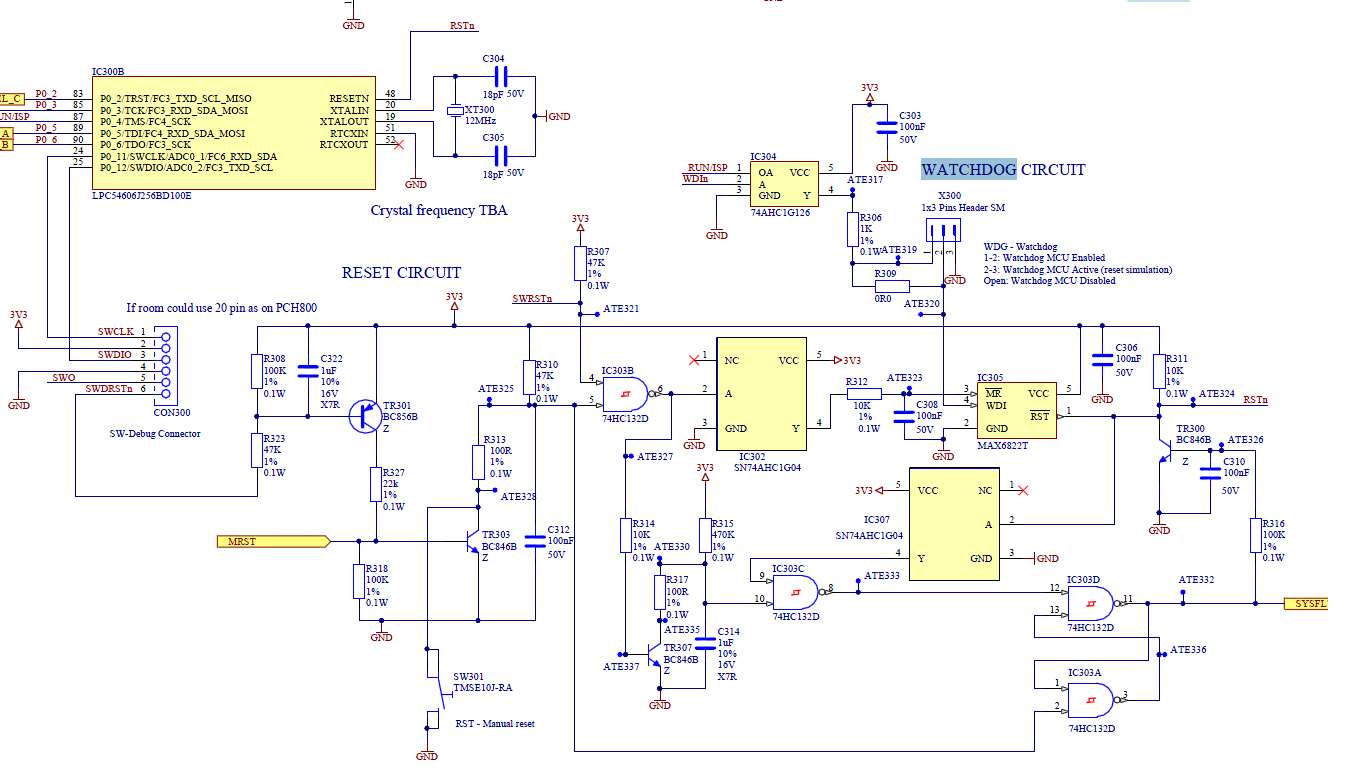
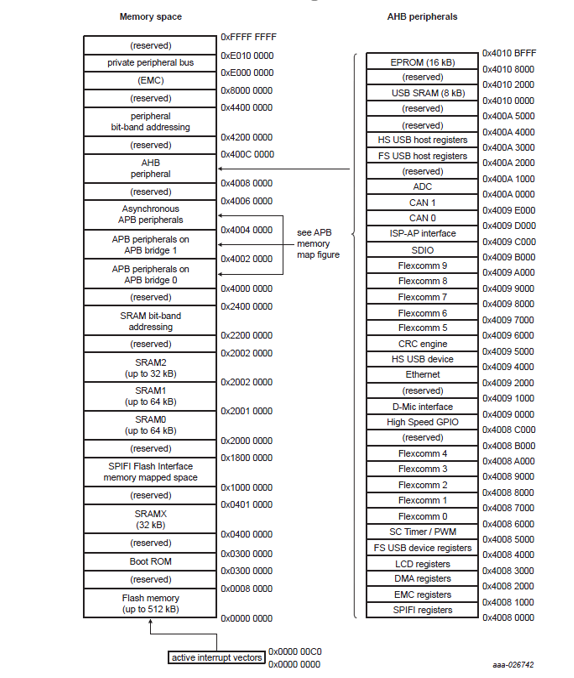
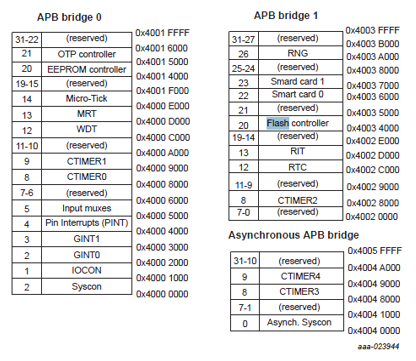


Figure 3. Watchdog circuit

# 4.1 Memory Map



Task/Callback table

# 5. Scheduler

Task Control shall be organised by round robin scheduler and timer interrupts.

Communications interfaces will be performed by interrupt. DMA will be used to take processing time out of the main loop. This applies particularly to RS485 Communications with the Loop Driver. UI related RS485 Communications do not need DMA control as much. Nor does the USART interface.

Responding to faults is the next highest priority. Monitoring inputs via ADC should be scheduled by high resolution timer and can be serviced within the main loop.

Reading and driving of GPIO via I2C and Multiplexer I/O are low priority and can be scheduled outside of the Main Loop via low resolution timer. These relate to switches and UI tasks.

The main loop will poll for state changes in each domain. Very little house keeping processing should be performed by the Main Loop as this will defer high priority task response times. I.e. ideally, when no state changes occur in the system, the main loop should be doing nothing except kicking the watch dog.

Each thread/task spawned by the Main loop should be as short as possible. This can be monitored by ensuring a very tight watchdog constraint.

A system tick counter will be used to schedule tasks at a rate appropriate to their priority.

6. Modules

**SCH\_ProcessSystemTick**

Called from system tick interrupt

Decrement counters for each function and expire when 0.

Run any functions that are marked as ‘interrupt’. Mark others for main loop.

**scheduler.c (system module)**

**SCH\_LoopTaskHandler**

- Called from main loop

Search table for expired ‘main loop’ functions and execute one by one

Task/Callback table

Function 4

Function 5

Function 6

Function 1

Function 2

Function 3

# 6.1 GPIO Module

The LPC54606J256 has 6 x 32 pin ports. We are using ports 0 and 1. A single pin can be configured as one of three types. These are digital, I2C and Analogue. Each pin has a subset of eight functions available to it. These include digital I/O, serial I/O, I2C I/O, ADC I/O. The following interface should be provided to configure an individual pin:

Each pin is configured using the following register fields:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Function  IO,CAN,SCL, TIMER,  SCT,SDA,  SCL,CLK,  USB. | Mode  0=inact,  1=pulldwn,  2=pullup,  3=repeater | I2CSlew  0=i2c mode,  1=gpio mode | Invert  0=invert disable,  1=invert input | Digital mode  0=analog,  1=digital | Filter off  0=glitch filter enable,1=glitch filter off  Figure 4. Pin Configuration Register | Slew/I2Cdrive  0=slow slew,1=fast slew/ 0=low drive(4mA), 1=high drive(20mA) | Open drain/I2C Filter off  0=push-pull,1=open drain/ 0=glitch filter enable,  1=glitch filter off |  |

On power up each GPIO client is responsible for initialising their own GPIO pins via the GPIO interface as follows. Pins should not be accessed directly within client modules. There is no initialisation for this module itself as this is a service module only. Note, there are also toggle and mask registers which can be used to change pin state more quickly. (Refer: NXP\_LPC546xx\_User\_Manual Rev 1.3)

**Stepper Motor**

Configure USART Pins

**Loop Comms**

Configure USART Pins

**UI Comms**

Configure USART Pins

**Monitored Inputs**

Configure Digital output pins for MUX select. Configure ADC input pin for reading values

**Valve/Relay Controllers**

Configure digital output pins

**Status indicators**

Configure I2C pins

**Serial Comms**

Configure USART Pins

Additional APIs

Clear port clear register

Clear port set register

Set Port clear register

Set Port set register

Program PIN register

Validate input params Register address

Set Port direction register

Function Name,

Port number,

Pin number,

Pin type,

Function, Number,

Mode,

Invert,

Digi mode,

Filter off,

Slew,

OD,

I2c slew,

I2c drive,

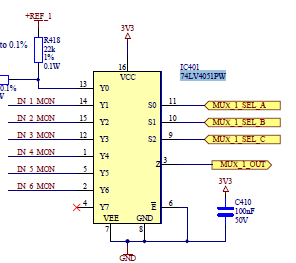
I2c filter off

Figure 5. GPIO module interfaces

# 6.2 Multiplexer Module

The first client of the GPIO module is the multiplexer module. This allows extension of GPIO for accessing the monitored input and switch input modules. However, it should not refer directly to these as it could be used to extend the GPIO for any future purpose.

