The FCAL CAN Bus

Claire Tarbert

June 27, 2010

1 Overview

The GlueX forward calorimeter (FCAL) will be comprised of 2800 lead glass blocks each viewed by a photomultiplier tube (PMT) assembly. This assembly includes a phototube plus a Cockroft-Walton base supplying high voltage to the PMT dynodes. Control of the base electronics is centered around a CAN-enabled STM8S208RB microcontroller (MCU) [1]. Details of the CAN protocol can be found in appendix A. When linked together, the bases form a 2800-node CAN bus which facilitiates communication with the bases in order to, for example, set high voltages and for monitoring purposes. This communication system relies on two pieces of software: a firmware program running on the MCU carrying out instructions on reception of a CAN message, and a complimentary server program which acts as the user interface to the bus and generates the appropriate CAN messages.

The firmware is specific to the STM8S208RB chipset and the base electronics in general. Similarly, the server program is unique to the hardware being used to access the CAN bus. At present, we have two devices available for connecting to the bus: USB-CAN dongles from Peak Systems and an Ethernet-CAN gateway from AnaGate. Server programs for both of these devices have been developed for use with Windows and Linux operating systems.

This document summarises the main features of both the PMT firmware (section 2) and the various server programs (section 3). Some basic instructions on how to install and use the software and a list of possible future developments are also included.

2 STM8S208RB Firmware

The microcontroller on the FCAL bases must be able to:

- Support CAN communication

- Set HVs (*i.e.* use a DAC)
- Read HVs (*i.e.* read an ADC)
- Communicate with an ID chip
- Provide a 1MHz clock for the base electronics

It must be able to receive a CAN message, execute the instruction contained in the message and send a CAN message back if requested. The chip must have the potential to carry out these functions and an operating system, *i.e.* firmware, installed that exploits all of these features. Listed below are the main features of the STM8S208RB:

- 24MHz 8-bit MCU
- 128 kBytes flash memory (program memory)
- 2 kBytes (for bootloader)
- 6 kBytes RAM
- 10 bit ADC with 16 channels
- CAN 2.0B interface (up to 1MBit/s)
- SPI, I^2C, UART communication
- 16MHz internal clock
- Nested interrupt controller with 32 interrupts

The STM8S208RB does not have an integrated DAC and instead uses an external DAC from Linear Technology. The following sections describe the firmware operation.

2.1 Firmware Program: "BaseFirmware"

The STM8S208RB firmware is written in C and built using the RKit-STM8 C compiler and assembler from Raisonance within the Ride7 development environment. Ride7 produces an application in hex form which can be loaded into the flash memory of the microcontroller using either the Ride7 or RFlasher tools. When the MCU boots, it runs the program as a stand alone operating system. The program is structured as follows:

1. All pins are configured in output mode.
2. The high speed internal clock is configured.
3. The clock to all unused peripherals *e.g.* I^2C, is disabled.

4. Pins used as inputs are configured.
5. Request for ID is made to external ID chip to be used in CAN message arbitration.
6. CAN communication is configured.
7. Interrupts are enabled.
8. Program enters an infinite loop.
9. On reception of a valid CAN message (valid messages being those that have successfully passed the CAN arbitration), an interrupt is generated.
10. The message data is extracted and, depending on the content, actions carried out.

A list of the MCU ports, their uses and initial configuration is given in section B.

2.2 Internal Clock

The STM8S208RB can in principle accept a high speed external clock source up to 24MHz. It also has a configurable 16MHz internal RC oscillator. This internal oscillator has been configured to act as a 2MHz clock for the CPU. It has also been prescaled to use as an output on one of the chip's pins providing a 1MHz clock for the rest of the base electronics. A list of the appropriate clock registers to set to achieve this configuration and their values can be found in table 7.

2.3 Communication with ID chip

The DS2401 chips each have a 48-bit registration number which can be used to uniquely identify the FCAL bases. Communication between the MCU and the DS2401 is done serially along a single wire. The chip is connected to one of the general purpose input output ports (GPIO) on the MCU. As the MCU pulls the GPIO pin high or low, the leading edge of the pulse activates a timing circuit in the ID chip. The ID chip then translates specific timing sequences as different commands. Similarly it can transmit information to the MCU via timed sequences across the single wire.

