# Driving an LCD display over SPI

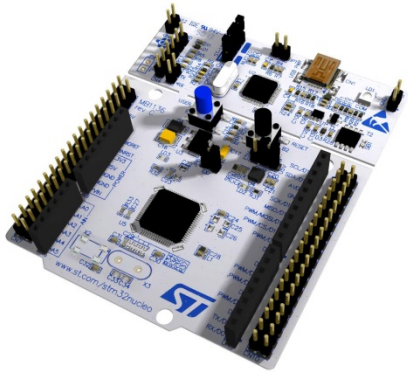
10.10.2016

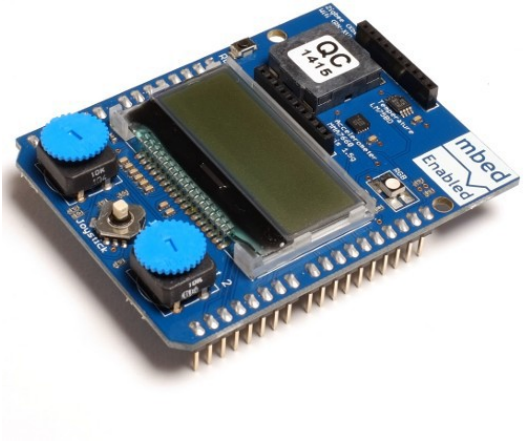
Programming Microcontrollers

C:\Keil_v5\ARM\PACK\Keil\STM32F1xx_DFP\1.1.0\Device\Include\stm32f10x.h

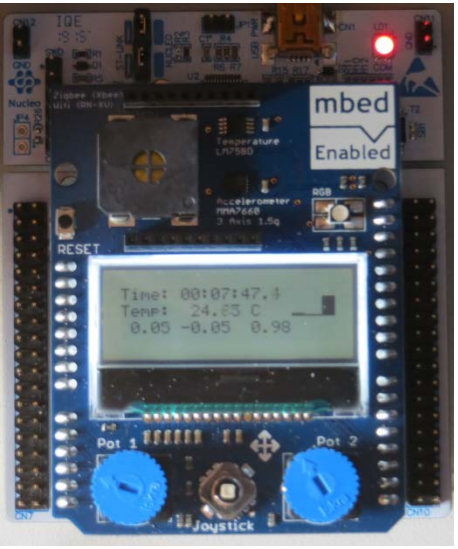
LCD Liquid Cristal Display
SPI Serial Peripheral Interface

# Why drive a display?





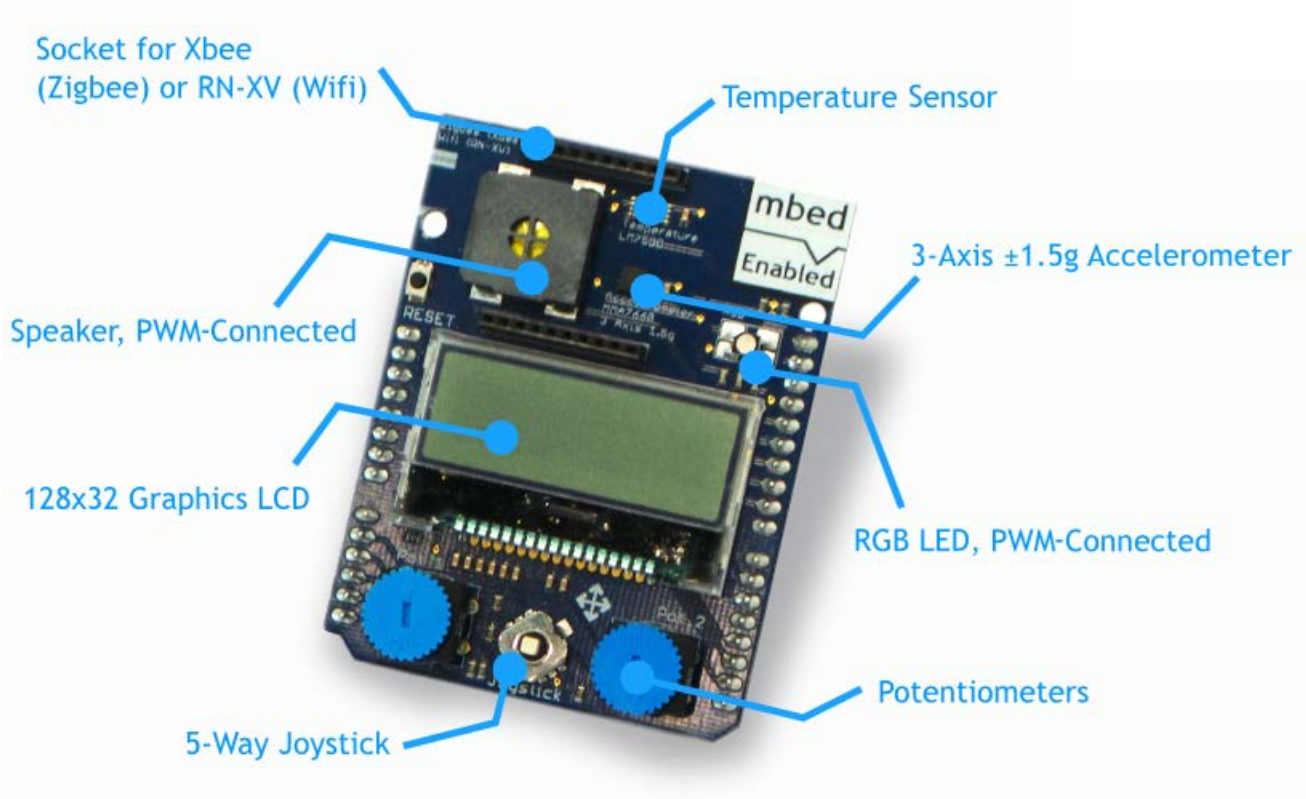

full sunlight




- a small embedded controller has no intrinsic display capabilites
- a display might be useful during development and debugging

# mbed 016-01 application shield by Keil

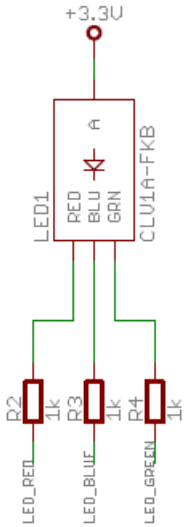
Interfaced with Arduino Uno PinOut




https://developer.mbed.org/

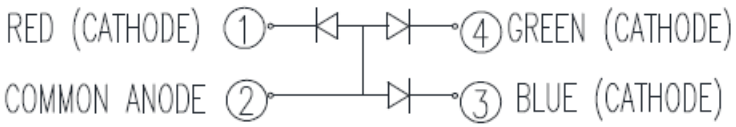
# RGB LED

Common Anode!

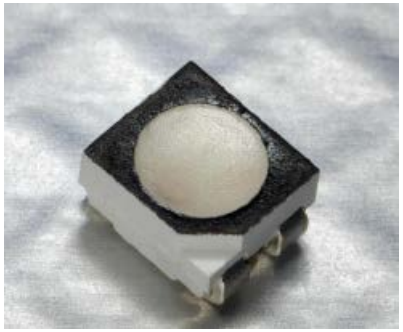



LED: R G

| LED | R | G | B |
|---|---|---|---|
| Shield: | D5 | D9 | D8 |
| Port | PA9 | PC7 | PB4 |



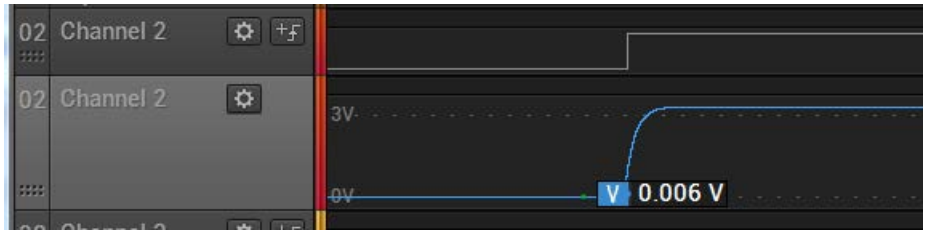

**Cree® PLCC4 3-in-1 SMD LED CLV1A-FKB**



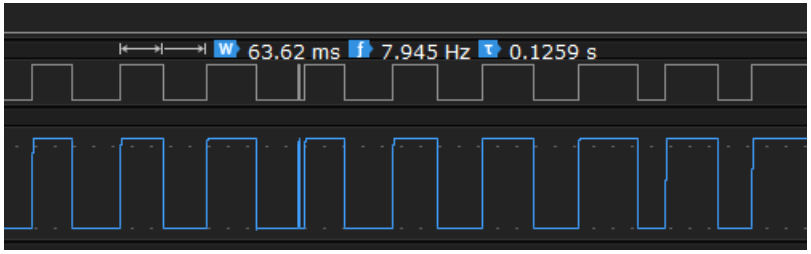
!! LED R does not work on NUCLEO STM32F103, electrically ok !!

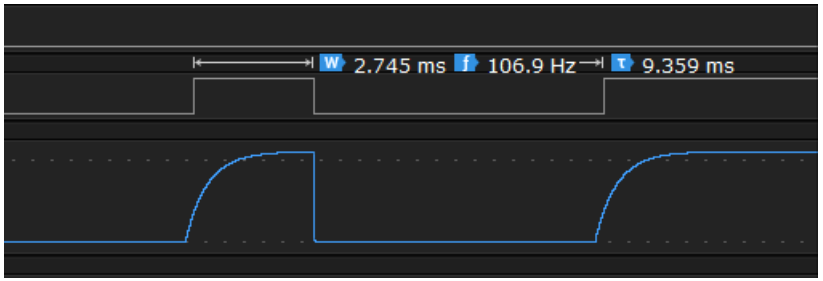
# Logic Analyzer

➔ Analyse hardware signals, both digital and analog !