RISK: We would like confirmation of chip selected. Suggest using Y7 on each chip as an input with different voltage per chip as feedback for confidence.



There are three 74LV4051 8-channel multiplexers.

MUX 1 selects 6 Input monitor lines

MUX 2 and 3 select the following inputs:

* 26+
* Valve 1 out+
* Valve 1 out-
* Valve 1 current
* 24v
* Valve 2 out+
* Valve 2 out-
* Valve 2 current
* Sounder +
* Sounder –
* Relay 1 mon
* Relay 2 mon

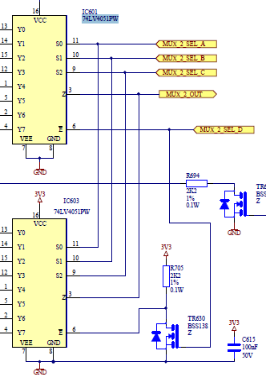


Figure 6. Multiplexer Schematic

pAdcConvSeqAReg->adc\_ConvSeqAReg The interfaces to this module include:

Switch Inputs Module

ADC Module

***Initialise Mux***

Sets the MUX select pins as digital outputs and the

MUX output pins as digital input for MUX1 and

Multiplexer Module

Analogue inputs for MUX 2 and 3. This is done using the

GPIO Module

GPIO module.

Figure 7. Input Modules Hierarchy

***Select Channel***

Takes the MUX number (1, 2, 3) and channel number and selects

The MUX input line to output via the Z pin.

***NOTE: There are twelve analogue channel inputs on the LPC54606J256 but this hardware solution only uses five and multiplexes the source lines. Since this is not enough, a chip selector signal is also used to switch between MUX\_2 and MUX\_3***

# 6.3 ADC Module

The ADC module is a client of the multiplexer module (See figure 5. above). There are two ADC channels dedicated to the MB monitored inputs - ADC0\_3 and ADC0\_4 - and further three dedicated to the DB monitored inputs – ADC0\_5, ADC0\_6 and ADC0\_11. They are configured as Type A pins (See figure 2. Above), using the AdcConvSeqA registers only.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Channels  12 bits represtenting which of the 12 ADCs to use | Trigger  6 bit value specifying trigger method(Output pin, timer, input pin etc. | Trigger polarity  0=neg edge, 1=positive edge | Sync bypass  Enable trigger sync | Start  1 = start ADC conversion  Figure 6. ADC Configuration Register | Burst  Causes continuous reads | Single step  Results in one conversion only | lowprio  allows B sequence to interrupt A |
| Mode  Determines which register to read after conversion (DMA) | seqaena  Enables/disables sequence A |

A read is triggered by enabling the ADC interrupt (See Interrupts Module) and setting the ADC interrupt enable register. The start flag is set and the interrupt will be triggered when the conversion of all active channels is complete. The data will be saved in local registers that will be available to the Monitored Inputs Module. The Monitored Inputs module will run a scheduled task to select the multiplexers and trigger all ADC channel reads.

The sequence is: The Hardware comms module will run a scheduled task to cycle through all MUX channels and trigger ADC reading samples for every input. Monitored Inputs module will run a parallel scheduled task to periodically poll averaged readings for the hardware comms module and qualify the input state according to a pre-defined thresholds table.

The following modules hierarchy illustrates this process.



* Monitored Inputs polls readings from HW comms and processes all inputs values thresholds
* HW comms select MUXs and ADC channels, reads and saves averaged results
* ADC Module deals with low level ADC channel sampling and interrupts.

Monitored Inputs Module



MUX Read Buffer

Hardware comms Module

ADC Read Buffer

ADC Module

Multiplexer Module

GPIO Module

Figure 7. Monitored Input Modules Hierarchy

# 6.4 Monitored Inputs Module

The Monitored Inputs module is an Application Layer module that schedules input reads of multiplexed analogue measurements as per Figure 7 above. Its responsibilities are to schedule timed ADC conversions of each input in turn. It will then process these reads and perform range checks on them.

Configured thresholds will be used to determine state conditions for these inputs depending on their meaning. It is aware of the meaning and location of each configured input and how to talk to the ADC and MUX to request them. Each input will be in one of the following states:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0%-20% | 20%-25% | 25%-45% | 45%-60% | 60%-100% |
| Open Circuit | Normal | Low Short Circuit | Active | Short Circuit |

Figure 8. Threshold levels

The Monitored Inputs module is not considered to be highest priority. Hence, it should not drive any of the extinguishing module behaviour. Instead, it will be checked regularly by the following clients;

Extinguishing Engine - The extinguishing engine will regularly keep check on the monitored inputs in order to make decisions about state changes that it is allowed to perform.

User Interface - Status indicators will be updated by the User Interface module. This is by I2C Comms and RS485 to connected LEDs and UI daughter board. This may include buzzer.

Loop Interface - The Loop driver will report state changes via MX message to the panel

Fault Manager - Faults and hardware issues are monitored and logged by the faults manager.

# 6.5 Hardware Comms Module

This module is responsible for maintaining I/O operations with various peripherals and for a synchronized MUX selection and channel switching while reading all the monitored input ADC data.

The main input reading function is called by the scheduler every 5ms and alternates between two states; ADC triggering and MUX selection, so enough time is given for both to finish their tasks. This function is also responsible to drive the sounder Z test pulses before and after the ADC sampling.

Stop ADC Timer

Trigger ADC read

Drive Z Pulse ON

MUX Selection

ADC Triggering

Drive Z Pulse OFF

Averaged ADC Values

Raw ADC data

(5 channels)

Start ADC Timer

Update Sampling Table

Select MUX and Channel

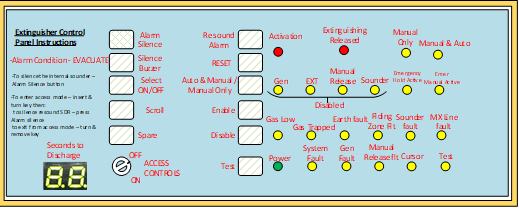
ADC Interrupt

Figure 9. Hardware comms input reading mechanism

# 6.6 User Interface Module

SECONDARY SOLUTION There are two proposals for User Interface. A separate SLU daughter board using

RS485 connection (currently in existence):



PRIMARY SOLUTION: And a simple on board LED and switch array using I2C chips as follows:

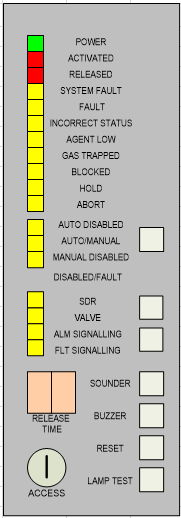
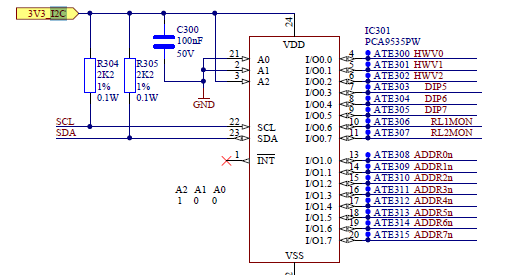
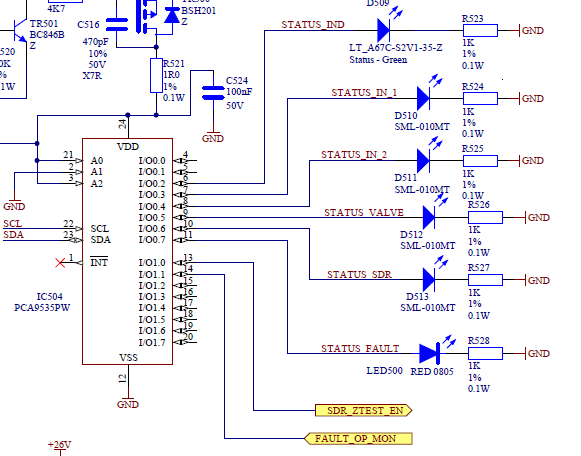


Figure 10. I2C devices on board

Faults will be indicated by number on the 7-segment LED display. An engineering mode will be provided

to scroll through fault codes. No logging will be performed. The UI module acts as a slave and will provide

APIs to drive LEDs directly from the Extinguishing Engine, the Faults module, the Monitored inputs

module.