The specific sequence to read back the ID from the chip is:

1. Transmit initialisation pulse.
2. Send the read ID command.
3. Configure the GPIO in input mode and read back the ID.

The timings for each of these actions are listed in the DS2401 data sheet [4]. To calibrate the pulse timing of the final boards, the serial communication between the MCU and ID chip should be observed on a scope. An example of the timing is shown in figure 1.

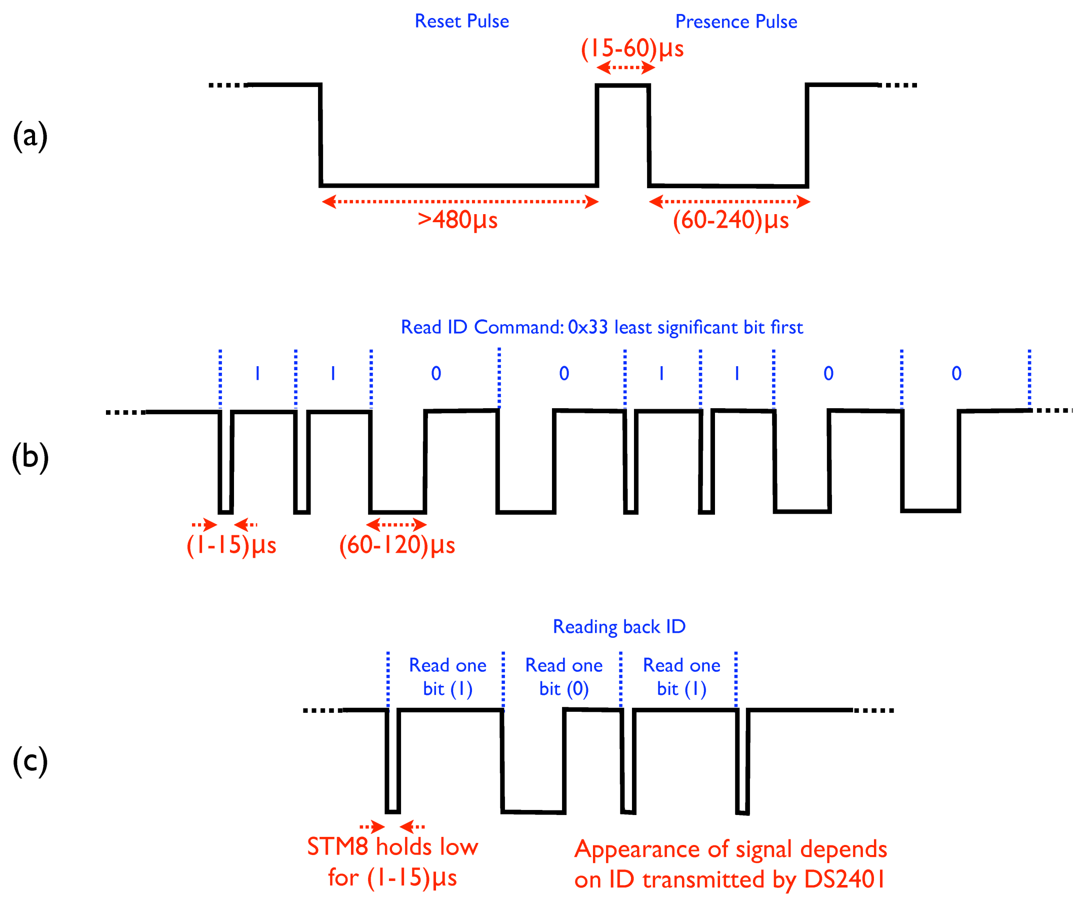

Figure 1: Illustration of correct ID chip timing you should see on a scope.

2.4 CAN communication

The PMT firmware is configured to communicate at a baud rate of 50kBit/s. If necessary this can be changed by adjusting the oscillator and baud rate prescaler registers in the $CAN\_InitTransmit()$ function.