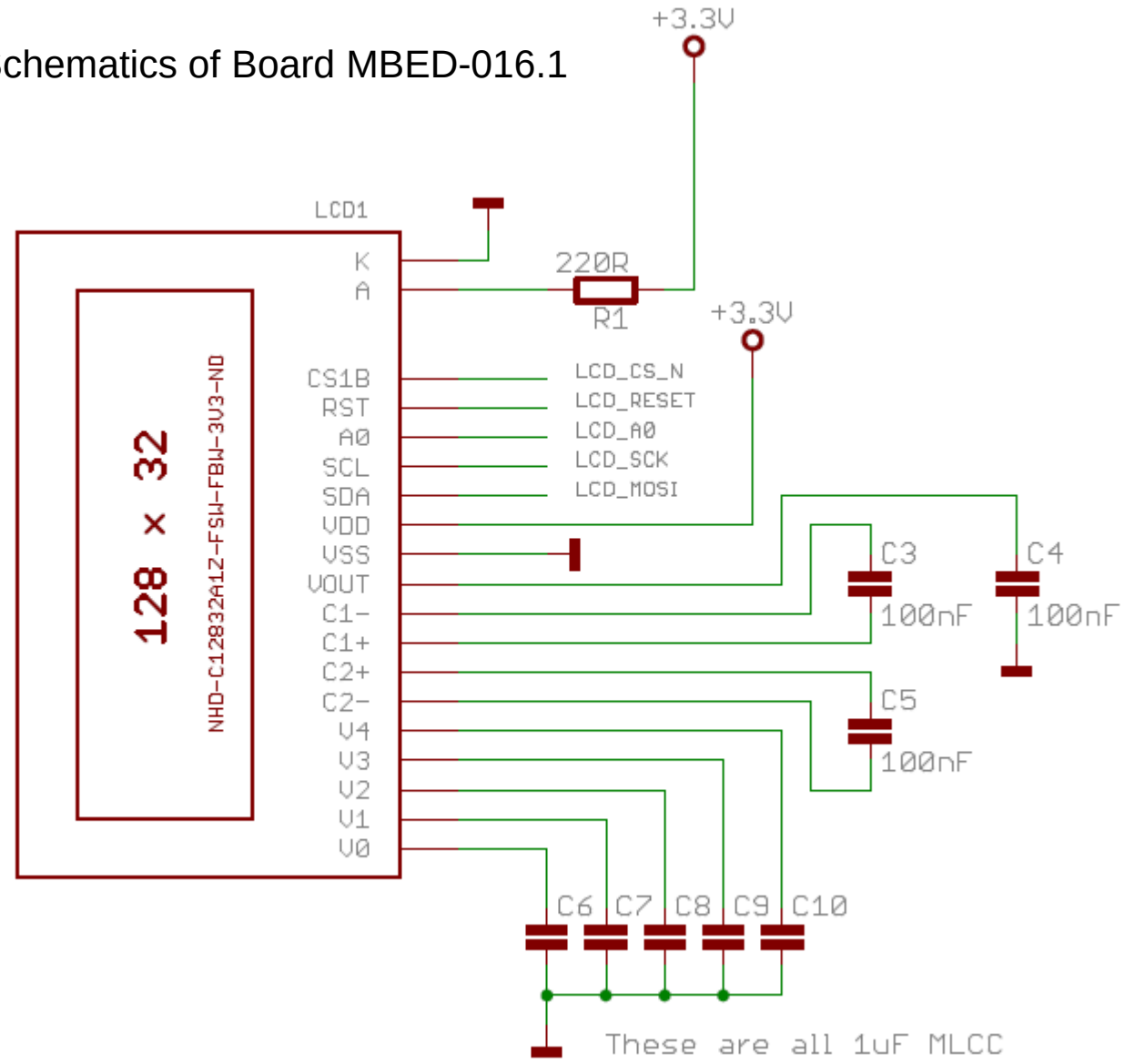

User button on PC13 Nucleo



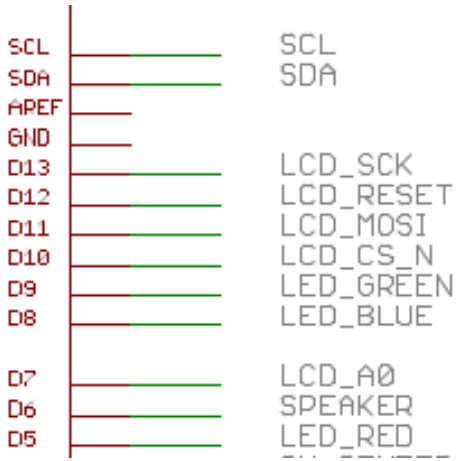



# Locate LCD pins

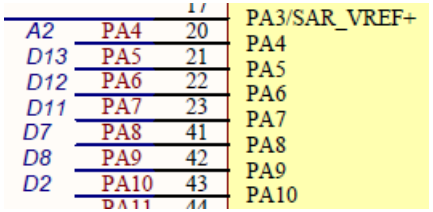
## Schematics of Board MBED-016.1



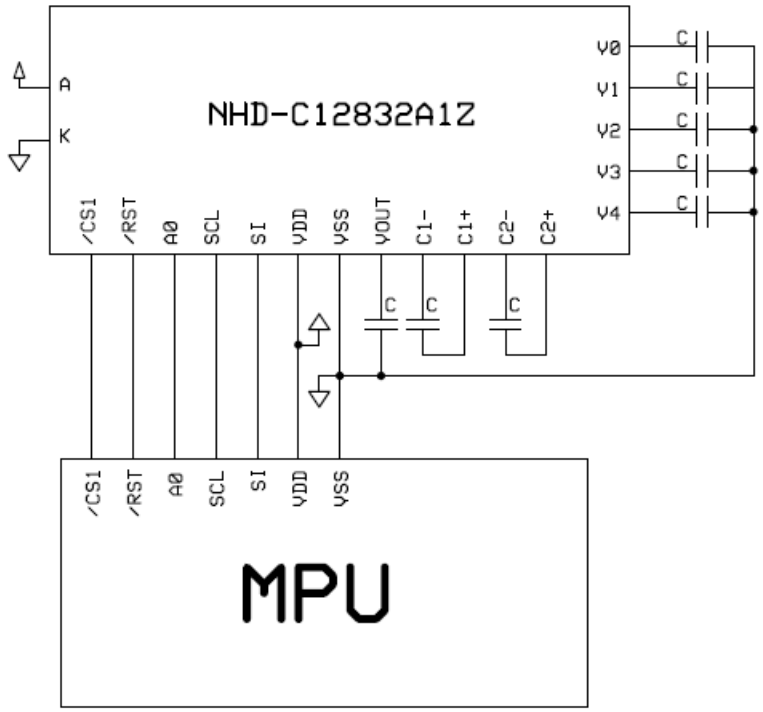

## Arduino R3-pins




## NUCLEO-64 pins



# Connections to LCD (display data sheet)




A0 = high, display data
A0 = low, command data

# Display Controller pin functions
(ST7565 page 9/56)

## ST7565

**System Bus Connection Pins**

| Pin Name | I/O | Function | No. of Pins |
|---|---|---|---|
| D5 to D0<br>D6 (SCL)<br>D7 (SI) | I/O | This is an 8-bit bi-directional data bus that connects to an 8-bit or 16-bit standard MPU data bus.<br>When the serial interface is selected (P/S = "L") :<br>D7 : serial data input (SI) ; D6 : the serial clock input (SCL).<br>D0 to D5 are set to high impedance.<br>When the chip select is not active, D0 to D7 are set to high impedance. | 8 |
| A0 | I | This is connect to the least significant bit of the normal MPU address bus, and it determines whether the data bits are data or a command.<br>A0 = "H": Indicates that D0 to D7 are display data.<br>A0 = "L": Indicates that D0 to D7 are control data. | 1 |
| RES | I | When /RES is set to "L," the settings are initialized.<br><br>The reset operation is performed by the /RES signal level. | 1 |
| CS1<br>CS2 | I | This is the chip select signal. When /CS1 = "L" and CS2 = "H," then the chip select becomes active, and data/command I/O is enabled. | 2 |

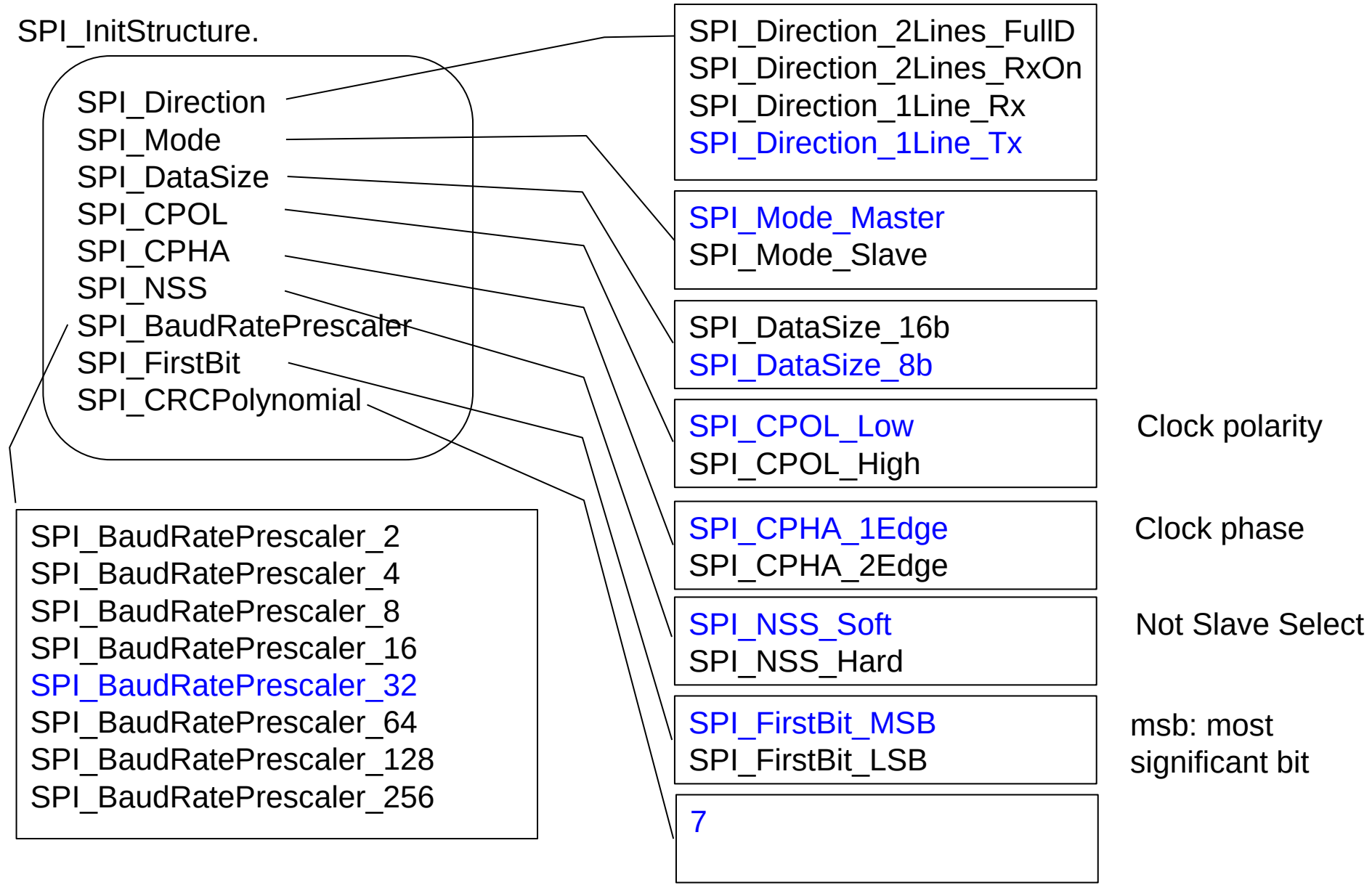
# Configuration of SPI pins (uC datasheet)