Extinguishing active

Manual release

Alarm

Valves

Sounders

Relays

Buzzer

Sounder fault

Gas trapped

Gas low

Figure 10. UI Clients

Watchdog

Battery

Power

Earth

LEDs will be refreshed at a similar rate to buttons. They are accessed directly by modules shown in Figure

10 above. NOTE: There is an additional need to add a 6x6 I2C matric interface to select up to 40 LEDs and

buttons.

### Switches and buttons

We will aim for a sample rate of 10-50ms and confirmed status in 100ms. (​Similar to two alarm inputs from detectors) we should design at least two samples for input buttons, for safety.

Dip switches will be used to configure the releasing timer. They can also be used for selecting and

deselecting certain features. An Upper UI layer will be needed to manage these features.

Poll requests will be queued in the I2C driver. The UI will receive responses from the I2C driver via

interrupt and queue them for processing in the main loop (higher priority control via system tick). Care

must be taken to ensure the poll request queue does not overflow. Ideally it should be only depth on ‘1’

but there may be a need to have larger.

The module structure for this is:

Lower UI ‘comms’module (i2ccomms.c)

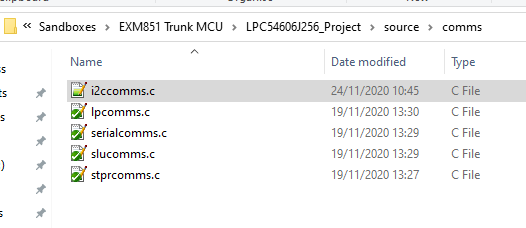
(i2ccomms)

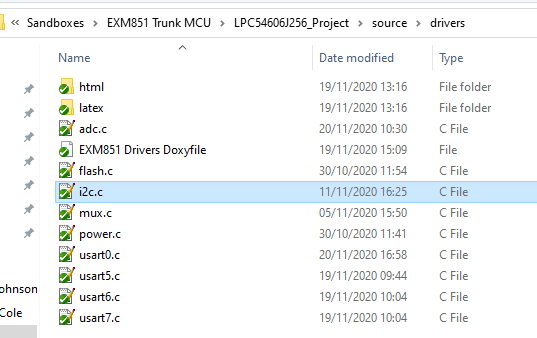


I2C Driver ‘driver ‘module (i2c.c)

PCA9535PW

Figure 11. Lower UI layers

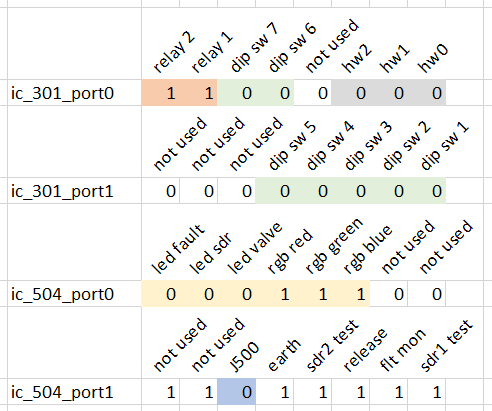




### Dip Switch Tests

**Test 55**

This test allows the reading of IC301 and IC504 input ports



**Test 56**

This test allow to write data to specific i2c addresses and registers Current valid addresses: 37(0x25) and 36(0x24)

Ex: 37,02,24 - Writes 00011000 to port 0 of IC504 - Blue led ON

# 6.7 Upper User Interface

The following classes of UI object will be monitored. As state changes occur, this will be reflected in these objects. These will result in events generated to the Extinguishing application, features enabled/disabled, configurations such as timers modified and system status being interrogated/displayed.

Switches –

* Pre-discharge delay
* Buzzer disable
* Latched Faults
* Instant annual release
* Buzzer pulses at end of delay
* No discharge pressure switch
* Actuator S/C Disable
* Reset inhibit period
* Clear actuators after 1min
* Silence alarm before reset
* Auto/manual
* Configure SLU
* No Low pressure input
* Key

Buttons –

* Silence alarm
* Silence Buzzer
* Select
* Reset
* Re-sound alarm
* Auto mode
* Enable
* Scroll
* Test

LEDs -

7-Segment Display

Buzzer

Sounder

|  |
| --- |
| The following standards related use cases apply to the Extinguishing application: |
| 1. Electrical Automatic Control and Delay Device Software Requirements **(4.3 EN12094-1 Standard)** |
| 1. Compulsory and optional functions **(4.3.2 EN12094-1 Standard)** |
| 1. Transmission of extinguishing signal **(4.5 EN12094-1 Standard)** |
| 1. Activation of Alarm Devices **(4.5 EN12094-1 Standard)** |
| 1. Indication for the supply with power (4.7 **EN12094-1** Standard) |
| 1. Activated Condition **(4.8 EN12094-1 Standard)** |
| 1. Indication of activated condition **(4.9 EN12094-1 Standard)** |
| 1. Released condition **(4.10 EN12094-1 Standard)** |
| 1. Indication of Released Condition (4.11 **EN12094-1** Standard) |
| 1. Resetting of the Activated condition and the Released condition:  **(4.12 EN12094-1 Standard)** |
| 1. Fault warning condition **(4.13 EN12094-1 Standard)** |
| 1. Indication of Fault Warning Condition (4.14 EN12094-1 Standard) |
| 1. Disabled condition **(4.15 EN12094-1 Standard)** |
| 1. Delay of extinguishing signal (Must) **(4.17 EN12094-1 Standard)** |
| 1. System Delay (4.7 VDs2454) |
| 1. Signal representing the flow of extinguishing agent **(4.18 EN12094-1 Standard)** |
| 1. Monitoring of the status of components **(4.19 EN12094-1 Standard)** |
| 1. Emergency hold device **(4.20 EN12094-1 2 Standard)** |
| 1. Control of flooding time (**4.21 EN12094-1 Standard)** |
| 1. Initiation of secondary flooding (**4.22 EN12094-1 Standard)** |
| 1. Manual only mode (**4.23 EN12094-1 Standard)** |
| 1. Control of extended discharge (**4.28 EN12094-1 Standard)** |
| 1. Emergency abort device (**4.27 EN12094-1 Standard)** |
| 1. Triggering of equipment outside the system **(4.26 EN12094-1 Standard)** |



# 6.5 SLU Interface Module

Communication with the SLU will be via serial port.

The existing SLU protocol will be ported from the MZX

panel software. An API will be provided allowing the UI Module to access SLU LEDs and buttons.

(Add command protocol description)

# 6.8 Extinguishing Application

The overall state of the system will be controlled by the Extinguishing application. The standards related

use cases constraining this state machine are listed in Section 7 below. States include alarm, releasing,

count-down timing, sounder active, valves open/shut, and relays active.

The following states and transitions are anticipated:

Complete

Extinguishing

Fire

Hold-off

Delay timer expiry

Hold button

Monitoring

Reset

Fault

Manual release selected or fire message from panel

Hold released

Extinguishing empty

Figure 13. Basic extinguishing application

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| State/Events | **Monitoring** | **Fire** | **Hold-off** | **Extinguishing** | **Complete** | **Fault** |
| **Manual release** | Fire |  |  |  |  |  |
| **Gas flow detect** | Fire |  |  |  |  |  |
| **Panel fire** | Fire |  |  |  |  |  |
| **Delay timer expiry** |  | Extinguishing |  |  | Monitoring |  |
| **Hold off pressed** |  | Hold-off |  |  |  |  |
| **Hold off released** |  |  | Fire |  |  |  |
| **Abort pressed** |  | Monitoring | Monitoring |  |  |  |
| **Monitored input fault** | Fault |  |  |  |  |  |
| **System fault** | Fault |  |  |  |  |  |
| **Gas released** |  |  |  | Complete |  |  |

**Monitoring State**

Read monitored inputs

Check for faults detected

UI Poll for keys pressed

Update Display LEDs (power)

Check received loop packets

Figure 13. State transition table extinguishing application

**Fire State**

Send MX message to panel

Update activated LED

Update Manual Release LED

Update Sounder LED

Start delay timer

Turn on sounder

Check for faults

UI Poll for keys pressed

**Extinguishing Complete**

Send MX message to panel

Start inhibit timer

Close Valve 1 and monitor for closed

Close Valve 1 and monitor for closed

Poll UI again

Update Extinguishing released LED

**Extinguishing State**

Send MX message to panel

Lock UI

Pulse buzzer

Open Valve 1 and monitor for opened

Start delay timer

Open Valve 2 and monitor for opened

Monitor extinguishing level and start timer

Monitor for gas flow

Figure 14a.