On reception of a valid CAN message, an interrupt is generated and the firmware breaks out of the infinite loop in order to perform the requested action. Only valid messages generate an interrupt; valid messages are those which successfully pass the CAN arbitration (for details of the CAN protocol see appendix A). To do so, the incoming message must meet the following conditions:

- The 8-byte ID field of the message must be either 0×0 or $0\times00000000XXYYZZ$ where $0\times XXYYZZ$ are the 3 least significant bytes of the board ID.

- The RTR bit must be zero.
- The IDE bit must be 1 (extended messaging format).

The arbitration is configured in the $CAN\_InitTransmit()$ function.

Once an interrupt has been generated by CAN activity, the firmware parses the data field of the CAN frame. The first byte of the data field is the command byte and indicates the action the firmware should take. Extra bytes may be needed to define the command further. A list of messages understood by the bases is given in table 1 and messages that can be generated by the bases, in table 2. Messages transmitted by a base are always encoded with the base ID in the identifier field to ensure that they are not read by other bases on the bus.

2.5 Communication with DAC

The LTC2630 is a 12 bit DAC from linear technology and can be controlled using a 3-wire Serial Port Interface (SPI). On the STM8S208RB, 3 of the pins have dual purposes as both general purpose input output ports and as the MOSI, MISO and clock pins for SPI communication with peripheral devices. A fourth pin (CSS) is used to switch between different SPI modes. An example of the correct relative timing between the clock (SCK), MOSI (SKI) and CSS ($\bar{C}S$) pins can be seen in figure 2. This can be looked at with a scope for debugging purposes. Once the serial port interface has been correctly configured,

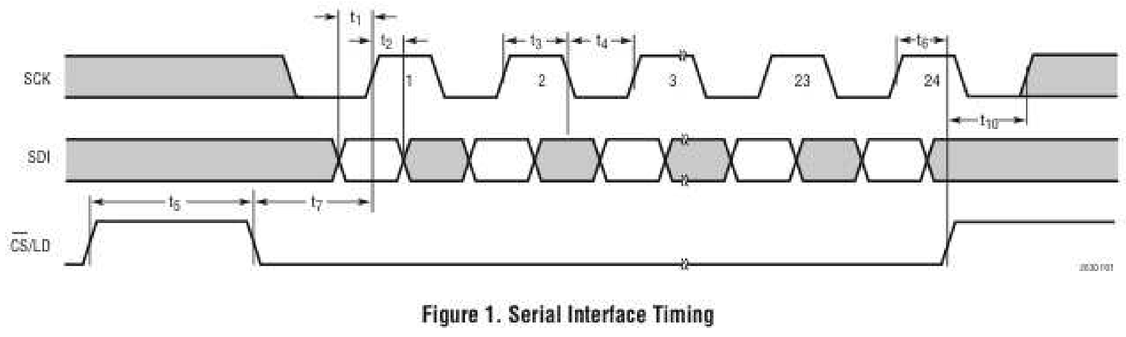

Figure 2: 3-wire SPI timing needed for communication with DAC[4].

the DAC operation is as follows [4]:

1. Initialise STM8S208RB SPI peripheral.
2. Pull chip select pin (CSS) low.
3. Send 4 bit command telling DAC to write and update DAC register.

Purpose	Command Byte	# Extra Bytes	Extra bytes	Reply from base
Read ADC	0xBB	1	0x00: Read medium voltage (bottom) 0x01: Read medium voltage (top) 0x02: Read 1st dynode voltage 0x03: Read photocathode voltage 0x04: Read DAC output voltage 0x05: Read temperature monitor 0x06: Read current monitor	y
Set Voltage	0xCC	2	New voltage (0→4096) 1st byte: most significant 8bits of new voltage 2nd byte: least significant 4bits of new voltage	n
Enable HV (experts only)	0x99	2	0x22 0x00: Enable MVT 0x20 0x00: Disable MVT 0x11 0x00: Enable MVB 0x10 0x00: Disable MVB 0x88 0x00: Enable HVT 0x80 0x00: Disable HVT 0x44 0x00: Enable HVB 0x40 0x00: Disable HVB 0x00 0x05: Enable JAM HVB 0x00 0x04: Disable JAM HVB 0x00 0x0A: Enable JAM HVT 0x00 0x08: Disable JAM HVT 0x00 0x00: Read enable bit status	n n n n n n n n n n n n y
Request Board ID	0x88	0		y
Switch LED	0x77	1	0x00: Red 0x01: Green 0x02: Off	n
Test Pulser	0xEE	1	0x00: Fire test pulser until a CAN message is received 0x01: Enable external sync to fire pulser 0x02: Disable external sync	n
Request Low Power Mode	0xAA	0		n
Read Firmware Version	0x66	0		n

