Laboratory 08: DOG LCD SPI C Driver for AVR128DB48

Prof. Ken Short

Laboratory 8 Design Tasks

- Design Task 1: Hardware Interface to DOG Module
- Design Task 2: LCD DOG Driver in C
- Design Task 3: Basic Verification of DOG Module Using a Test Program

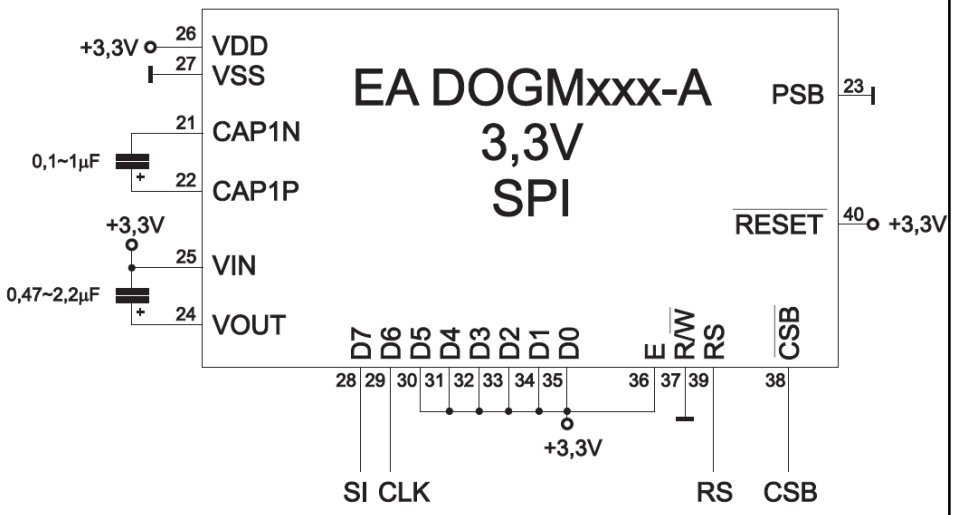
Designing an LCD SPI Hardware Interface

- Designing a hardware interface between the EA DOG 3-line 16-characters per line display.
- For the hardware design use signal assignments below. The GPIO expander hardware is to remain connected to the AVR128DB48, although you will not be using the GPIO in this laboratory. So, the DOG LCD and GPIO expander will be on the same SPI bus.
- PA4 for MOSI
- PA5 for MISO
- PA6 for SCK
- PA7 for /SS
- PC0 for RS of LCD

DOG Module SPI Interface

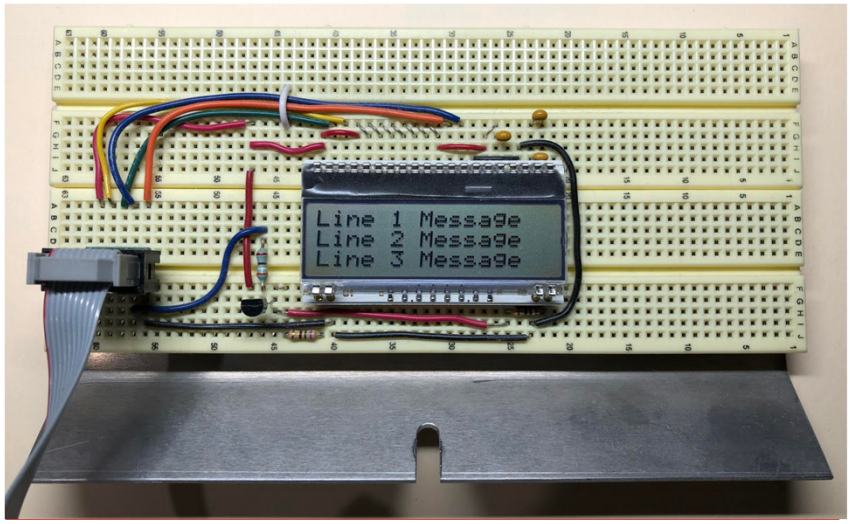
- There are several questions that must be asked regarding hardware design when interfacing two digital ICs
- Is your basic interface circuit logically (topologically) correct?
- Are the logic level voltage and current requirements of each IC met?
- Are the speed constraints of each ICs met?
- Are the pins' software configurations correct?