Ext Application

**Fault State**

Send MX message to panel

Activate buzzer

Display buzzer LED

Update LEDs for one or more of

the following:

System Fault, Fault, Incorrect Status

Agent Low, Gas Trapped, Blocked

Sounder fault, Earth fault

# 6.9 Releasing State Machine

The following state machine will be implementing within the Extinguishing application:

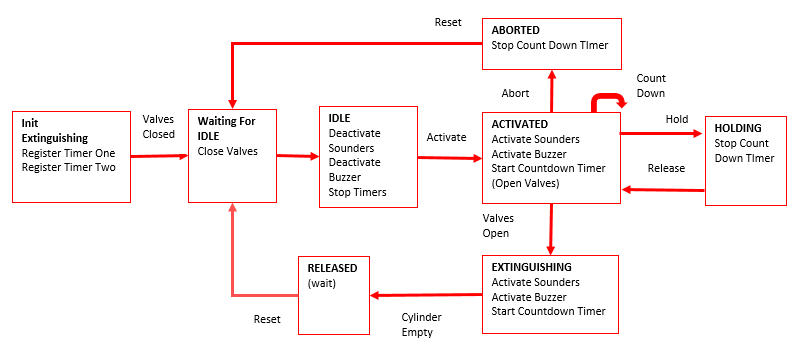
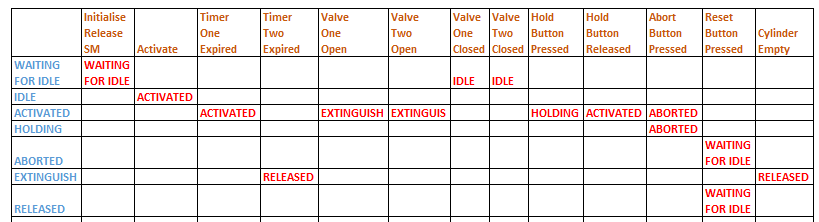


Figure 14b.

Releasing SM



# 6.10 Loop Driver

*An ARM M0 MCU will be provided as a loop interface. It has 32KB flash and 8KB SRAM. The countdown*

*timer will not be updated via the loop protocol due to inherent lags on the communications with the*

*panel. If communication is lost with the loop then the extinguishing module can be switched to manual*

*only mode.*

Several interface options have been proposed. Each requires different amounts of software.

LOOP

ASIC Chip

MX Comms purpose only

All analog and digital inputs tied to ground

Interface uC

UART comms buffer

talks via UART to main board

MX+?

Isolator (opto)

NXP Main Board processor

Figure 15. Loop driver physical interface

The following physical interfaces have been proposed.

## ASIC with processor control

One possibility is using a single ASIC and an MCU, which will write and read data to and from the ASIC’s EEPROM (between polling of the ASIC). The MCU will carry out this data transfer by inhibiting ASIC’s loop driver and switching ASIC FSK input (from internal discriminator to external discriminator). This approach has not been not developed to date apart from the Engineering Management Tool EMT850 which continuously inhibits the loop driver and changeover FSK input with additional hardware added to the module. This approach allows the possibility of data exchange of up to 201 bytes between the CIE and the ECD. This data transfer allows the CIE to track the status and mode of operation of each of monitored input and output (on the ECD). This technique will probably have reduced embedded firmware requirements compared to simulating the ASIC.

## ASIC only

The least preferred interface is a hardware solution which uses two ASICs. The limitation of this approach is the effective data bandwidth as a fraction of the ECD’s I/O states are transmitted back to CIE during a poll. This approach uses the ASICs’ digital and analogue inputs to monitor the outputs of addressable I2C devices, which are connected to the main MCU via complex I2C opto-couplers. These opto-couplers would consume several mA and require significant board area.

Software simulated interface

Another interface is an MCU based unit which can simulate the ASIC’s functionality. The ASIC is a slave device to the master CIE device and as such replies to the master’s FSK data packets with slave FSK data packets. This approach has been partly researched for new products as well as the new R&D protocol simulator box. The simulation of the ASIC functionality includes the following; FSK reception of data packets from the MZX loop, FSK generation of data packets as a slave device, simulation of digital inputs & outputs and the analogue inputs and the interrupt facility. The simulation must cope with reception of Global commands (commands addressed to all slave devices on the loop) but does not need to respond to any except GlobAdd (global address request) which only occurs during loop initialization. The ability to respond to interrupts and generate an interrupt is a Goal. With this technique, the simplest software development is to map the device as a block of addresses to allow several bytes of data (up to 32) to be exchanged between CIE and ECD.

## EXM851 Signals currently to be supported by Panel

## (Reference: EXM851 Loop API Requirements Rev 0.2)

The following diagram is the first proposal for ASIC interface from Panel team:

SUMMARY of OPTIONS

**Panel**

**EXM851**

**(ASIC)**

First Stage Alarm

Second Stage Alarm

Reset

Panel Fault

Disable Auto Mode

Activated

Gas Flowing/Released

Local Fault

Do0

Do1

Do2

Do3

Do4

Ai0

Ai1

Ai2

Analogue Loop

Set Out

Fast AV

ASIC Chip

MX Comms purpose only

All analog and digital inputs tied to ground

Interface uC

UART comms buffer

talks via UART to main board

Isolator (opto)

NXP Main Board processor

**A B C D**

## Interface A

1. Use SetOUT and FastAV (as per above diagram) commands to set commands and retrieve status
2. Use ReadEE and WriteEE commands to pass data via ASIC EEPROM

## Interface B

1. Write to ASIC EEPROM via FSK input (switching out the Loop comms) using MX+
2. Drive and read ASIC GPIO+ADC lines directly from Interface uC

## Interface C

1. Pass through MX+ protocol into serial port of interface uC for Interface A solution 1
2. Proprietary serial protocol for Interface A solution 2

## Interface D

1. Proprietary serial protocol (interpreted by Interface uC)
2. MX+ protocol (pass through)

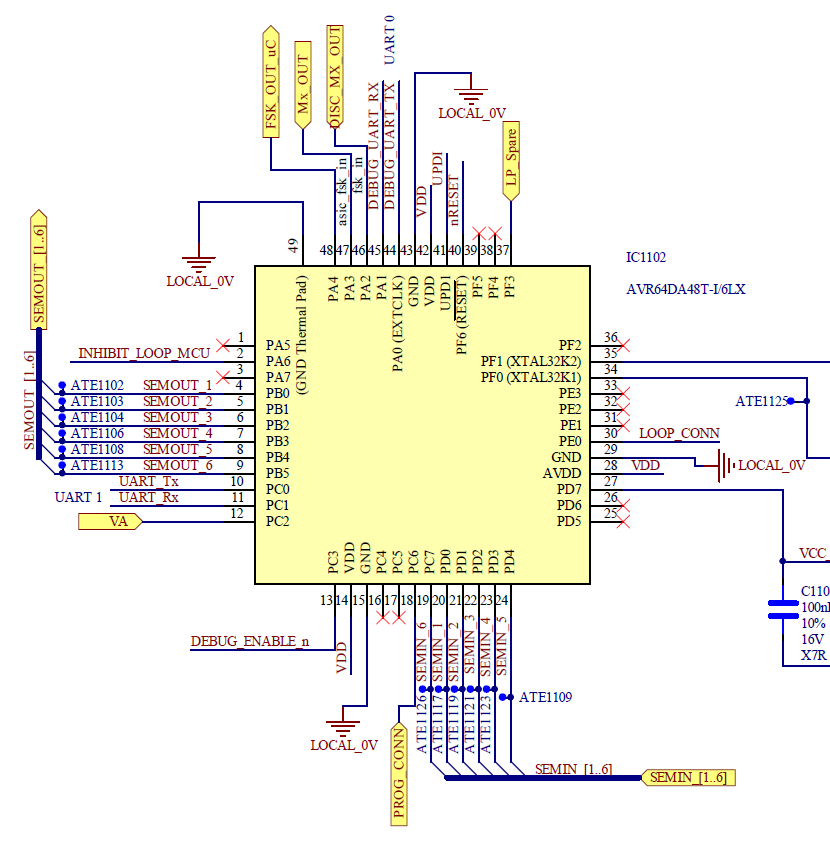
# 6.11 MX Protocol

The following data ultimately (but not at first) has to be passed between the CIE and ECD via MX Protocol.