Table 1: Messages understood by firmware

Purpose	**Command Byte**	**# Extra Bytes**	**Description**
Read ADC	0xBB	2	Requested voltage.
Board ID	0x88	6	Request for the board ID.
Firmware Version	0x66	2	Firmware version.

Table 2: Messages generated by firmware

4. Send 4 bits that will be ignored by the DAC.
5. Send 12 bits (most significant bit first) containing the digital value to write to the register.
6. Pull the chip select pin high.

2.6 Low Power Mode

Pulling the CAN_STBY pin (PC3) high significantly reduces the power consumption of the board, however, with CAN_STBY high, the MCU is unable to generate interrupts due to activity on the CAN_RX pin *i.e.* it can not parse CAN messages. Therefore, to enter a lower power mode, CAN_STBY is pulled high and interrupts are instead enabled on PC1 which is OR-d with the CAN_RX pin. On reception of a subsequent CAN message, the MCU detects activity on PC1, and an interrupt is generated. The function associated with this interrupt disables interrupts on PC1 and CAN_STBY is pulled low. With CAN_STBY now low, the CAN message can generate an interrupt on the CAN_RX pin, and the message is parsed as normal.

2.7 Interrupts

At present interrupts can be generated by activity on the CAN_RX pin (reception of a valid CAN message) or when the base is in low power mode, by activity on pin C3. The interrupt routines are specified in $firmware\_IT.c$ and a complete list of interrupt vectors is given in the RKit-STM8 C compiler manual [8].

2.8 Installing a new firmware version and In Application Programming

The STM8S208RB has the capability to download a firmware upgrade over CAN, however, the default STM8S bootloader is written in such a way that this would require an external oscillator connected to the chip. There is no external oscillator on the FCAL bases and a custom bootloader for the STM8S208RB needs to be written if firmware updates are to be made in this way.

At present, the flash memory can be overwritten via the Single Wire Interface Module (SWIM) pin [2]. In addition to the 10-pin connector providing power and a means of trasmitting via CAN, the bases are equiped with a separate 4-pin connector giving access to the SWIM pin. Using this connector, the USB-SWIM dongle, and the Ride7 development environment the flash memory on the MCU can be completely overwritten with a new firmware version (instructions on how to do this are given in appendix D.1).

This method of completely overwritting the flash memory is termed in circuit programming (ICP). The firmware, in principle, can also be updated via in application programming (IAP). In this technique, part of the flash memory is write-protected and a custom bootloader is stored there. Writing to the correct sequence of configuration registers on the MCU will cause the MCU to access this custom bootloader instead of the default one. The bootloader could be used to overwrite the rest of the (not write-protected) flash memory with a new firmware version downloaded over CAN. This is in development [6].

3 Server Programs

The firmware for the bases is not stand alone; a secondary program running on a PC is needed that does the following:

- Connects to the FCAL CAN bus.
- Generates CAN messages containing instructions for the bases.
- Sends messages over the bus.
- Reads and understands messages generated by the FCAL bases.

3.1 USB-CAN Communication

Previously, the connection between PC and CAN bus was via a USB-CAN dongle from Peak System Technik. Two server programs (**baseControl** and **TestServ**) were written using libraries supplied with the dongle for generating messages and sending them via USB. These interactive programs (for Linux and Windows respectively) give the user a list of options to choose from - set HV, read HV etc - and generate a CAN message that will be understood by the firmware accordingly.

3.2 Ethernet-CAN communication

The CAN-USB protocol requires a PC connected directly to the CAN bus via a USB dongle and as such means a PC would need to be attached to the FCAL. To eliminate the need for a PC in the experimental hall, we have purchased a TCP/IP-CAN gateway device from AnaGate Gmbh. In principle, the gateway is connected directly to the network and assigned an IP address. TCP-IP messages can then be sent to the gateway over the network, which in turn sends a CAN message over the bus. Similarly, when the device receives a CAN message, it relays it across the network.