Table 5. Medium-density STM32F103xx pin definitions (continued)

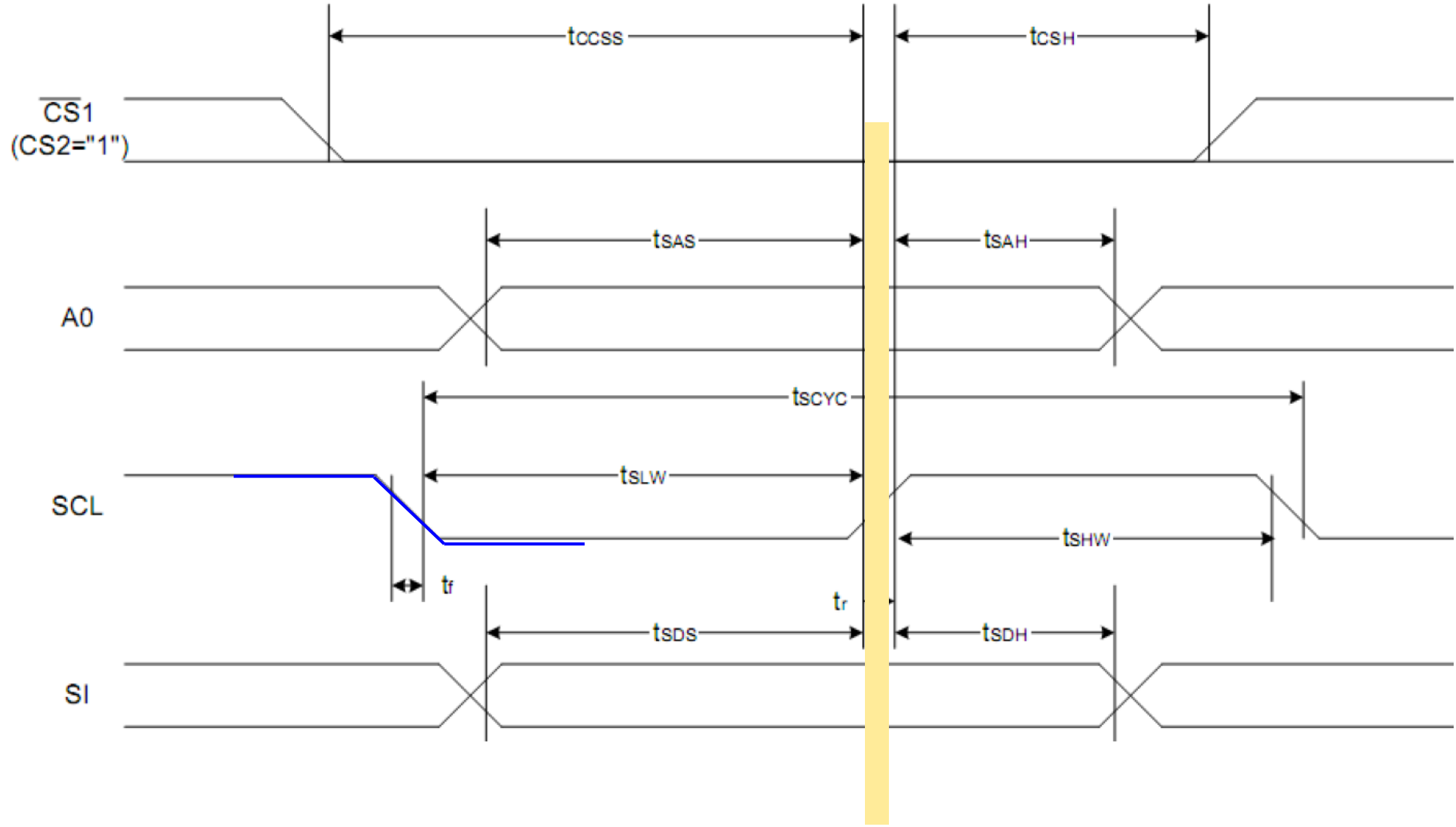
| Pins | | | | | | | Pin name | Type(1) | I / O Level(2) | Main function(3) (after reset) | Alternate functions(4) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LFBGA100 | UFBG100 | LQFP48/UFQFPN48 | TFBGA64 | LQFP64 | LQFP100 | VFQFPN36 | | | | | Default | Remap |
| G3 | M3 | 14 | H3 | 20 | 29 | 11 | PA4 | I/O | | PA4 | SPI1_NSS(8)/ USART2_CK(8)/ ADC12_IN4 | |
| H3 | K4 | 15 | F4 | 21 | 30 | 12 | PA5 | I/O | | PA5 | SPI1_SCK(8)/ ADC12_IN5 | |
| J3 | L4 | 16 | G4 | 22 | 31 | 13 | PA6 | I/O | | PA6 | SPI1_MISO(8)/ ADC12_IN6/ TIM3_CH1(8) | TIM1_BKIN |
| K3 | M4 | 17 | H4 | 23 | 32 | 14 | PA7 | I/O | | PA7 | SPI1_MOSI(8)/ ADC12_IN7/ TIM3_CH2(8) | TIM1_CH1N |
| G4 | K5 | - | H5 | 24 | 33 | | PC4 | I/O | | PC4 | ADC12_IN14 | |

# SPI Configuration (1)
## STM32Fx (stm32f10x_spi.h)



# LCD timing diagram

**The 4-line SPI Interface**




sampling point for data

# ST7565 Serial interface

figure 1 page 13/56

## The Serial Interface

When the serial interface has been selected (P/S = “L”) then when the chip is in active state (/CS1 = “L” and CS2 = “H”) the serial data input (SI) and the serial clock input (SCL) can be received. The serial data is read from the serial data input pin in the rising edge of the serial clocks D7, D6 through D0, in this order. This data is converted to 8 bits parallel data in the rising edge of the eighth serial clock for the processing.

The A0 input is used to determine whether or the serial data input is display data or command data; when A0 = “H”, the data is display data, and when A0 = “L” then the data is command data. The A0 input is read and used for detection every 8th rising edge of the serial clock after the chip becomes active. Figure 1 is a serial interface signal chart.

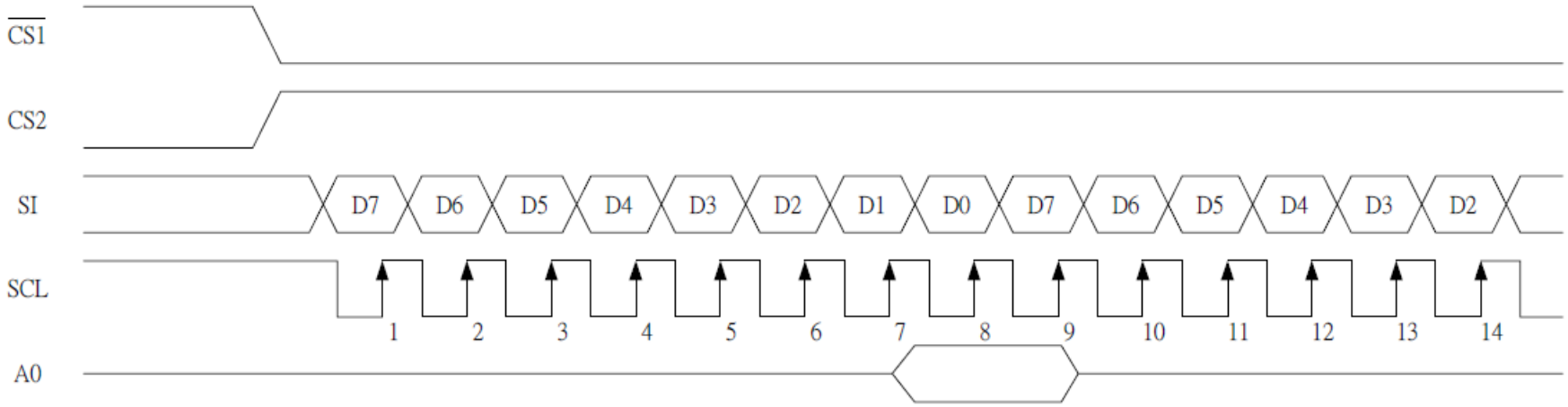
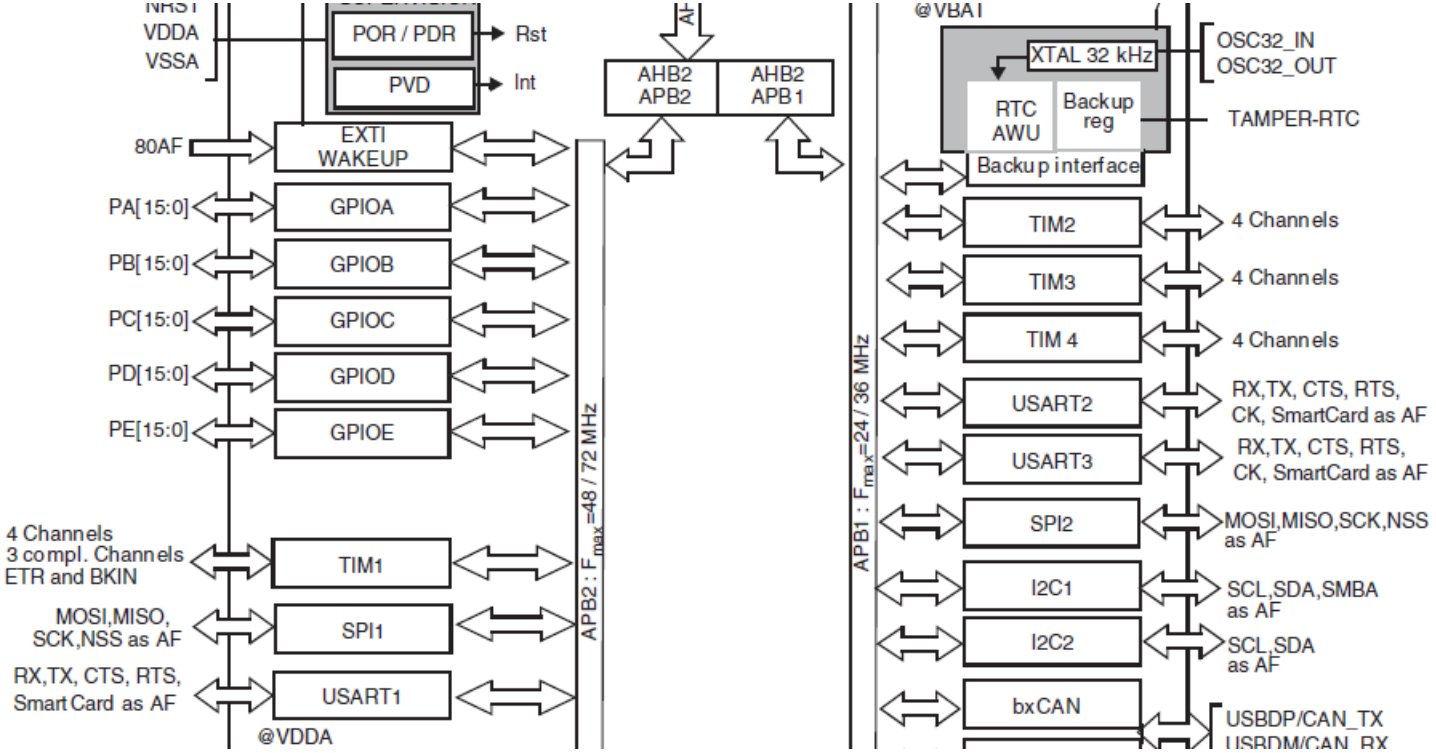