DOG Module Configured for 3.3V Operation



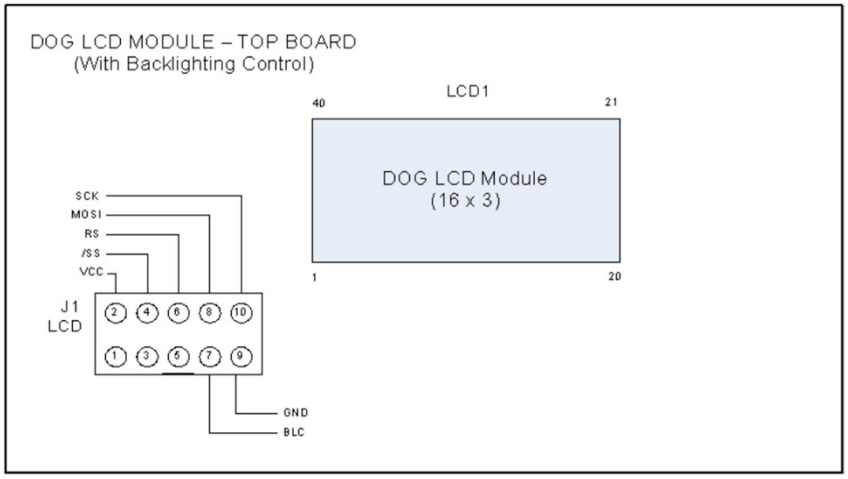
3/23/2022 © Copyright Kenneth Short 2022 5

DOGM163W-A LCD with White Backlight Configured for 3.3V Operation



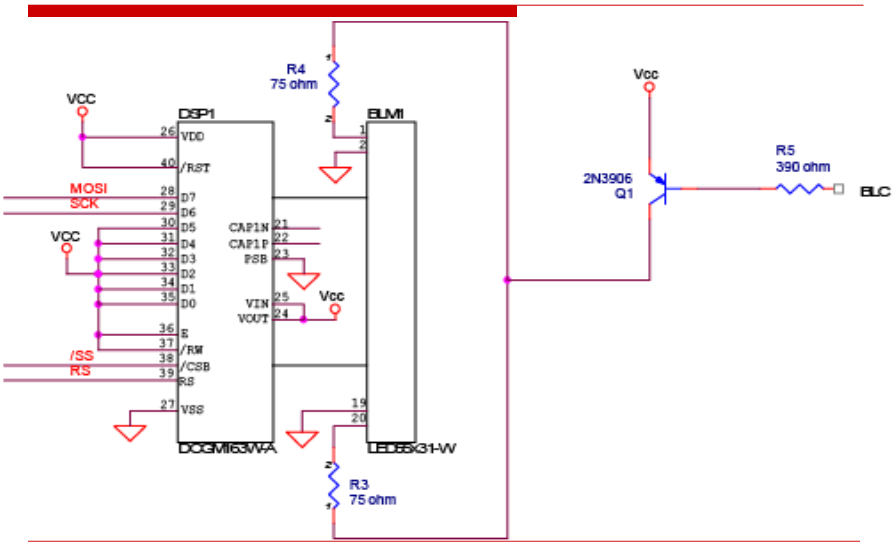
3/23/2022 © Copyright Kenneth Short 2022 6

DOG Module Breadboard Layout



3/23/2022 © Copyright Kenneth Short 2022 7

DOG Module with Backlight



3/23/2022 © Copyright Kenneth Short 2022 8

LCD Power Needs

- LCD: 250 uA @ 3.3V
- LED backlight for LCD: 3 – 80 mA @ 3.3V
- The MIC5353 on the Curiosity board supports a maximum current load of 500 mA.

white EA LED55x31-W	Forwar voltage	Current max.	Limiting resistor	
			@ 3,3 V	@ 5 V
Connected in parallel	3,2 V	60 mA	1,6 ohm	30 ohm
Connected in series	6,4 V	30 mA	-	-

DOG Input Logic Level Requirements

(VDD = 4.5 V ,TA = -35℃ to 85℃)