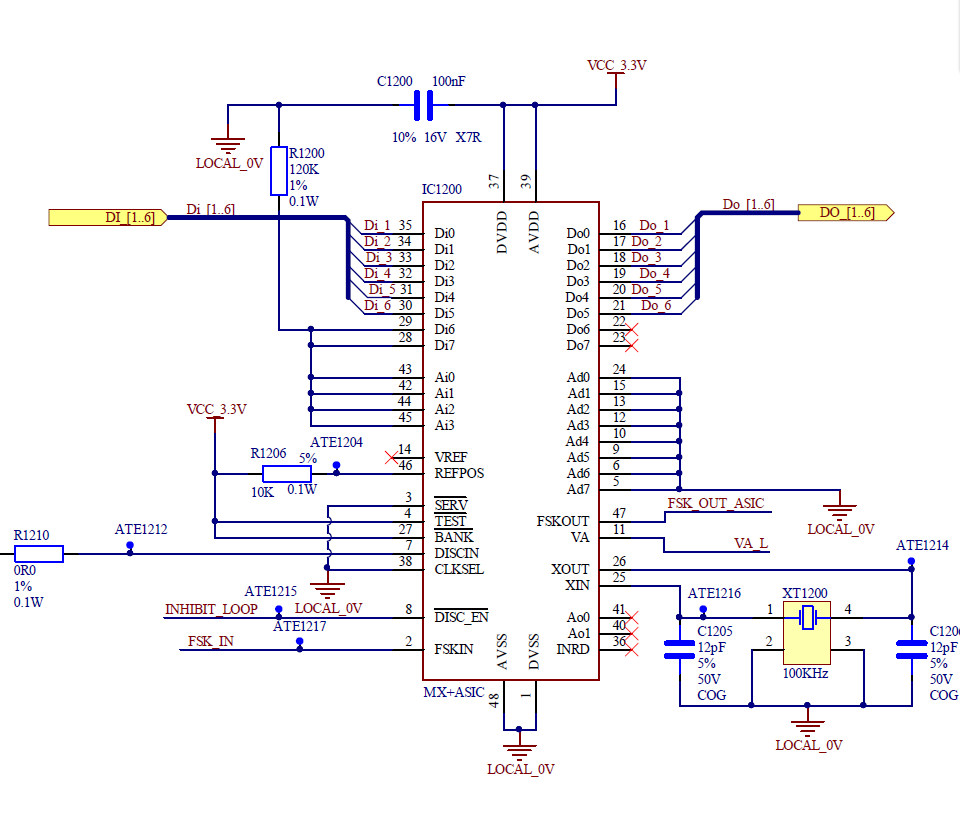
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ECD to CIE |  |  | CIE to ECD |
|  | **State / Condition / Function** |  | **State / Condition / Function** | **Mandatory** |
| Quiescent | Quiescent |  | FDAS fault | T |
| Fault | Common fault |  | Alarm signaling fault |  |
| circuit/TP fault |  | fault signaling fault |  |
| earth fault |  | First stage alarm |  |
| power fault, system fault indicated only at ECD |  | Second stage (confirmed) alarm | T |
| Disablement | Common Disabled |  | Manual release |  |
| Group disabled |  | Disable/re-enable command |  |
| Sounders |  | Valves | T |
| Valves including stepper motor |  | Released signal | T |
| Released signal |  | fault signal | T |
| Fault signal |  | spare |  |
| Spare/FPE |  | Circuits 1-32 |  |
| Extinguishing System Warnings | Blocked Warning |  | Could be conflict with |
| FES status |  | disablement if only |
| Blocking device fault |  | some circuits of a |
| Agent low Warning |  | function are disabled |
| Gas in Manifold Warning |  | Fault status request |  |
| Manual release | Manual release only |  | Sounder silence/resound | T |
| Manual release disabled |  | Reset command | T |
| Manual release activated |  | Access level 2 | Inf |
| ECD states | Activated |  | Access level 3 | Inf |
| Hold active |  | Test command |  |
| Abort activated |  |  |  |
| Gas Released |  |  |  |
| Second stage release active |  |  |  |
| Extended flooding |  |  |  |
| Cylinder Bank selected | Cyl Bank 1 active |  |  |  |
| (does not affect ECD state) | Cyl Bank 2 active |  |  |  |

# 6.12 Loop Interface

## Loop Processor



## Loop Interface ASIC



## Loop Protocol

A proprietary loop interface protocol will be designed to transfer groups of statuses between the Loop Interface board and the extinguishing module. It is based on functionality of objects. It will contain all future statuses and information but interpreted by the interface uC according to its ability to interface to the ASICs 8 digital inputs, 8 digital outputs and four analogue channels.

The protocol will provide messages to the relating to the following groups:

1. Sounders
2. Valves
3. Release
4. Extinguishing System
5. ECD
6. Alarm

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Group** | **Panel To ECD** |  |  |  |  |  |  |
| **Sounders** | Silence | Resound | Isolate |  |  |  |  |
| **Valves** | Isolate | De-isolate |  |  |  |  |  |
| **Release** | Manual release | Isolate |  |  |  |  |  |
| **Manual Controls  - Manual Release - Abort Button - Hold Button** |  |  |  |  |  |  |  |
| **Extinguishing  System** | Isolate (signals) | De-isolate (signals) | Reset  (System) | Access  Level | Test | Status  Request |  |
| **ECD** |  |  |  |  |  |  |  |
| **Alarm** | First Stage  Alarm | Second Stage Alarm | Signaling  Fault |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Group** | **ECD To Panel** |  |  |  |  |  |  |  |
| **Sounders** | Disabled/ Isolated | Enabled | Active | Inactive | Fault |  |  |  |
| **Valves** | Disabled/ Isolated | Enabled | Active | Inactive | Fault |  |  |  |
| **Release** | Disabled/ Isolated | Enabled | Active | Inactive | Fault |  |  |  |
| **Manual Controls  - Manual Release - Abort Button - Hold Button** | N/A | N/A | Active | Inactive | Fault |  |  |  |
| **Extinguishing  System** | Blocked | Agent Low | Gas in  Manifold | N/A | Fault - Common - Circuit - Internal - Power - Earth - System - Blocking Device | Cylinder Bank  Active (1 or 2) | Quiescent |  |
| **ECD** | Activated | De-activated | Hold active | Aborted | Gas released | Second Stage  Release | Extended  Flooding |  |

# 6.13 Serial Communications

Multiple interfaces. These include stepper motor communications, UI daughter board communications,

serial port communications for test and configuration/firmware upgrade, loop driver module

communications. State machines need to be created for each of these (all similar, illustrate example case

here). <Use existing PLX code. Modify for ARM M3 platform>.

* Flexcom configuration
* Clock configuration
* Interrupt handling
* Buffering
* Configuration
* Read state machine
* Write state machine (half duplex)

## USART Driver State Machines

USART Driver modules will be created for each peripheral. These include Stepper Motor, SLU and ISO. Each driver will contain half duplex state machines controlling the RS485 interfaces individually. A figure of eight style of TX and RX buffering is recommended: (USART0.c, USART1.c, USART3.c, USART6.c) New USART modules can be added as required.

Packet to send

Packet sent

Packet received

USART\_FC1.c

USART\_FC0.c

USART\_FC3.c

RX Packet

TX Packet

Ring Buffer

Figure 13. USART Flexcom 0, 1, 3, 4, 6 etc.

Packet RX detected

Figure 16. UART Modules

## USART Configuration

### GPIO Configuration

Each USART module will initialise its own GPIO pins, Flexcom registers and Clock configuration. Interrupts will be configured to ensure events are created for state changes which result in non-blocking processing.