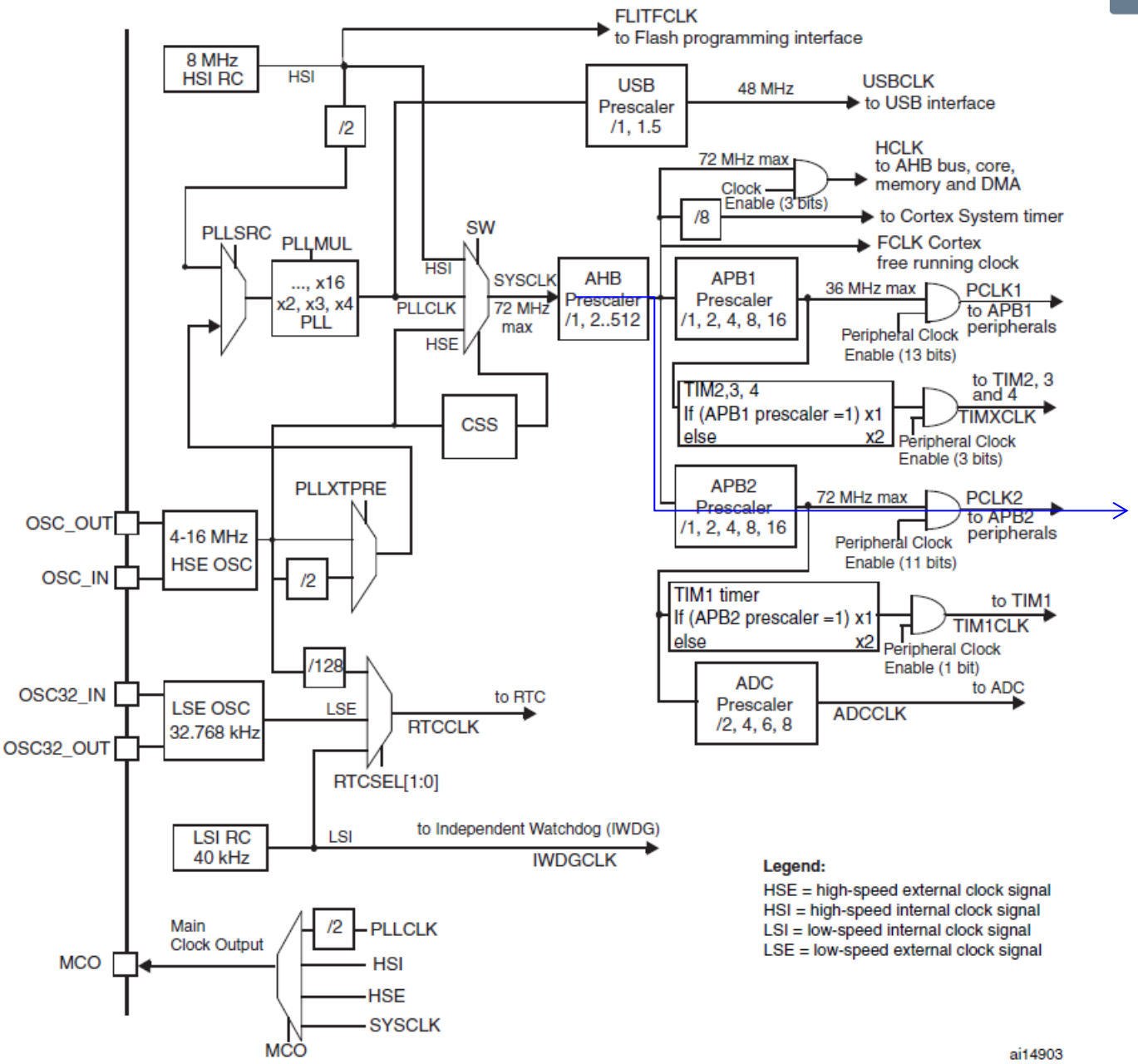

Figure 1

# SPI1 is on which Peripheral bus?



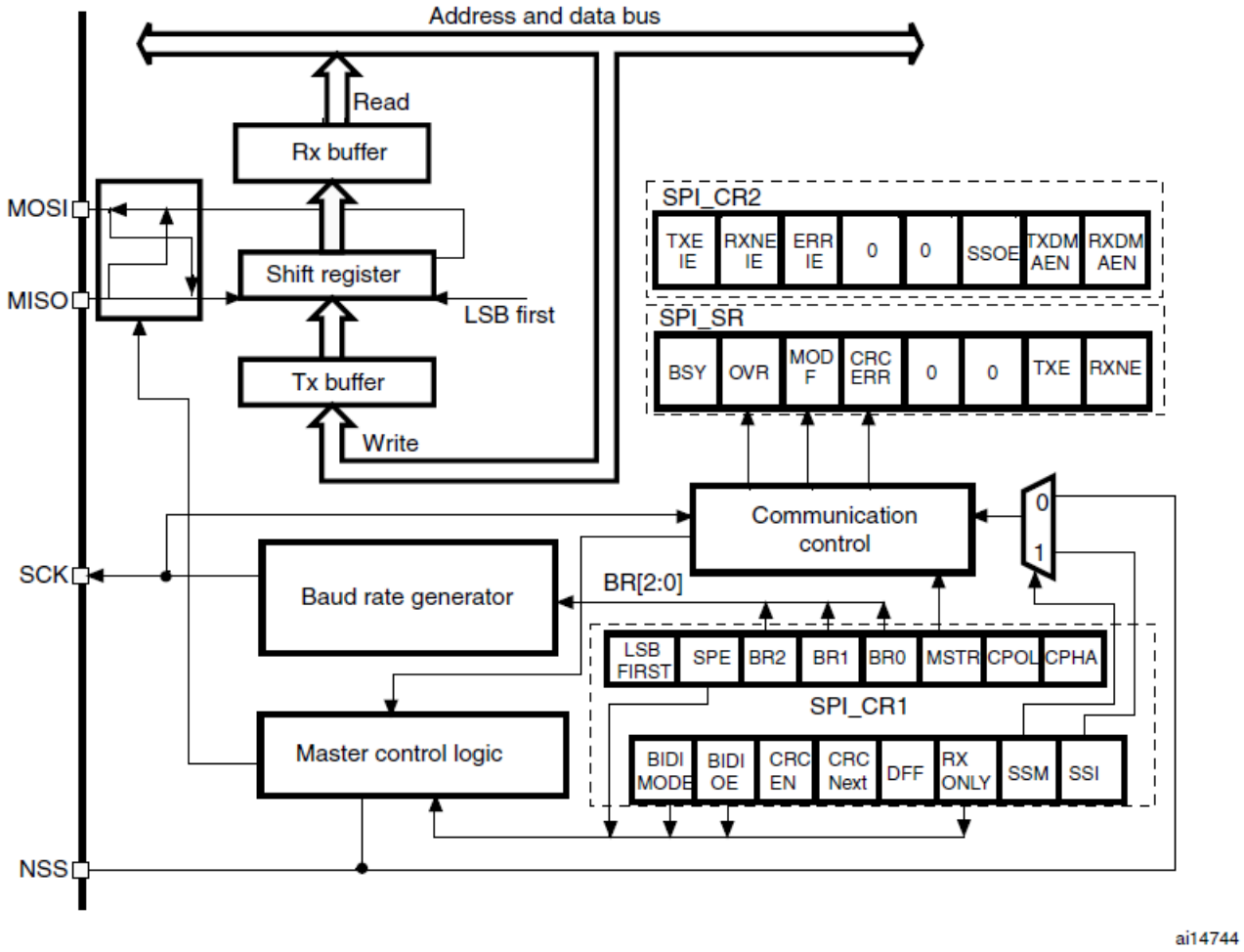

➔Datasheet of STM32F103

# ...running at ?? MHz



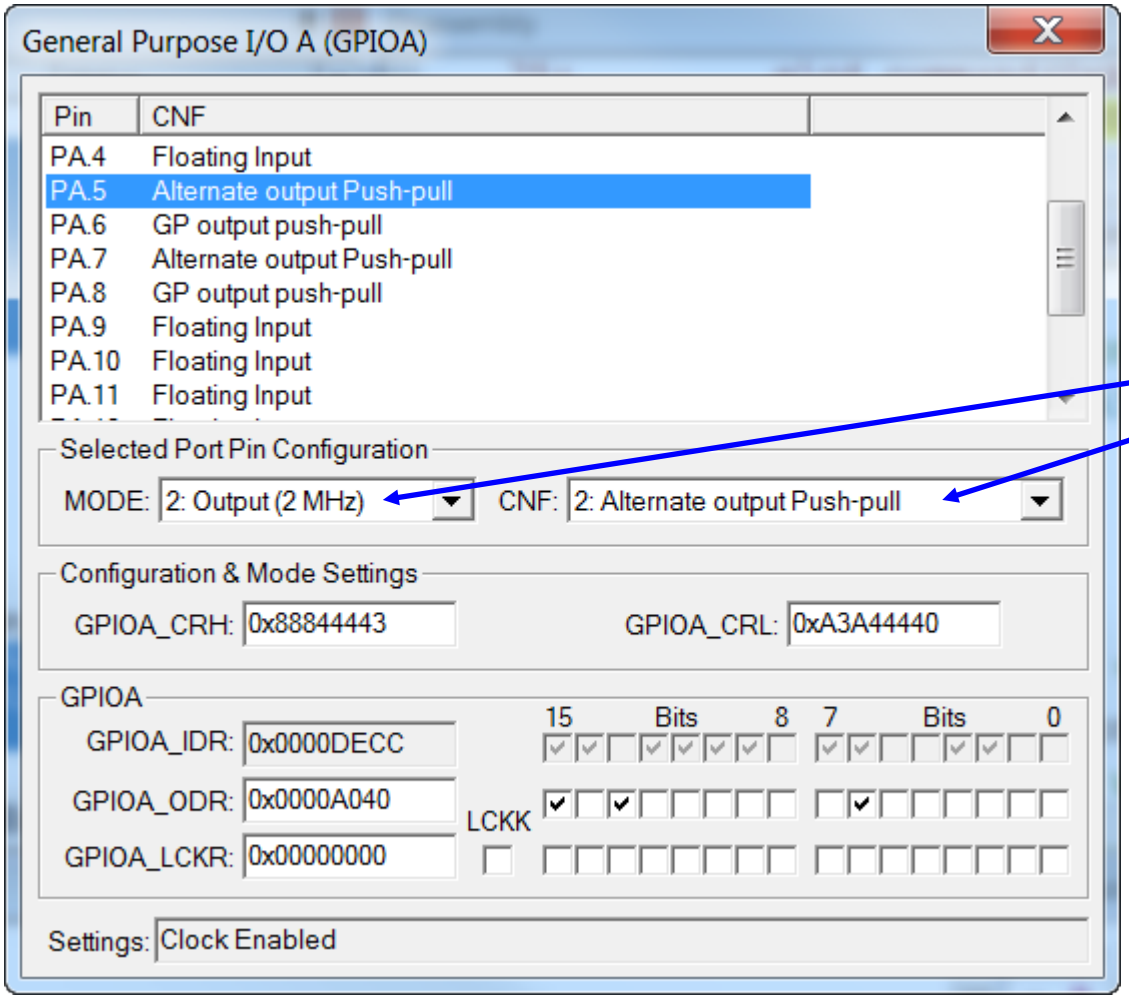

datasheet
STM32F103
Clock tree

# SPI block diagram

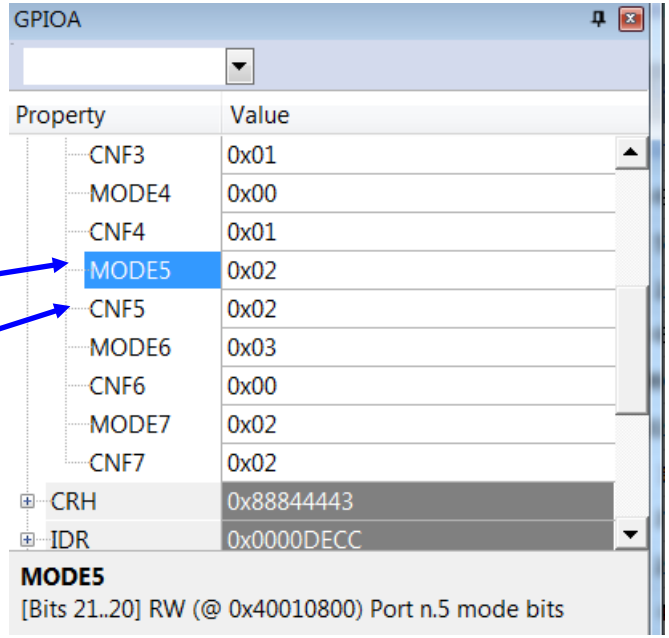



source: RM0008 STM32F10x reference manual

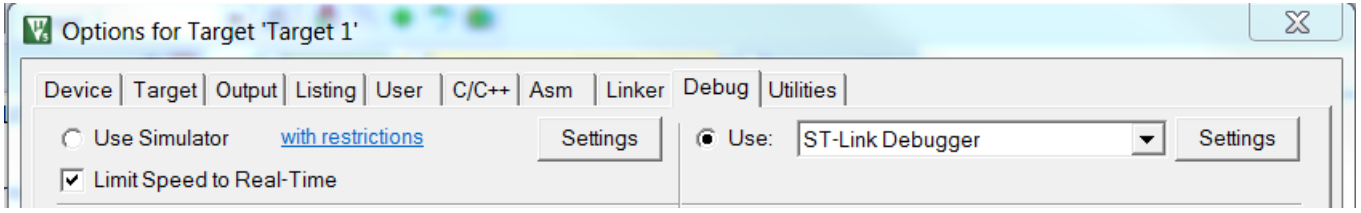
# Setup of GPIO PA5 und PA7 as SPI



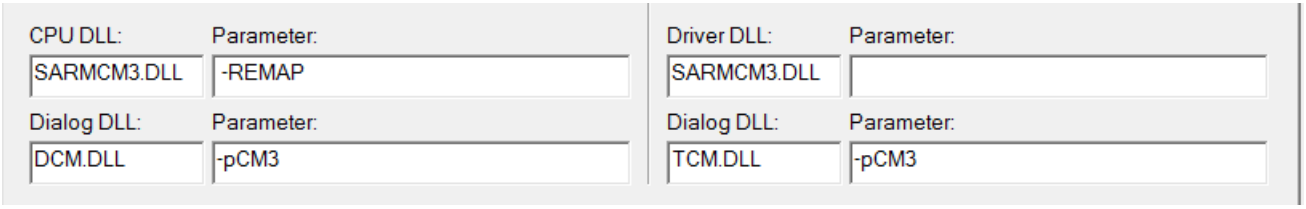

Äquivalente Systemviewerdarstellung



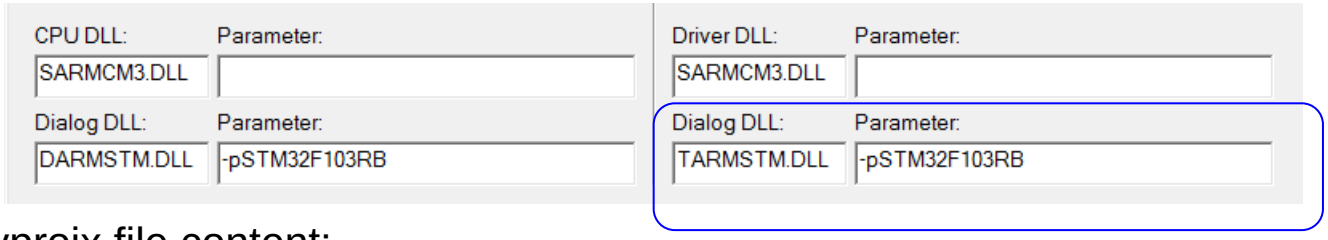
# If Keil uVision does not simulate the peripherals in STM32F103RB chip:




Debugging with standard Cortex-M3 Systemviewer:




**Debugging peripherals in detail using STM32F10x uC:**




*.uvprojx file content:

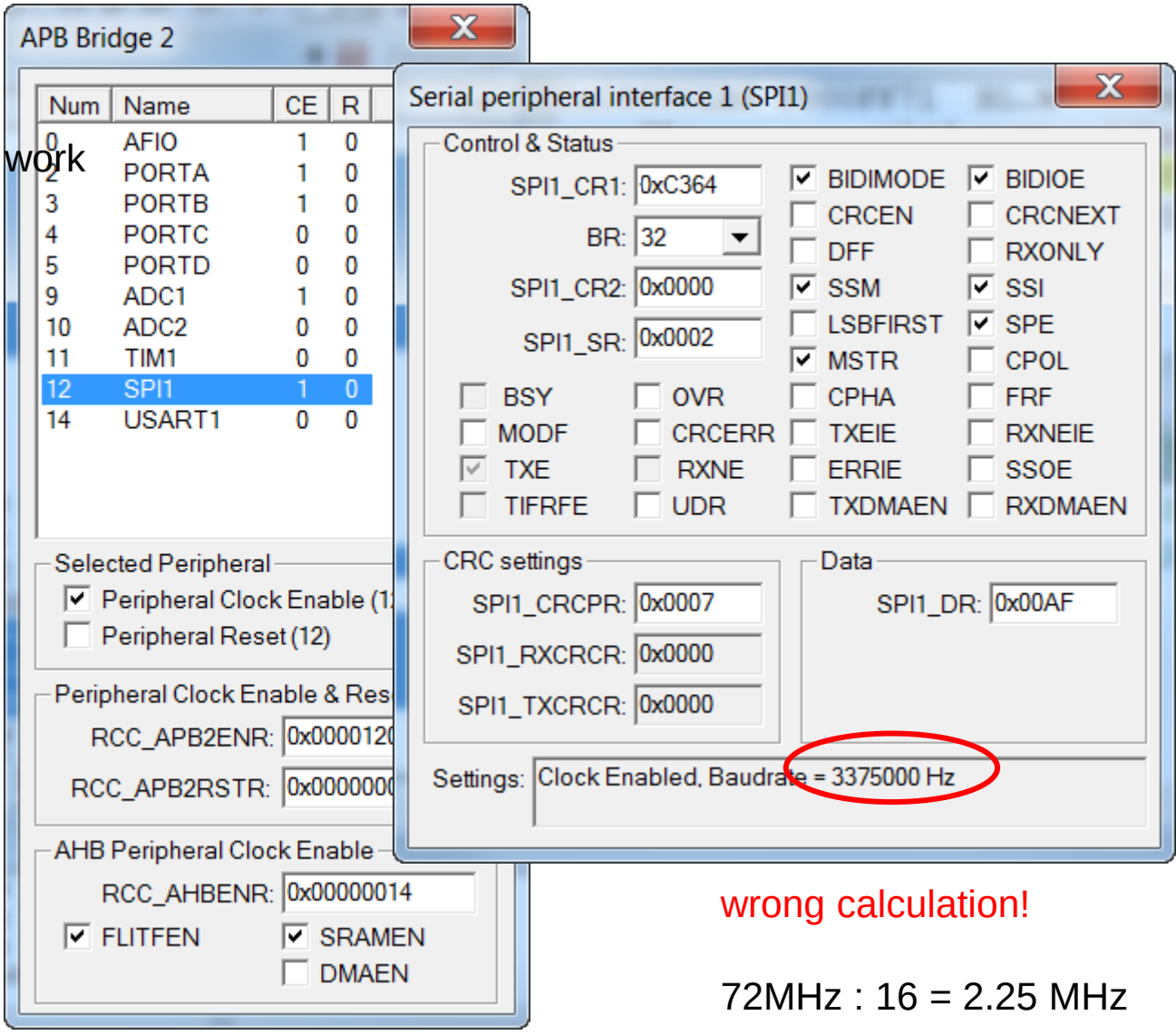
```
110:        <DllOption>
111:          <SimDllName>SARMCM3.DLL</SimDllName>
112:          <SimDllArguments> -REMAP</SimDllArguments>
113:          <SimDlgDll>DCM.DLL</SimDlgDll>
114:          <SimDlgDllArguments>-pCM3</SimDlgDllArguments>
115:          <TargetDllName>SARMCM3.DLL</TargetDllName>
116:          <TargetDllArguments></TargetDllArguments>
117:          <TargetDlgDll>TCM.DLL</TargetDlgDll>
118:          <TargetDlgDllArguments>-pCM3</TargetDlgDllArgumen
119:        </DllOption>
120:        <DebugOption>
```

```
110:        <DllOption>
111:          <SimDllName>SARMCM3.DLL</SimDllName>
112:          <SimDllArguments></SimDllArguments>
113:          <SimDlgDll>DARMSTM.DLL</SimDlgDll>
114:          <SimDlgDllArguments>-pSTM32F103RB</SimDlgDllArgum
115:          <TargetDllName>SARMCM3.DLL</TargetDllName>
116:          <TargetDllArguments></TargetDllArguments>
117:          <TargetDlgDll>TARMSTM.DLL</TargetDlgDll>
118:          <TargetDlgDllArguments>-pSTM32F103RB</TargetDlgDl
119:        </DllOption>
120:        <DebugOption>
```