Symbol	Characteristics	Test Condition	Min.	Typ.	Max.	Unit
VDD	Operating Voltage	-	4.5	-	5.5	V
V_{LCD}	LCD Voltage	V0-Vss	2.7	-	7.0	V
VIN	Power Supply	-	-	-	3.5	V
I_{DD}	Power Supply Current	VDD=5.0V (Use internal booster/follower circuit)	-	240	340	μA
V_{IH1}	Input High Voltage (Except OSC1)	-	0.7 VDD	-	VDD	V
V_{IL1}	Input Low Voltage (Except OSC1)	-	-0.3	-	0.8	V
V_{IH2}	Input High Voltage (OSC1)	-	0.7 VDD	-	VDD	V
V_{IL2}	Input Low Voltage (OSC1)	-	-	-	1.0	V
V_{OH}	Output High Voltage (DB0 - DB7)	I_{OH} = -1.0mA	0.8 VDD	-	VDD	V
V_{OL}	Output Low Voltage (DB0 - DB7)	I_{OL} = 1.0mA	-	-	0.8	V
R_{COM}	Common Resistance	V_{LCD} = 4V, I_d = 0.05mA	-	2	20	KΩ
R_{SEG}	Segment Resistance	V_{LCD} = 4V, I_d = 0.05mA	-	2	30	KΩ
I_{LEAK}	Input Leakage Current	V_{IN} = 0V to VDD	-1	-	1	μA
I_{PUP}	Pull Up MOS Current	VDD = 5V	65	95	125	μA
fosc	Oscillation frequency	VDD = 5V,1/17duty	350	540	1100	kHz

AVR128DB48 I/O Pins Common Characteristics

Symbol	Description	Min.	Typ.	Max.	Units	Conditions
V_{IL}	I/O PORT:					
	• With Schmitt Trigger buffer	—	—	$0.2\times V_{DD}$	V	
	• With I^2C levels	—	—	$0.3\times V_{DD}$	V	
	• With SMBus 3.0 levels	—	—	0.8	V	
	$\overline{\text{RESET}}$ Pin	—	—	$0.2\times V_{DD}$	V	
	TTL level	—	—	0.8	V	
Input High Voltage						
V_{IH}	I/O PORT:					
	• With Schmitt Trigger buffer	$0.8\times V_{DD}$	—	—	V	
	• With I^2C levels	$0.7\times V_{DD}$	—	—	V	
	• With SMBus 3.0 levels	1.35	—	—	V	
	$\overline{\text{RESET}}$ Pin	$0.8\times V_{DD}$	—	—	V	
	TTL level	1.6	—	—	V	

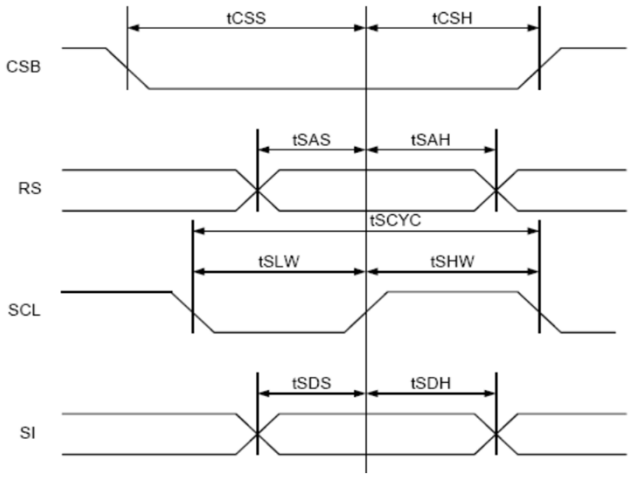
AVR128DB48 I/O Pins Maximum Output Current

Input Leakage Current(2)						
I_{IL}	I/O PORTS(3)	—	<5	—	nA	$GND \le V_{PIN} \le V_{DD}$, pin at high-impedance, 85°C
		—	<5	—	nA	$GND \le V_{PIN}$(5) $\le V_{DD}$, pin at high-impedance, 125°C
	$\overline{\text{RESET}}$ Pin(4)	—	±50	—	nA	$GND \le V_{PIN} \le V_{DD}$, pin at high-impedance, 85°C
Pull-up Current						
I_{PUR}		—	140	—	µA	V_{DD} = 3.0V, V_{PIN} = GND
Output Low Voltage						
V_{OL}	Standard I/O Ports	—	—	0.6	V	I_{OL} = 10 mA, V_{DD} = 3.0V
Output High Voltage						
V_{OH}	Standard I/O Ports	V_{DD}-0.7	—	—	V	I_{OH} = 6 mA, V_{DD} = 3.0V
Pin Capacitance						
C_{IO}	Op amp output	—	9	—	pF	
	V_{REF}pin	—	7	—	pF	
	XTAL pins	—	4	—	pF	
	Other pins	—	4	—	pF	

Logic Level Verification

- The logic level check list must be used to verify the voltage and current logic level requirements for each pin.
- In general there are four cases that must be verified. Use the logic level compatibility check list that you have used before.