Each USART PIN is configured as TYPE D with the digital mode set to 1 and the function number set according to the **Table 173. (Type D I/O Control registers: FUNC values and pin functions)** inNXP\_LPC546xx\_User\_Manual (Rev 1.3). Each pin may have a different function number for their respective Flexcom blocks

### Flexcom configuration

Each Flexcom block has a full set of USART configuration registers which will need to be set according to the interface required. (parity, stop bits, data length and TX/RX may vary)

e.g. Flexcomm0 (General Purpose serial port)

pUsart0CfgReg->usart\_ConfigReg.enabled = 1;

pUsart0CfgReg->usart\_ConfigReg.parity = 0; /\* 0=none,1=reserved,2=even,3=odd \*/

pUsart0CfgReg->usart\_ConfigReg.stopLen = 0; /\* 0=1 stop bit, 1=2 stop bits \*/

pUsart0CfgReg->usart\_ConfigReg.dataLen = 1; /\* 0=7 bits,1=8 bits,2=9 bits \*/

pUsart0CfgReg->usart\_ConfigReg.ctsEn = 0;

Each Flexcom block will require a clock source to be selected:

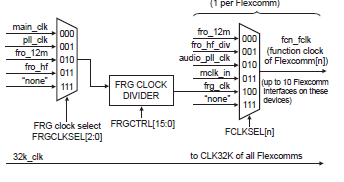


Figure 17. USART Clock sources

/\* Per FLEXCOMM clock 0x2B0 -> 0x2D4 4 bytes per FC\*/

**#define** FCLKSEL\_12MHz 0x00000000 //12000000U

**#define** FCLKSEL\_96\_48\_MHz 0x00000001 //96000000U,48000000U

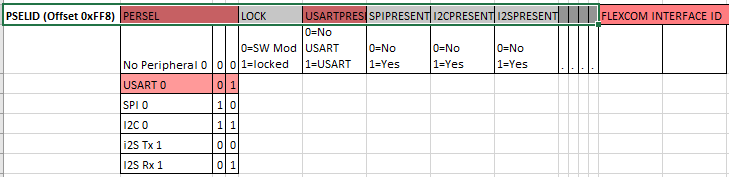
**#define** FCLKSEL\_AUDIO\_PLL 0x00000002

**#define** FCLKSEL\_MCK 0x00000003

**#define** FCLKSEL\_FRG\_CLK 0x00000004

**#define** FCLKSEL\_NONE 0x00000007

Each Flexcom block has a PSELID register that needs to be set to select the USART function:



### Fractional Rate Generator

FRGCTRL must be calculated according to the Baud Rate required. The following registers should be set.

|  |  |  |  |
| --- | --- | --- | --- |
| **FRGCLKSEL (0x2E8)** | | SEL (0:2) | |
| Fractional Rate Gen FRG Clock Source  select (0x2E8) | | 0x0 Main Clock | |
|  | | 0x1 System PLL | |
|  | | 0x2 FRO 12 MHz | |
|  | | 0x3 FRO 96 or 48 MHz | |
|  | | 0x7 None | |
| **FCLKSEL0** (0x2B0 | | SEL (0:2) | |
| **FCLKSEL1** (0x2B4) | | 0x0 FRO 12 MHz | |
| **FCLKSEL2** | | 0x1 FRO 96 or 48 MHz | |
| **FCLKSEL3** | | 0x2 Audio PLL | |
| **:** | | 0x3 MCK (MCLK) | |
| **FCLKSEL9** (0x2D4) | | 0x4 FRG Clock | |
|  | | 0x7 None | |
| **FRGCTRL Fraction Baud Rate  Generator (0x3A0)** | DIV (always 0xFF) | | MULT (Numerator) (0 -> 255) | |

Then configure the baud rate divider BRGVAL in the BRG register. The baud rate divider divides the Flexcom Interface function clock (FCLK) to create the clock needed to produce the desired baud rate.

***Generally: baud rate = [FCLK / oversample rate] / BRG divide with specific register values: baud rate = [FCLK / (OSRVAL+1)] / (BRGVAL + 1)***

|  |  |
| --- | --- |
| **USART\_BRG (0x20)** FCLK Divisor  before use by USART | BRG (15: 0) |
|  | 0 = FCL |
|  | 1 = FCLK/2 |
|  | 2=FCLK/3 |
|  | 4 = FCLK/4 |
|  | 0xFFFF = FCL/65535 |
|  |  |
| **USART\_OSR (0x28)** Over sample register (default 16) | 0x4 -> 0xFF function clocks  used to sample Tx and  Rx data |

### USART FIFO

Each USART has a configurable FIFO buffer which can trigger interrupts when the following conditions occur:

FIFOSTATUS register bits: txerr, rxerr, txempty, txnotfull, rxnotempty, rxfull, txlvl, rxlvl:

The corresponding NVIC interrupt controller flags must be set. (There is one for each Flexcom block)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **NVIC (0xE000E000)** | Bit 14 | Bit 15 | Bit 16 | Bit 17 | Bit 18 | Bit 19 | Bit 20 | Bit 21 |
| **ISER0 (enable reg)** | ISE\_FC0 | ISE\_FC1 | ISE\_FC2 | ISE\_FC3 | ISE\_FC4 | ISE\_FC5 | ISE\_FC6 | ISE\_FC7 |

***(NOTE: FC0 to FC7 Correspond to Interrupt Vectors 14 to 21 respectively)***

The FIFO is configured by two registers.

FIFOCONFIG register bits: enableTx, enableRx, size, dmaTx, dmaRx, wakeTx, wakeRx, emptyTx, emptyRx

FIFOTRIGCONFIG register bits: txlvlena, rxlvlena, txlvl, rxlvl

## Stepper motor UART

RX Buffer

Contains a stream of bytes that are later parsed within the scheduler

RX empty

TX FIFO empty

TX IDLE interrupt

TX UART FIFO FULL

TX Buffer

- Add STX

- Add packet length

- Populate packet

- Add ETX

- Increment wrIndex

- Set FC0 trigger

TX FIFO not empty

**USART\_TX\_IDLE**

Log TX state (if enabled)

Disable TX Interrupt

Enable RX interrupt

**USART\_INIT\_STATE**

usartTxState = USART\_TX\_IDLE

usartRxState = USART\_RX\_IDLE

Clear TX and RX buffers

Set pins for Type D, digimode 1, funtion 1

Set config register

Enable FIFO RX and TX

Set FIFO trigger level 0 (1byte TX and RX)

Enable rxlvl interrupt

Turn on FC 0 interrupt

**USART\_RX\_IDLE**

Log RX state (if enabled)

Clear status register

set rxlvl interrupt for ready to receive

STX

**USART\_TX\_TRANSMITTING**

Check RX state

Check TX FIFO count

Log TX state (if enabled)

Disable TX IDLE Interrupt

Disable RX LEVEL interrupt

Set TX FIFO Trigger

Check TX FIFO Byte at current rdIdx

If it is STX get next byte and save as packet length. If length is greater than UART FIFO load max UART FIFO and save bytes remaining.

If all bytes are loaded Check for ETX. and go to USART\_TX\_WAITING\_TO\_TRANSMIT

**USART\_TX\_WAITING\_TO\_**

**TRANSMIT**

**USART\_RX\_RECEIVING**

Log RX state (if enabled)

While rxlvl not zero copy next data byte to buffer

Is end of packet character received?

Post packet to client.

Clear status word

Set rxlvl interrupt

RX Not empty

STX

Usart5 is dedicated to RS485 communications with the stepper motor. It utilizes a TX, RX and Data enable pin. Packets are expected to range in length from 11 to 16 bytes (full size of the internal modem RX FIFO). In order to avoid overflow we immediately place received bytes into the RX buffer from the FIFO to be parsed later. The format of a received packet is as follows:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Header (496) | Frame number | Number of bytes | Dest type | Dest  all | Sys number | Sender type | Sender address | Function code | Function sub -code | Data bytes (up to 6) | checksum |

### USART5\_ParseRxBuffer

The RX buffer is a ring. It contains a read and a write index. Provision is made for a packet crossing the boundary of the buffer and wrapping around the top of the ring.The buffer is searched for a header byte , the length is then extracted and likewise the crc. Basic checks are done to ensure the length is in range and the CRC is non-zero. The packet is then copied locally and forwarded up to stprcomms module and cleared. Stprcomms is where the CRC is checked and the Function Code and Function Subcode are ‘cracked’.