Setting up the LCD Display to work

- define RCC_
- define GPIO_
- define SPI_

- Set uVision defines

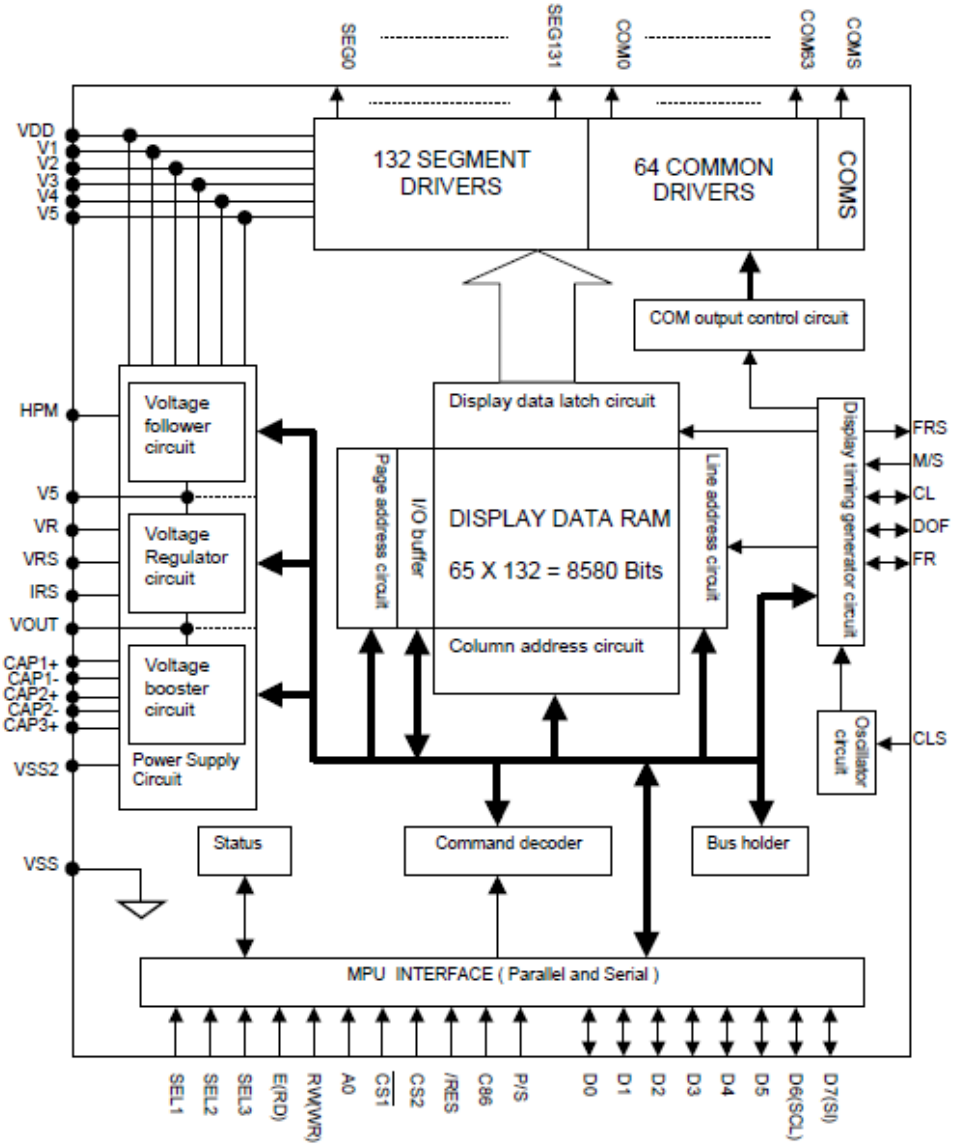



wrong calculation!

72MHz : 16 = 2.25 MHz

# LCD driver chip ST7565 (Sitronix)

## block diagram page 7/56




Data to display is written into the DISPLAY DATA RAM

# LCD minimum Software

Code to be written in order to really see the inverted points!!

```
// Write to SPI
RCC_Config_...
GPIO_Config_...
//deactivate Slave Select
GPIO_ResetBit(GPIOB,GPIO_Pin_6);
SPI_Config_...
delay(10); //ms
```

```
// Initialise GLCD
glcd_reset();
glcd_ST7565R_init();

// Define

glcd_command(0xA5); while(1);
```

Logic Analyzer

One Byte only!

```
Set SS to 0
spi_write()
Set SS to 1
while(1);
```

Many Bytes:

```
Set SS to 0
// wait for SPI available
spi_write()
// wait for SPI sent
set SS to 1
```

# Step 1: Initialise Clock, GPIO, SPI just SCK and MISO

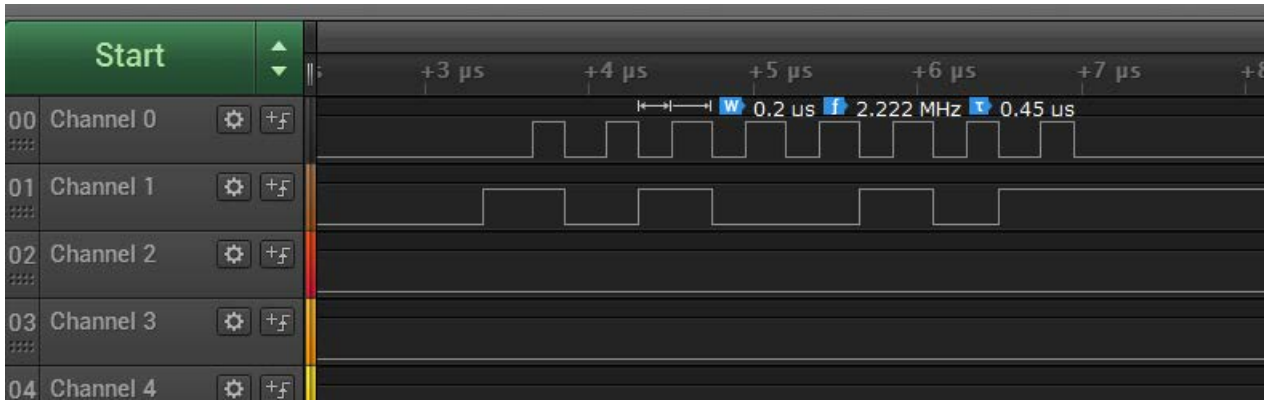
```
void RCC_configuration(void)
{
        //$TASK SPI
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA |
        RCC_APB2Periph_GPIOB | RCC_APB2Periph_SPI1, ENABLE);

void GPIO_configuration(void)
{
        GPIO_InitTypeDef GPIO_InitStructure;

        //$TASK SPI
        /* Set up GPIO for SPI pins (SCK PA5, MOSI PA7) */
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_7;
        GPIO_Init(GPIOA, &GPIO_InitStructure);
```

SPI_ Configuration …

main:

```
        SPI_I2S_SendData(SPI1, (uint16_t) 0xA5);
        while(1);
```

# Step 2: Start reading SPI: Clk, MISO Logic Analyzer, NO SPI decoding




SCK and MOSI are shown, no trigger, no interpretation! Raw data

# Step 3: Initialise NSS slave select line and drive it when sending

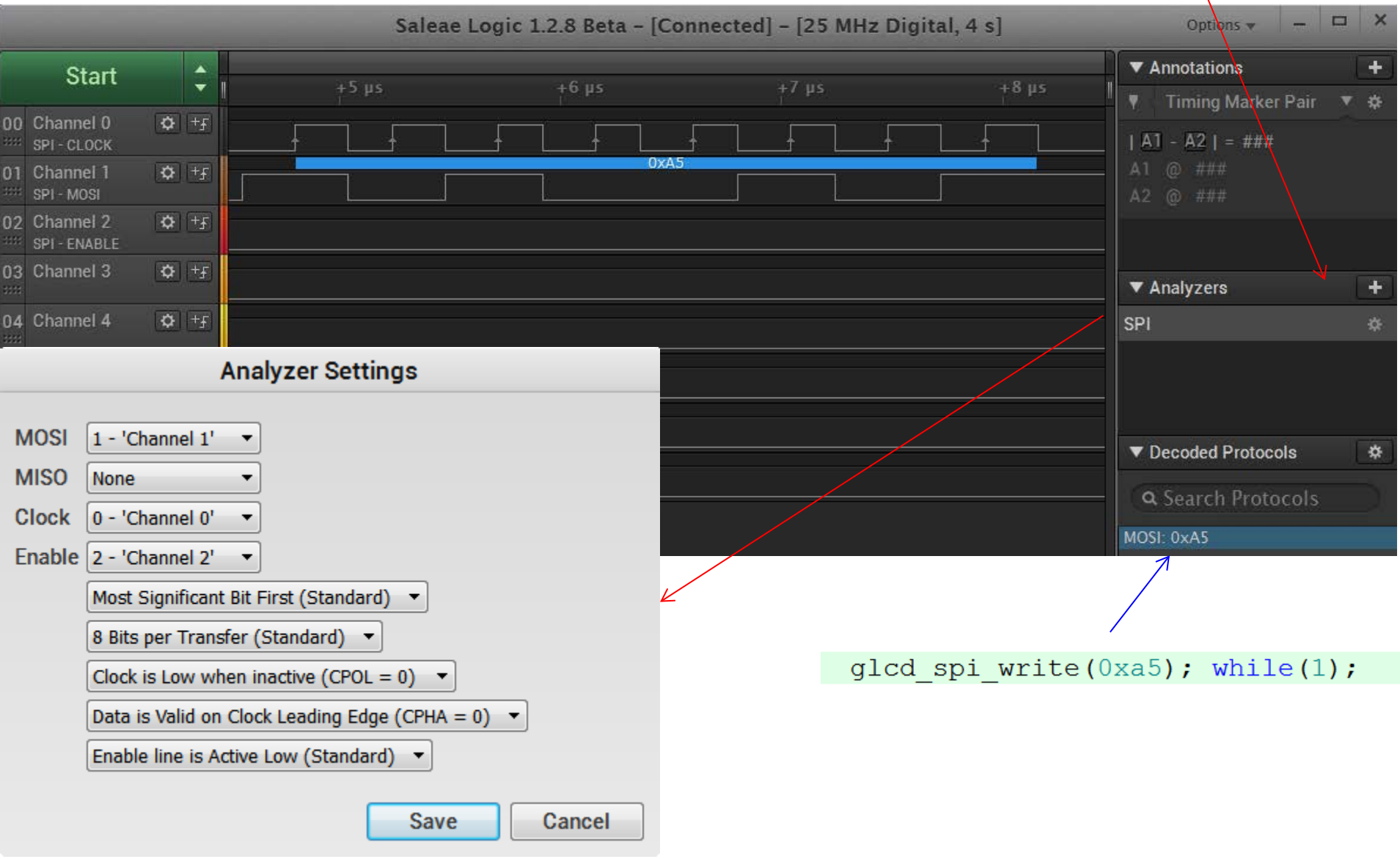
GPIO_Configuration:

```
//$TASK SPI
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;

/* SS pin (PB6) */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
GPIO_Init(GPIOB, &GPIO_InitStructure);
```