3.3 AnaGate CAN Quattro

The main features of the AnaGate CAN Quattro device [5] are:

- 4 independent CAN buses
- 4 digital inputs/outputs
- TCP/IP and CAN settings configurable with a standard web browser
- C++ functions for sending and receiving CAN messages provided in Linux and Windows libraries
- Rack mountable

3.4 Server Program: NetControl

The libraries of C++ functions for controlling the TCP-IP gateway provided by AnaGate have been used to port **baseControl** for use with the AnaGate CAN Quattro - the new program is called **NetControl**.

Although NetControl has the same functionality as baseControl, the simplicity of the AnaGate library functions (listed in table 3) mean it has to be structured slightly differently. However, this simplicity also makes it very easy to use. All of the AnaGate functions, bar CANSetCallback(), have a complimentary function in the Peak CAN-USB libs. CANSetCallback() is used to define a function that will be called when a CAN message is received. If the CANSetCallBack() function has not been called, then any messages received by the gateway will be lost.

Function	**Description**
CANOpenDevice	Opens a TCP/IP connection to AnaGate device.
CANSetGlobals	Sets global variables needed for the CAN bus.
CANWrite	Sends a CAN message to the AnaGate device.
CANSetCallback	Defines a function to be called when the AnaGate device receives a CAN message.
CANSetCloseDevice	Closes an open TCP/IP connection.

Table 3: AnaGate library functions

Instructions how to compile and set up your environment to run NetControl can be found on the TaskD wiki.

4 Outstanding Tasks (as of June 27, 2010)

- In application programming.

A CAN

A.1 CAN protocol

CAN is a message based protocol, designed to allow microcontrollers to communicate without a host computer. Each node(base) on the bus is able to send and receive messages, but not simulataneously. If the bus is free any node may begin to transmit a message. If two nodes simultaneouly transmit a message CAN uses an arbitration field to allow the dominant node to transmit its data. The recessive node then resumes transmitting after the bus is clear. The arbitration field (figure 3) is a 29 or 11 bit identifier using nondestructive bitwise arbitration. Meaning that once an arbitration has taken place the recessive message need not start from the beginning. The bitwise operation views logical 0 as the dominant bit and logical 1 as the recessive bit. A message becomes recessive as soon as the bit underconsideration is recessive when compared to a dominant bit as illustrated in figure 4.

The CAN message also contains a control field, an 8 byte data field, and a CRC checksum field. The control field alerts when the identifier field has ended and the CRC field is an error detection field. CAN sets flag bits once a certain threshold of errors have occured for a particular node. The node is then turned off of the bus when it has reached a critcal number of message errors and then can no longer send or receive messages. The node can then only be turned back on by a hardware or software reset.

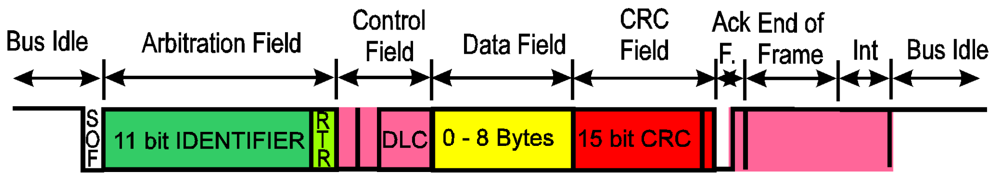

Figure 3: CAN frame format.