DOG Module Timing Waveforms



DOG Module Timing Specifications

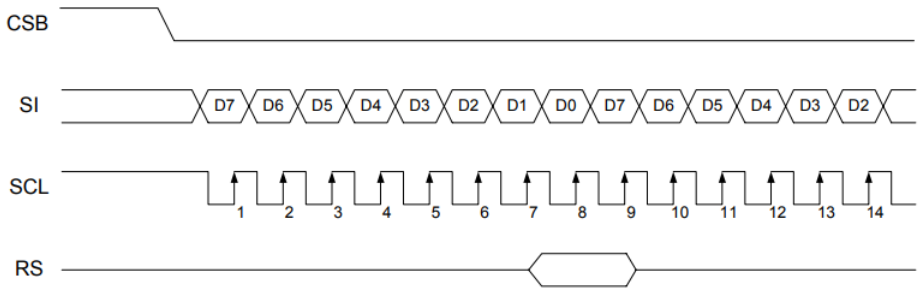
(Ta = 25°C)

Item	Signal	Symbol	Condition	VDD=2.7 to 4.5V Rating		VDD=4.5 to 5.5V Rating		Units
				Min.	Max.	Min.	Max.	
Serial Clock Period	SCL	t_{SCYC}	—	200	-	100	-	ns
SCL "H" pulse width		t_{SHW}		20	-	20	-	
SCL "L" pulse width		t_{SLW}		160	-	120	-	
Address setup time	RS	t_{SAS}	—	10	-	10	-	ns
Address hold time		t_{SAH}		250	-	150	-	
Data setup time	SI	t_{SDS}	—	10	-	10	-	ns
Data hold time		t_{SDH}		10	-	20	-	
CS-SCL time	CS	t_{CSS}	—	20	-	20	-	ns
		t_{CSH}		350	-	200	-	

*1 All timing is specified using 20% and 80% of V_{DD} as the standard.

3/23/2022 © Copyright Kenneth Short 2022 15

SPI Timing Waveform for ST7036



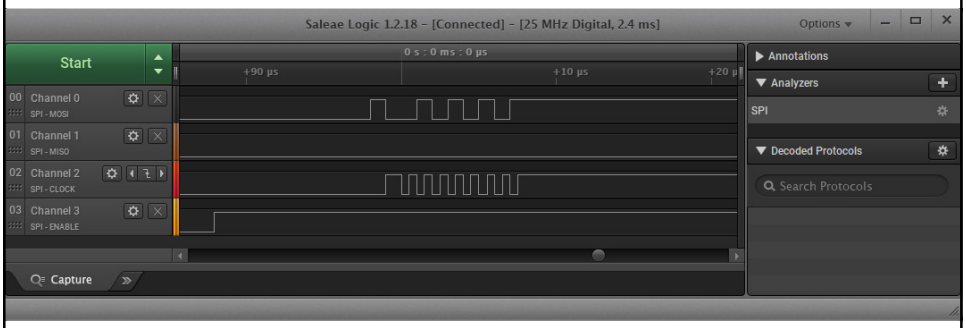
3/23/2022 © Copyright Kenneth Short 2022 16