# Step 4: Extend glcd_spi_write with select line switching

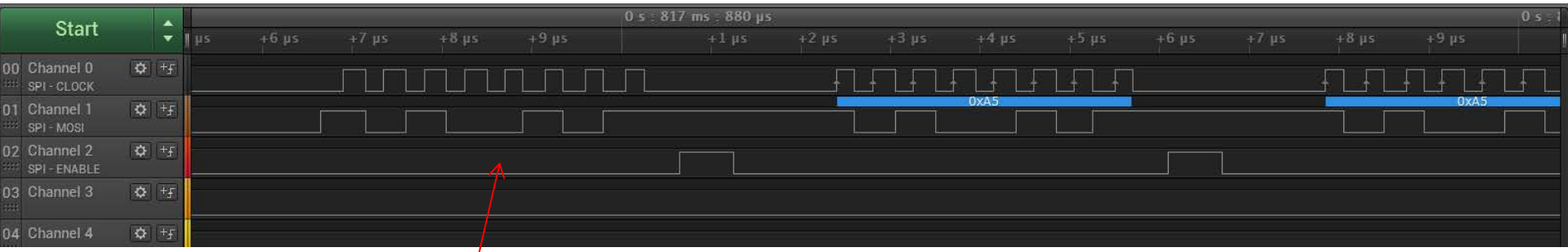
```
void glcd_spi_write(uint8_t c)
{
        //$TASK glcd
        /* Activate GLCD SELECT line*/
        GPIO_ResetBits(GPIOB,GPIO_Pin_6);

        /*!< Loop while DR register in not empty */
        while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);

        SPI_I2S_SendData(SPI1, (uint16_t) c);

        /* Wait until byte has been written */
        while(SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_BSY) != RESET);

        /* Deactivate GLCD SELECT line */
        GPIO_SetBits(GPIOB,GPIO_Pin_6);
}
```

# Logic Analyzer: Setup SPI analyzer




```
glcd_spi_write(0xa5); while(1);
```

# Inspect SPI output right after the µC reset

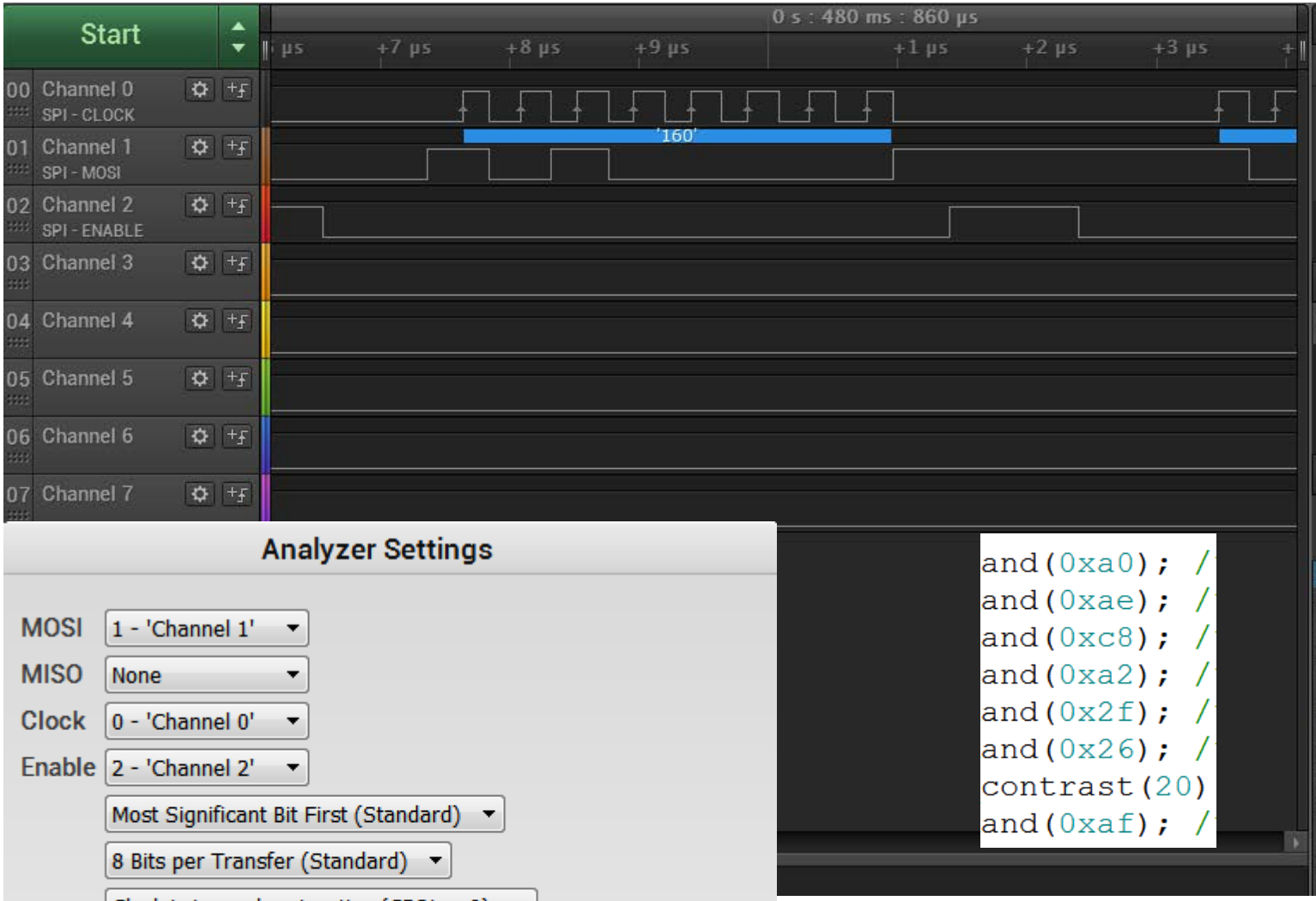


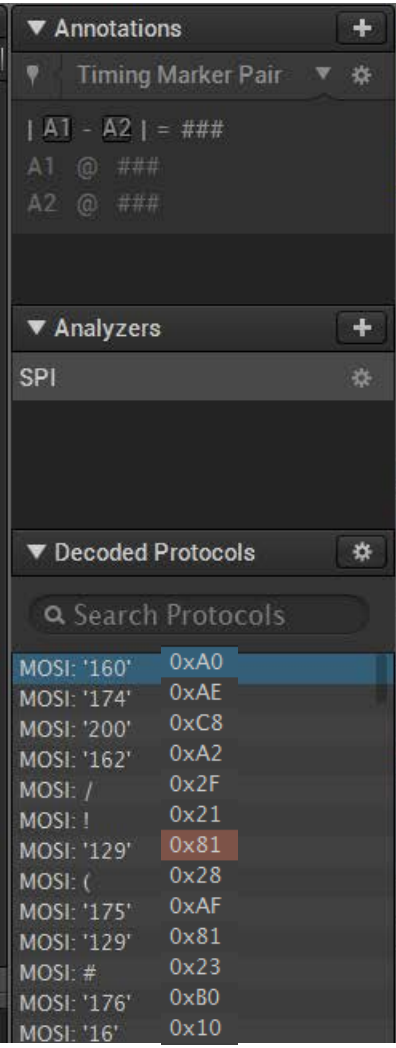

```
while(1){
  glcd_spi_write(0xa5);
}
```

Why is there no correct SPI interpretation of the first Byte sent?

What is the net data throughput we get here?
What is the protocol efficiency? (net data rate / baudrate )