B STM8S208RB port designation

The pin designation for the final base design is given in table 4

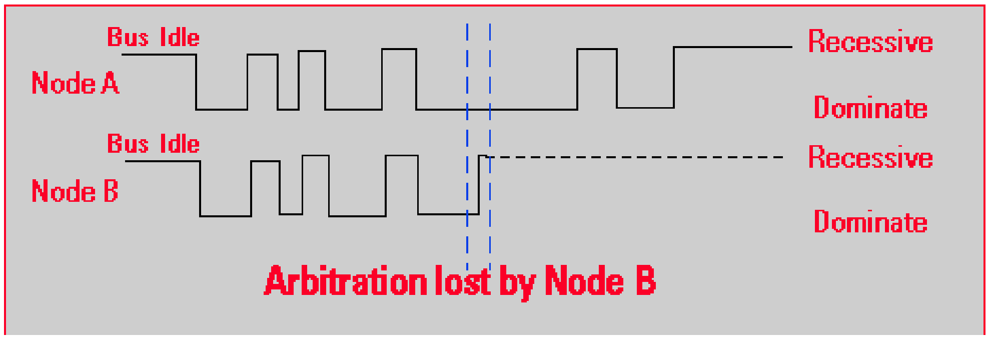

Figure 4: CAN bus arbitration.

Table 4: STM8S208RB pin assignments

Purpose	**Port**	**Pin #**	**Initial Configuration**
Reset	NRST	1	-
	VCAP	6	-
ADC_VREF (ref voltage+)	V_{REF+}	18	-
	VDDA	19	-
	VSSA	20	-
ADC_VREF (ref voltage-)	V_{REF-}	21	-
MON_TEMP (temp monitor)	PB6/AIN6	24	Floating input
MON_CURR (current monitor)	PB5/AIN5	25	Floating input
MON_VDAC	PB4/AIN4	26	Floating input
MON_VCATH	PB3/AIN3	27	Floating input
MON_VDY01	PB2/AIN2	28	Floating input
MON_MVT	PB1/AIN1	29	Floating input
MON_MVB	PB0/AIN0	30	Floating input
	PC1	34	Input
	PC2	35	Input
CAN_STBY	PC3	36	Push-pull output (fast)
SPI_CS (for DAC)	PC4	37	Push-pull output (slow)
SPI_SCK (for DAC)	PC5	38	-
SPI_MOSI (for DAC)	PC6	41	-
CAN_Tx	PG0	43	Push-pull output (fast)
CAN_Rx	PG1	44	Pull-up input
	PG5	49	Push-pull output
	PG6	50	Push-pull output
DS2401 (ID chip)	PE4	52	Output

Continued on next page

Table4 – continued from previous page

Purpose	Port	Pin #	Initial Configuration
LED (red)	PE2	54	Open drain output (slow)
LED (green)	PE1	55	Open drain output (slow)
CLK_CCO (clock output)	PE0	56	
SWIM_DATA	PD1	58	-
ENB_MVB	PD2	59	Push-pull output
ENB_MVT	PD3	60	Push-pull output
ENB_HVB	PD4	61	Push-pull output
ENB_HVT	PD5	62	Push-pull output
JAM_HVB	PD6	63	Push-pull output
JAM_HVT	PD7	64	Push-pull output
Unused	PA1, PA2, PA3 PA4, PA5, PA6 PF0, PF3, PF4 PF5, PF6, PF7 PE5, PE6, PE7 PB7, PC7, PI0 PG2, PG3, PG4 PD0	2, 3, 9 10, 11, 12 22, 17, 16 15, 14, 13 33, 32, 31 23, 42, 48 45, 46, 47 57	Push-pull output (slow)
Unused	PH0 - PH7 PI1 - PH7 PC0	No pin out on 64-pin package	Push-pull output

C 10-pin ribbon cable assignment

The pin assignment for the 10-pin ribbon cable used to connect the bases is given in table 5. Pins 2, 5 and 7 are bridged together. Pins 6 and 8 are also bridged together.

D Raisonance Tools

The STM8S208RB has been supplied with an integrated development environment from Raisonance Tools for writing application code for microcontrollers. It comes with the RKit-STM8 C compiler, linker and assembler and when used with an RLink dongle, can be used to directly upload an application to a chip. It also provides an integrated simulator which allows you to debug the application without running it on a chip. Running the simulator, gives an indication of which registers will be set when the program is run.

Pin	Use
10	Synch_H
9	Synch_L
8	+24V
7	Ground
6	+24V
5	CAN_GND
4	CAN_H
3	CAN_L
2	Ground
1	Reset

Table 5: 10-pin ribbon cable assignment

D.1 Setting Up a New Firmware Project in Ride7

This section describes how to start a new project and upload it to the STM8S208RB. The chip can be programmed through the R-Link dongle.