Main subroutines Provided in C Driver and lcd_dog_asm_driver_m324a.inc

- init_lcd_dog
  Initializes ST7036

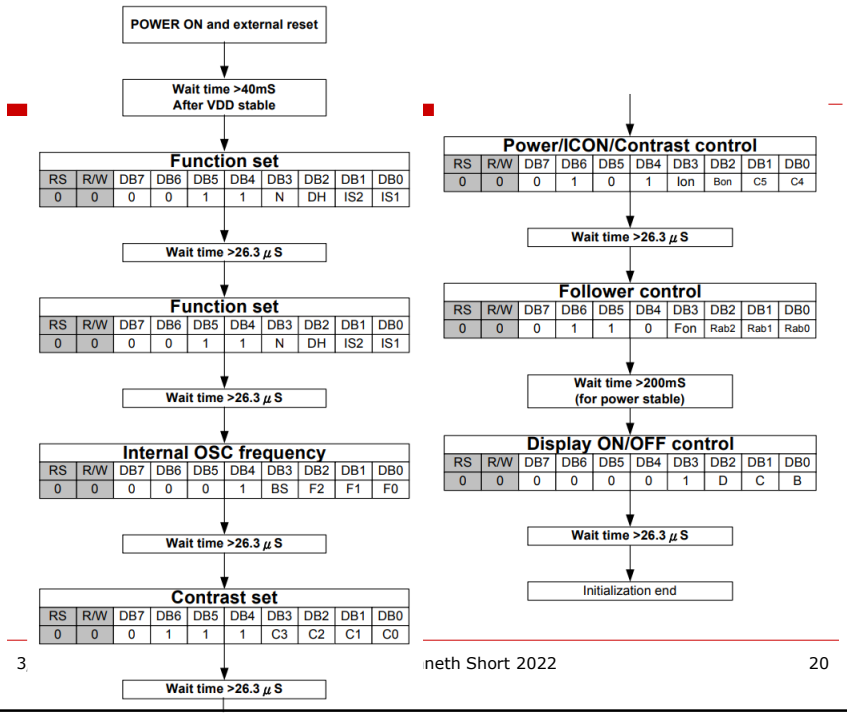
- update_lcd_dog
  Transfers contents buffer to ST7036's DDRAM

Dummy Byte Sent to Set SPI0 IF



init_lcd_dog Function

- The steps that must be carried out to initialize the DOG LCD are shown in the flowchart that follows.
- For the 26.3 us delay simply use the _delay_us delay with a parameter of 30 us.
- You should use the same command argument values sent in the lcd_spi_transmit_CMD calls.
- The C code provided also includes a command to clear the display as part of the initialization.



Instructions - ST7036 Execution Times

➢ **instruction table at "Normal mode"**
(when "EXT" option pin connect to VDD, the instruction set follow below table)

Instruction	Instruction Code										Description	Instruction Execution Time		
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		OSC= 380kHz	OSC= 540kHz	OSC= 700kHz
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM. and set DDRAM address to "00H" from AC	1.08 ms	0.76 ms	0.59 ms
Return Home	0	0	0	0	0	0	0	0	1	X	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.08 ms	0.76 ms	0.59 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	26.3 µs	18.5 µs	14.3 µs
Display ON/OFF	0	0	0	0	0	0	1	D	C	B	D=1:entire display on C=1:cursor on B=1:cursor position on	26.3 µs	18.5 µs	14.3 µs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	X	X	S/C and R/L: Set cursor moving and display shift control bit, and the direction, without changing DDRAM data.	26.3 µs	18.5 µs	14.3 µs

Instructions (cont.) - ST7036 Execution Times

Function Set	0	0	0	0	1	DL	N	X	X	X	DL: interface data is 8/4 bits N: number of line is 2/1	26.3 µs	18.5 µs	14.3 µs
Set CGRAM	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter	26.3 µs	18.5 µs	14.3 µs
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter	26.3 µs	18.5 µs	14.3 µs
Read Busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0	0	0
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM)	26.3 µs	18.5 µs	14.3 µs
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM)	26.3 µs	18.5 µs	14.3 µs

DDRAM Address

1	2	3	4	5	6		14	15	16
00	01	02	03	04	05	.....	0D	0E	0F
10	11	12	13	14	15	.....	1D	1E	1F
20	21	22	23	24	25	.....	2D	2E	2F

3/23/2022 © Copyright Kenneth Short 2022 23

How to Approach Porting Either the SAML21J18B C or Assembly Software to AVR128DB48 C

- Porting software and/or reconfiguring software for different hardware requires a methodical approach.
- You have files to port. One is the test program and the other is the LCD driver.
- Since you need the LCD driver to run the test program, the LCD driver is the logical place to start.
- SPI0 and its pin assignments were specified and you have the C program from Laboratory 8 for configuring SPI0 to use as a starting point.

3/23/2022 © Copyright Kenneth Short 2022 24

Designing a DOG LCD Software Driver

- The starting point for writing the LCD driver in C is the code from Laboratory 8 using SPI0 for SPI communications and either the C driver for the SAML21J18B from ESE381 S20:
  - C_driver_s20_SAML21J18B_main.c
- or the assembly language driver and basic test software from ESE280:
  - lcd_dog_asm_driver_m324a.inc file
  - DOG_LCD_BasicTest Program
- If you use the assembly language files to port from, you must port the assembly language driver and basic test program to a single file C program.
- Use the subroutine names from the assembly language program or function names from the provided C program as the function names in your C program.