**RX RING BUFFER(48 32-bit words)**

crc 1

**header 2 (496)**

frame num 2

frame num 1

**header 1 (496)**

crc 2

num bytes 1 (words)

num bytes 2 (words)

**RX RING BUFFER(48 32-bit words)**

RX2 Packet (16 bytes)

RX1 Packet (12 bytes)

RX2 Packet wrapped (16 bytes)

num bytes 1 (words)

frame num 1

**header1 (496)**

RX2 Packet (16 bytes)

RX1 Packet (12 bytes)

crc 1

crc 2

**header 2 (496)**

frame num 2

num bytes 2(words)

Complex example

Simple example

RX Buffer (48 bytes)

# 6.14 I2C Driver

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| <Use existing PLX code. Modify for ARM M3 platform>.  There are currently three I2C chips driving input switches and output LEDs. In future there will be more in order to drive a matrix of LEDs for the UI board. Below is an example of the status LED I2C driver chip.    Figure 18. I2C Connections  Scheduling of data in and out of the I2C port is done by modifying the control register and reading the status register on interrupt. There are a number of interrupts that can be enabled. See below for the register structures.  The following signaling pattern is needed to drive these chips.    Figure 13. Output TX sequence  There is a configuration command, a request data command and a write data command. The pattern above is for writing data. Each byte is shifted out one by one with an ‘acknowledge’ between each. As many bytes can be sent as thee are ports on the I2C chips. For TX the acknowledge byte is read in the I2C status register as TX\_READY (held in the mststate flags of the status register). It is preceded by an mst-pending interrupt (reflected by the mstpending flag in the status register). A stop flag is set when completing the transmission.  **Status Register**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | mst pending | mst state | mst arbloss | mst stperr | slv pending | slv state | slv notstr | slv idx | slv sel | slv desel | mon rdy | mon ov | mon active | mon idle | event timeout | scl timeout |   Controlling the output of the bytes is done by writing start, stop and continue to the Master Control register below. Each byte is acknowledged via mst-pending interrupt (read in the status register).  **Master Control Register**   |  |  |  |  | | --- | --- | --- | --- | | continue | start | stop | dma | |

TX Command

MST Pending ISR

TX Command

Queue

MST Pending ISR

MST Pending ISR

Figure 19. TX state machine

## Initialising the I2C port

The I2C port must be associated with its respective Flexcom interrupt bank. The clock for the Flexcom mechanism must be set as with USART port. The I2C port itself has a clock register and divider which is programmed as a fraction of the Flexcom clock. Any of the status interrupts can be enabled. The mst-pending interrupt is the primary one but it is useful to use all to diagnose state machine issues.

## Configuring

On power up the I2C chips must be configured by sending their output/input states per pin.

This is done using a ‘110’ binary command to each chip. These commands are queued into the TX state

machine in the same way as a port write.

## IDLE State

* On entry to IDLE check the Command Queue for another command to send.
* Self-transition to Transmitting Bytes state.
* Otherwise, clear status flags and interrupts

## Transmitting State

* Remove CMD from FIFO ring buffer.
* Send each byte one by one waiting for the ‘mstpending’ interrupt to schedule the next byte.
* As many bytes as there are ports to drive.
* (First byte is address, second is write command, third and fourth bytes map to the port pins of the device)
* Any byte can be NACK’d by reading mststate. (i.e. the following responses can be read….DATA\_TX\_READY, RX\_READY, NACK\_DATA, NACK\_CMD)

## Waiting for IDLE State

Once a transmission is complete, we must wait until the mst-pending flag is set before we can begin

another transmission. Hence, we send a ’stop’ control command and wait.

# 6.15 Timers/Counters

The NXP LPC 546XX processor has four timer-counters available. Two of these are part of the synchronous peripheral set and two are part of the asynchronous peripheral set of the MCU. They can be configured as free running timers which restart after a specific number of clock cycles. A prescaler register can be used to multiply up the resolution of each timer. Four match registers per timer are available which can be used to generate an interrupt at a specific timer value and the timer(s) restarted. The clock sources, resolutions and speeds can be selected using the following registers:

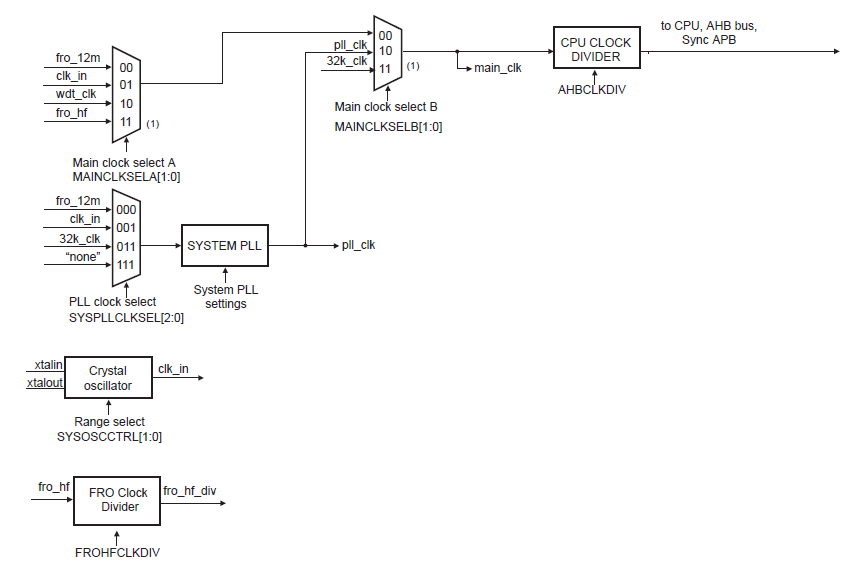


Figure 20. Clock path to CTIMERs

## MAINCLKSELA

This register selects one of the internal oscillators, FRO 12MHz, 48MHz, 96MHz, system oscillator, or watchdog oscillator.

## MAINCLKSELB

This register selects the clock source for the main clock. All clocks to the core, memories,

and peripherals are derived from the main clock. Sources include: MAINCLKSELA, PLL, or RTC 32KHz

The AHB block selects which timers should be enabled:

## AHBCLKCTRL (0,1,2)

These registers enable clocks to each peripheral. AHBCLKCTRL 1 enables CTIMERs 0, 1 and 2. However CTIMERs 3 and 4 are enabled via the ASYNCAPBCLKCTRL register. The clock path to the Asynchronous APB block is enabled via ASYNCAPBCLKSELA register (part of the SYSCON register set)

Note: Each clock register has their own set and clear registers as well. It is recommended that these are used instead of the clock control registers directly.

## Capture and Match

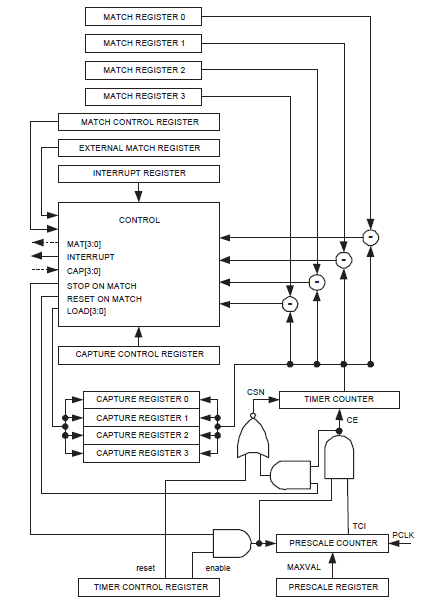
It is not expected that the capture feature be used initially as there is no signal transition which requires measurement. However, the match registers will be used for extinguishing timing features. 

Figure 21. Timer and Capture architecture

Timer and Counter Interrupt enabling

TIMERMATCHCTRL Register: Enables interrupt for each match register and specifies whether it

should reset or stop the timer after interrupt occurs.

TIMERINTERRUPT Register: Enables individual interrupts for match registers 0,1,2,3 and capture

registers 0,1,2,3

TIMERCTRL Register: Enables and resets the Timer/Capture feature

TIMERCAPTURE and TIMERMATCH Registers:

Hold up to four match values and capture values

COUNTERCTRL Register: Specifies Timer or Counter operation and trigger behavior

The CTIMERn bit should be set in the NVIC Interrupt enable register and the Vector table updated:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **BASIC INTERRUPT**  **CONFIGURATION** | **NVIC Register(0xE000E000)** | Bit 10 | Bit 11 | Bit 12 | Bit 13 |
|  |  | Mask 0x400 | Mask 0x800 | Mask 0x1000 | Mask 0x2000 |
|  | ISER0 (enable reg)  Offset 0x100 | ISE\_CTIMER0 | ISE\_CTIMER1 | ISE\_SCT0 | ISE\_CTIMER3 |
|  |  | Bit 4 | Bit 5 |  |  |
|  |  | Mask 0x10 | Mask 0x20 |  |  |
|  | ISER1 (enable reg)  Offset 0x104 | ISE\_CTIMER2 | ISE\_CTIMER4 |  |  |
|  |  |  |  |  |  |

Vectors for each timer are as follows:

* CTIMER0 = Vector 10
* CTIMER1 = Vector 11
* CTIMER2 = Vector 36
* CTIMER3 = Vector 13
* CTIMER4 = Vector 14

## Using Timers

The timer module will start several free running timers of varying resolutions. A client module will register a timeout value for the appropriate timer interrupt to access. The module contains a fixed table of timers with decrementing counters. The timer status is then checked by the client module within the main loop. The Timer module will provide APIs to create, start and stop timers.