1. Start Ride7.
2. Click on New Project icon.
3. Choose STM8S208RB chip.
4. Name project and choose the Target Directory.
5. Click on New File icon to create source file.
6. Save file to the choosen Target Directory of the project.
7. Right click on the project name in the project window.
8. Select Add...→Item...
9. Select source file.
10. Select Project→Make Project.
11. Select Debug→Start.

N.B. Due to a bug in the RKit C-compiler, the compiler optimisation level can not be set higher than level 1. Changing the optimisation level may also result in having to recalibrate the DAC serial timing, since the optimisation may remove or alter some of the delay functions (empty loops) being used. To check the optimisation level, from the Project drop down menu, choose Properties, then Optimisation.

D.2 Uploading a new firmware version using RFlasher

To install a new firmware version via the SWIM connector:

1. Start RFlasher.
2. Connect RLink dongle to CAN bus.
3. Click *Erase* button to erase STM8S208RB flash.
4. Click *Upload File* button to choose new firmware to install.
5. Click *Program* button.
6. Click *Reset and Run* button to boot new firmware on chip.

D.3 STM8S Firmware Library Files

ST Microelectronics supply a library of source and header files that can be dropped directly, more or less, into a project to provide utilities for common STM8S applications. These library files can be identified by the prefix "stm8s". For example, the file stm8s_adc2.h defines a group of variables that hold the addresses of the registers associated with the ADC. stm8s_adc2.c contains a group of functions designed to simplify the process of initialising those registers. When the base firmware was initially being developed, ST microelectronics did not provide software support for the CAN peripheral. Instead, Raisonance distributed code (stm8s_can.c, stm8s_can.h) that mimic the STM8S library functions but for the CAN peripheral. Since that time, ST Microelectronics have begun providing software support for CAN and if a new version of the ST firmware library is downloaded it includes files named stm8s_can.c, stm8s_can.h. However, the Raisonance code is much simpler to use and switching to th e ST version would require quite a bit of effort. As a result, it is the Raisonance code that is still used in the base firmware.

A full list of the library files used in BaseFirmware are listed in table 6.

E Clock registers

References

[1] RM0016 Reference Manual, STMicroelectronics, *http://www.st.com/*

[2] UM0470 STM8 SWIM communication protocol and debug module, STMicroelectronics, *http://www.st.com/*

[3] DS2401 Silicon Serial Number Data Sheet, Dallas Semiconductors *http://www.maxim-ic.com/*

Header File	Contents
stm8s_adc2	Prototypes/macros for the ADC2 peripherals
stm8s_can	Provided by Raisonance. Implements CAN functions that mimick the STMicroelectronics' STM8S library, but specifically for the CAN peripheral.
stm8s_clk	All function prototypes and macros for the CLK peripheral.
stm8s_gpio	All function prototypes and macros for the GPIO peripheral.
stm8s_conf	Library configuration.
stm8s_lib	Includes the peripherals header files in the user application.
stm8s	All generic macros and all HW registers definitions and memory mapping.
stm8s_spi	Function prototypes and macros for the SPI peripheral.
stm8s_type	All common data types.

Table 6: STM8S Library Files

Register	Value	Action
CLK_ICKR bit 0	1	Enables 16MHz high speed internal clock
CLK_SWR	0xE1	Selects 16MHz oscillator as master clock
CLK_SWCR	0x00000010	
CLK_CKDIVR	0x00011001	Prescales the clock by setting f_{master}=8MHz and f_{CPU}=2MHz.
CLK_PCKENR2 bit 7	1	
CLK_CCOR	0x00001001	Configures the clock as an output on one of the MCU pins.

Table 7: Clock registers to set

[4] LTC2630 DAC Data Sheet, Linear Technology

[5] Manual AnaGate CAN UNO/DUO/QUATTRO, Analytica, *http://www.anagate.de/*

[6] STMS in application programming using a customized bootloader, ST microelectonics Application Note AN2659, *http://www.st.com/*

[7] IU Cockroft-Walton PMT base talk for electronics workshop, *Gluex-doc-397-v2*.

[8] STM8, ST7 Compiler manual, *http://www.mcu-raisonance.com/*.