How to Approach Porting Either the SAML21J18B C or Assembly Software to AVR128DB48 C

- The program from Laboratory 8 was written to drive a MCP23S17 SPI slave, so you must determine the requirements (max. speed, CPOL, and CPHA) for the DOG LCD's ST7036 driver IC, since it will be the slave in this design.
- Looking at the subroutines and functions in the drivers provided you can see that the fundamental operations come down to writing commands and data to the DOG LCD.
- So, you can start with the lcd_spi_transmit_CMD subroutine.

Porting From Assembly to C

- The easiest way to do this is to first rewrite each SAML21J18B C function or assembly subroutine as a AVR128DB48 C function.
- Study each of the functions or subroutines to see what it does functionally. Write a similarly named AVR128DB48 C routine that provides the equivalent functionality.
- Studying the existing code and writing your new code will give you familiarity with the details of the design and you might wish, subsequently, to rewrite your new code to improve it.

3/23/2022 © Copyright Kenneth Short 2022 27

Functions in Module lcd in file lcd_dog_iar_driver.asm

Public functions – can be called from outside the module

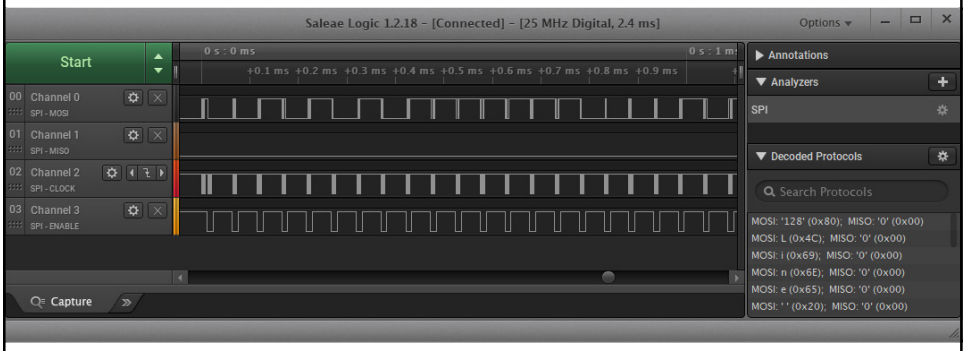
- init_lcd_dog
- update_lcd_dog

Local functions – can only be called from within the module

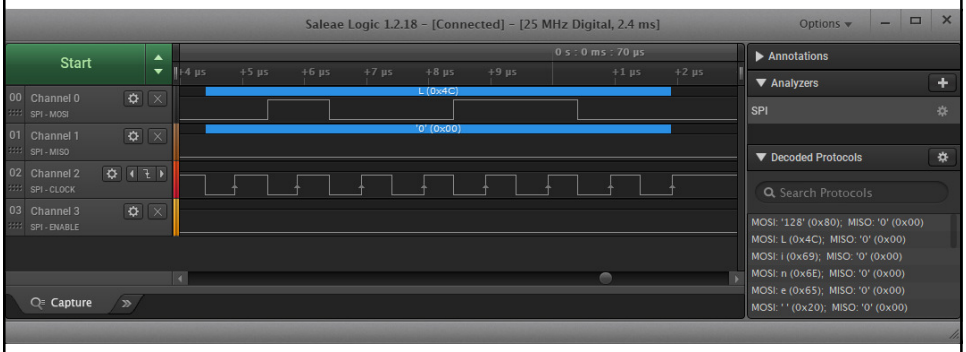
- delay_30uS
- v_delay
- delay_40mS
- init_spi_lcd
- and many others