INPUT: timer value

timer type

(count-down, reset)

RETURN: pointer to timer

CLIENT MODULE

(initialization)

**Timer table entry:**

Timer name

Timer type

(reset or countdown)

Timer value

Status (ACTIVE, NOT\_ACTIVE, EXPIRED)

TIMER MODULE

Timer table

(n entries)

register, start, stop, reset.

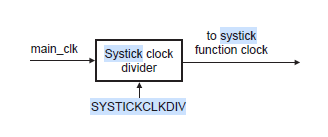
Register timer on start up. Start, stop, reset, check for expired during main loop.

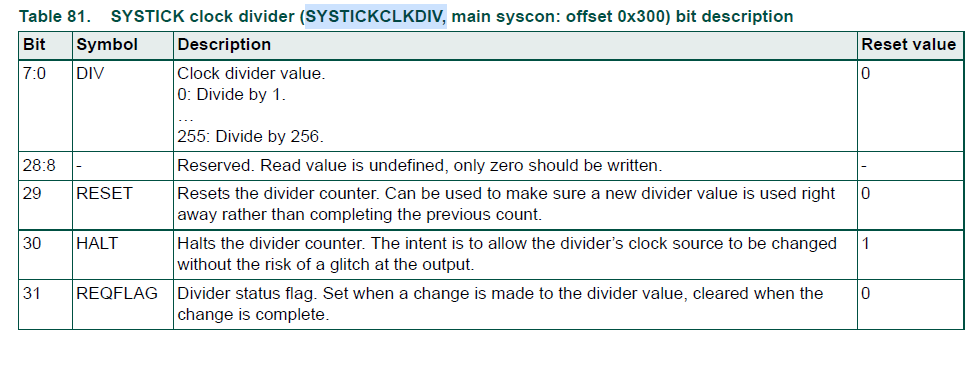
Timer interrupt

Tick depends on resolution. Decrements active timers.

## System Tick

The system tick timer is fed by the main clock.



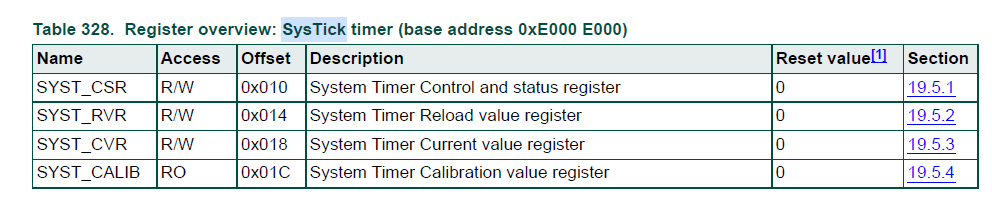


The SysTick timer is an integral part of the Cortex-M4. The SysTick timer is intended to

generate a fixed 10 millisecond interrupt for use by an operating system or other system

management software.

Control of it is via four registers: CSR, RVR, CVR, CALIB



A task control structure should be created which requires a module to register a callback at a specific rate depending on its priority. This callback is then triggered by the number of 10ms ticks registered. The tick counter expiry will signal the expiry of this tick count and the callback will be serviced by the main loop process.

In order to generate a 10ms tick, the reload register value is calculated as follows:

(systick clock frequence x 10ms) -1

= 12MHz x 10ms - 1

= 120000 – 1

= 119999

= 0x1D4BF

# 6.16 Stepper Motor Module

A module should be created that creates and receives packets to drive the stepper motor. It will have knowledge of the stepper motor behavior and control as well as its current state. It will provide an interface to the main extinguishing application. It will communicate to the stepper motor using the FC4 RS485 Usart interface. Telegrams are sent that modify EEPROM in the motor controller. The following status values can be returned:

NOT\_PROG bit 0 Unit address not set or release time not set

FAULT\_STEPPER bit 1 Motor winding open circuit

FAULT\_MEM bit 2 Checksum incorrect (Inhibit operation)

READY\_FOR\_REL bit 3 Unit is in home position

RELEASING bit 4 Motor is running forwards (opening)

REL\_RECEIVED bit 5 Release telegram received (Cleared by RETURN command)

RELEASED bit 6 Unit is in END position

RETURNING bit 7 Motor is running backwards

**Interface settings:** 9600 BPS, 9 data bits ,1 start bit, 1 stop bit, no parity.

**Telegram format:** Format: H, Fn, N, Dt, Da, Sn, St, Sa, **Fc, Fs**, D1, D2, C

**Command FC, FS D1, D2 Notes**

|  |  |  |  |
| --- | --- | --- | --- |
| System Check Request | 92,0 |  | Sent every 5s |
| System Check Reply | 92,1 | D1=release time  D2 = Status |  |
| Release | 143,0 |  | Transmit every 65ms  until status changes |
| Release Confirmation | 143,1 |  |  |
| Return | 143,2 | D1=release time  D2 = Status | Stop release |
| Return Confirmation | 143,3 | D1=release time  D2 = Status |  |
| Set Release Time | 208,2 | D1=release time  seconds |  |
| Set Release Time Confirmation | 208,3 | D1=release time seconds |  |
| Set Unit Address | 208,0 | D1=address (1->254) |  |
| Set Unit Address Confirmation | 208,1 | D1=new address |  |
| Update Checksum | 207,0 | D2=status | Must be done on release time or address update |
| Update Checksum Confirmation | 207,1 |  |  |
| Software Version Request | 93,0 |  |  |
| Software Version Reply | 93,1 | D1=High version  D2=Low version  D3= Project no. high  D4=Project no. low |  |
| Number of Restarts Request | 94,0 |  |  |
| Number of Restarts Reply | 94,1 | D1=number of restart | rollover after 255 |

Releasing Application

UI

Stepper Motor Module

Stepper Comms Module

USART Driver

Abort

Release activated timer (100ms)

Timer (5s)

**Not Programmed**

Alternate between sending release time, Checksum calculate and SysCheck request every five seconds until the SysCheck response yields ‘Ready to Release’ flag set.

This state is entered from all states if the NOT\_PROG flag is set

Start up

**Ready For Release**

Send system check request every five seconds to determine current stepper state.

Ready to release

**Return Activated**

Stop releasing activated timer. Alternated between sending a system check request and return request until we get a system check response to drive us to the next state.

System check

timer (5s)

System idle

System check

timer (5s)

**Release activated**

Stop system check timer send release request and start release activated timer

Release activated timer (100ms)

Timer (5s)

Ready to release

**Released**

Stop system release activated timer. Send a system check message and start system check timer.

System check

timer (5s)

System idle and key wound in

System check

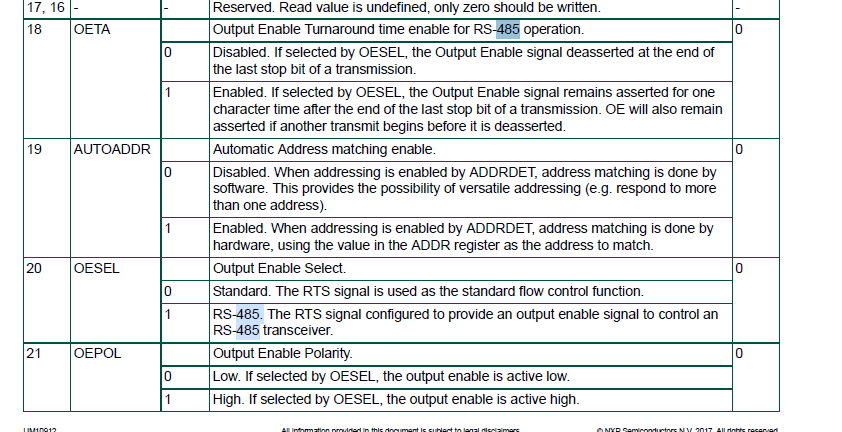
timer (5s)

Released

Figure 22. Stepper Motor State Machine

.17 RS485 Comms

In order to control the Half-duplex nature of the RS485 communication a half-duplex state machine must be written. The OESEL flag in the uart config register should be set so that the output enable signal can switch the power to the RS486 bus on and off when scheduling data out. Thus avoiding collisions from slave devices.



The following RS485 connections are assumed@

Interface Flexcomm ID TX pin (SCL) RX pin (SDA) OEN pin

Stepper Motor RS485\_1 FC4 P1\_20 P0\_20 P1\_30

SLU RS485\_2 FC3 P0\_9 P0\_9 P1\_15