# Logic Analyzer SPI



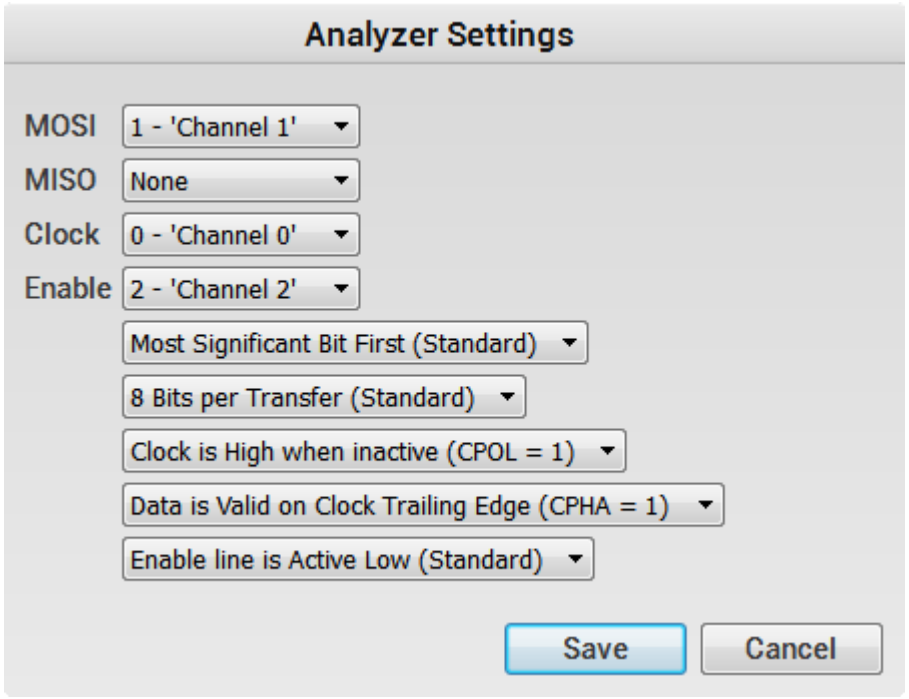



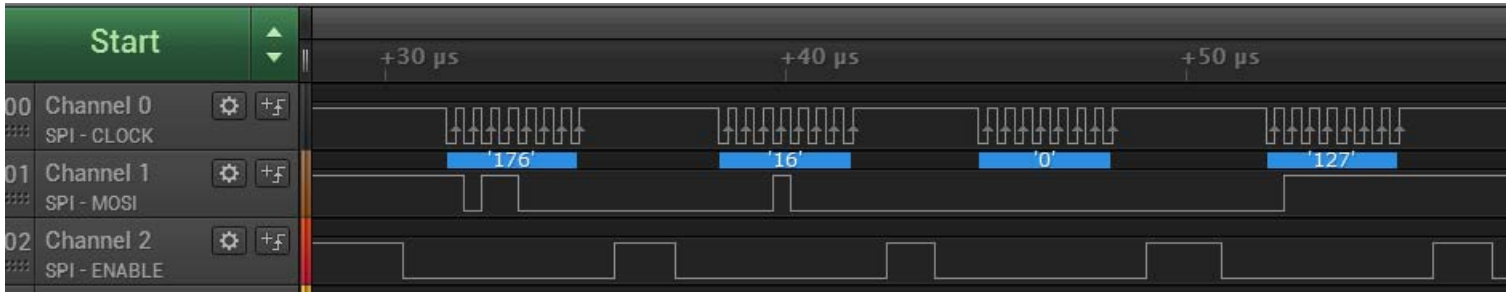




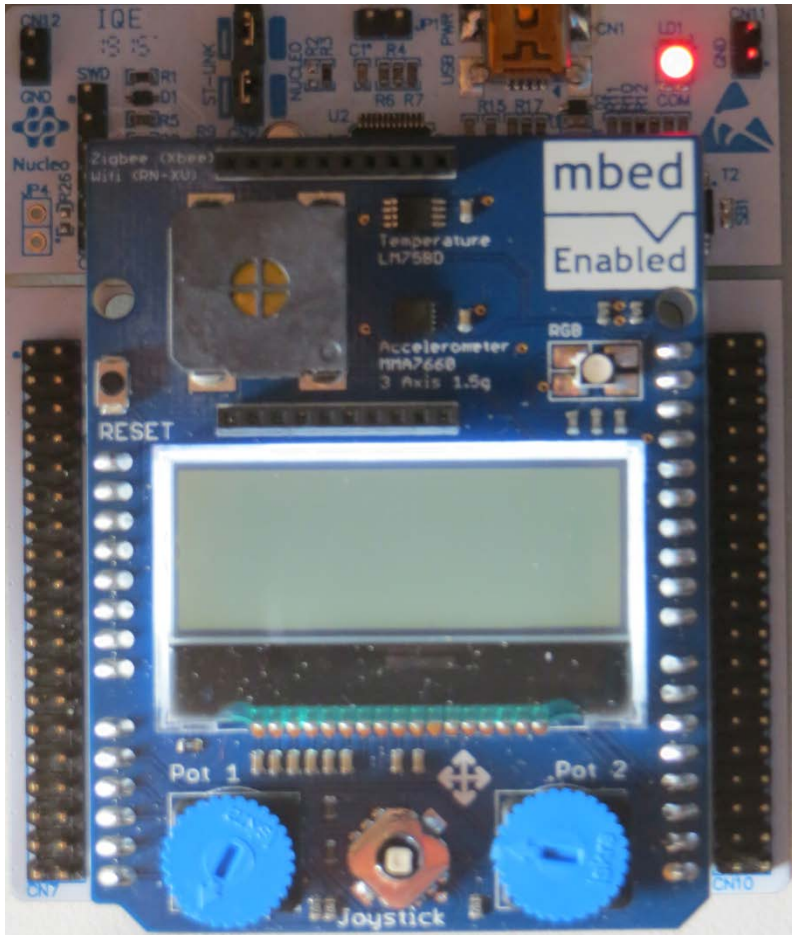
```
and(0xa0); /
and(0xae); /
and(0xc8); /
and(0xa2); /
and(0x2f); /
and(0x26); /
contrast(20)
and(0xaf); /
```

```
0.480869320000000,SPI,MOSI: '200'
0.480875200000000,SPI,MOSI: '162'
0.480881040000000,SPI,MOSI: /
0.480886880000000,SPI,MOSI: !
```

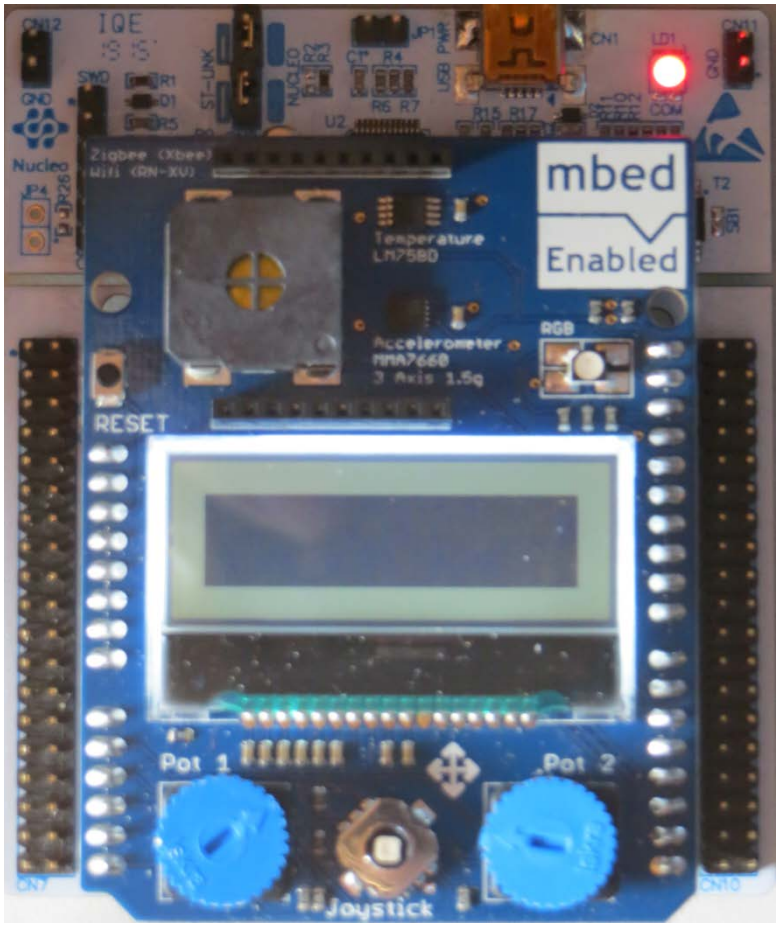

Analyzer Settings
MOSI
1 - 'Channel 1'
MISO
None
Clock
0 - 'Channel 0'
Enable
2 - 'Channel 2'
Most Significant Bit First (Standard)
8 Bits per Transfer (Standard)
Clock is High when inactive (CPOL = 1)
Data is Valid on Clock Trailing Edge (CPHA = 1)
Enable line is Active Low (Standard)
Save
Cancel


Start
+30 µs
+40 µs
+50 µs
00 Channel 0
SPI - CLOCK
01 Channel 1
SPI - MOSI
02 Channel 2
SPI - ENABLE
'176'
'16'
'0'
'127'

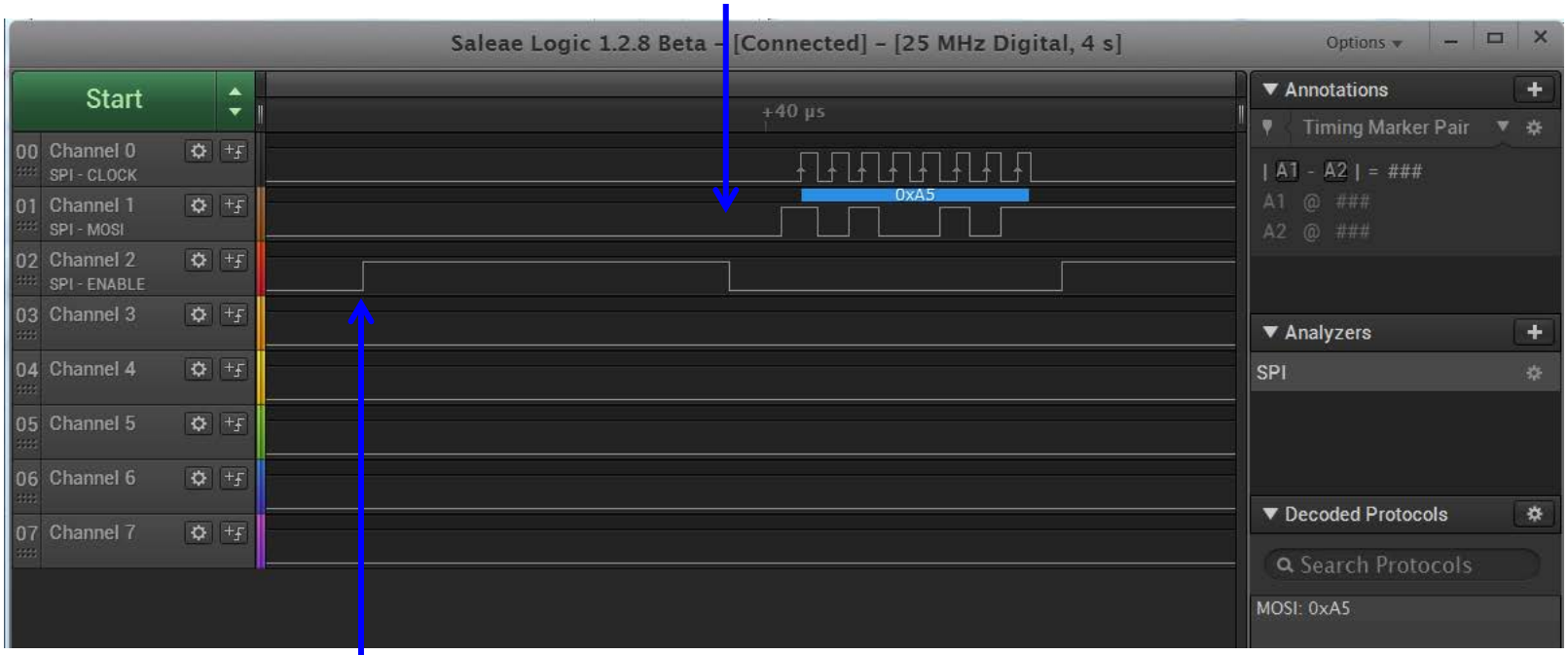
after reset:




after initialisation and Invert command



glcd_command(0xA5);




Enable must be high be default, right from the start:

GPIO_SetBits(GPIOB, GPIO_Pin_6);  // CS inactive high