3/23/2022 © Copyright Kenneth Short 2022 28

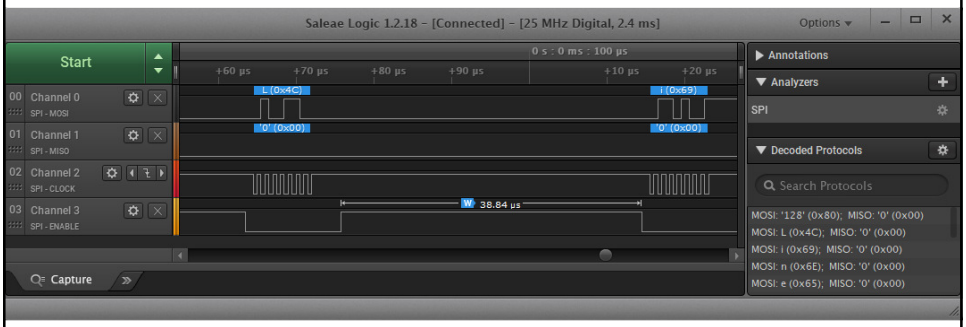
Buffer Contents Being Sent to LCD



Character L Being Transmitted with the Correct SPI Format for DOG LCD



Time Between Commands – Ideally > 26.3 us



Debugging Without the LCD and Logic Analyzer

- Even without an LCD and logic analyzer you can do a lot of debugging just with the Curiosity board.
- "Step into" a function the first time you use it. After you have debugged it, "step over" the function when you subsequently encounter it.
- Use the I/O window to verify that your software is properly configuring SPI0.
- Put you variables in a watch window to verify that the computational portion of your software is working properly.

Using an n + 1 String Buffer to Hold LCD Data Image

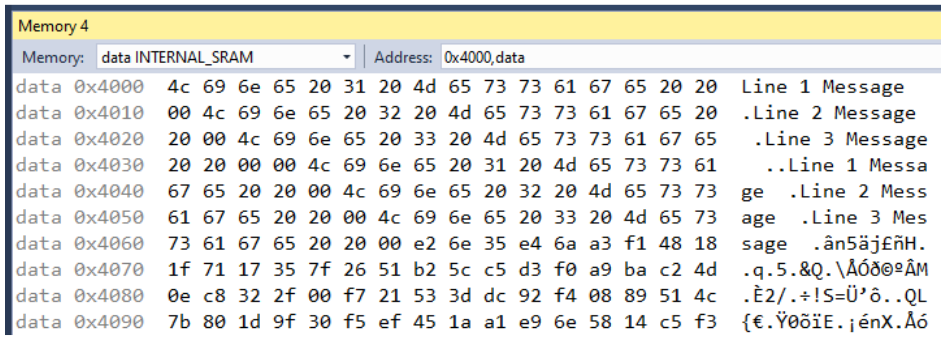
```
// Display buffer for DOG LCD using sprintf()
char dsp_buff1[17];
char dsp_buff2[17];
char dsp_buff3[17];
```


dsp_buff1 in Watch Window

Watch 1

Name	Value	Type
dsp_buff1	0x4034	char[17]{data}@0x4034
[0]	0x4c	char{data}@0x4034
[1]	0x69	char{data}@0x4035
[2]	0x6e	char{data}@0x4036
[3]	0x65	char{data}@0x4037
[4]	0x20	char{data}@0x4038
[5]	0x31	char{data}@0x4039
[6]	0x20	char{data}@0x403a
[7]	0x4d	char{data}@0x403b
[8]	0x65	char{data}@0x403c
[9]	0x73	char{data}@0x403d
[10]	0x73	char{data}@0x403e
[11]	0x61	char{data}@0x403f
[12]	0x67	char{data}@0x4040
[13]	0x65	char{data}@0x4041
[14]	0x20	char{data}@0x4042
[15]	0x20	char{data}@0x4043
[16]	0x00	char{data}@0x4044
dsp_buff2	0x4045	char[17]{data}@0x4045
[0]	0x4c	char{data}@0x4045
[1]	0x69	char{data}@0x4046
[2]	0x6e	char{data}@0x4047

dsp_buff 1, 2, and 3 in Memory Window with Contents Decoded in ASCII



String Print Formatted Function sprintf()

- sprintf – string print formatted – works in exactly the same way as printf, but returns formatted output in a string buffer, rather than printing the string.
- All the format strings work with sprintf. The only real difference is that it returns a string.

```
sprintf(buffer, "Temp = %d\n\r", temp);
```

- Requires <stdio.h>
- Other "safe" variations exist: snprintf, sprint_s

Filling the Buffer in the AVR128DB48 Using sprintf

```
sprintf(dsp_buff1, "Line 1 Message  ");
sprintf(dsp_buff2, "Line 2 Message  ");
sprintf(dsp_buff3, "Line 3 Message  ");
update_lcd_dog();
```


Other Commonly Used String Functions – Require <string.h>

- strcat - concatenate two strings
- strchr - string scanning operation
- strcmp - compare two strings
- strcpy - copy a string
- strncat - concatenate two strings – "safe version"
- strcpy_s - copy a string – "safe version"

