TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

DVK1906QT2-1-A1

Product Information
SSD1906QT2 Development Kit

DVK1906QT2-1-A1 is a development board of SSD1906QT2. It is intended to help users expedite their design-in of Solomon Systech LCD graphics controller.

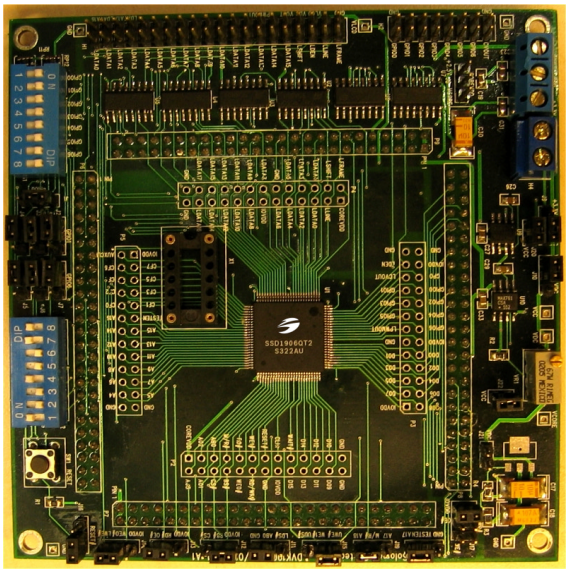


Figure 1: DVK1906QT2-1-A1 board

PACKAGE CONTENTES

DVK1906QT2-1-A1 package consists of the following items:

1. DVK1906QT2-1-A1 development board
2. Application program interface (API) routines in C Language

SYSTEM REQUIREMENT

DVK1906QT2-1-A1 is served as a start point in developing application with SSD1906QT2. To manipulate this graphics controller, a microcontroller board is required to connect to this board. Moreover, a LCD module is attached to development board to display image from SSD1906QT2. The +3.3V power supply for IOVDD of SSD1906QT2 can be supplied either through:

1. IOVDD pins found in MCU connector sockets P2, P3, P4 and P5.
2. Socket H3 pin +3.3V or VCC and select through jumper J9.

DEVELOPMENT BOARD DETAILS

The development board does not include a microcontroller because SSD1906QT2 is able to interface many types of MCU and it'd better for the developers to connect the preference MCU by themselves. It shares the same reason for the arrangement of LCD module.

ORDERING INFORMATION

Item	Ordering Part Number
SSD1906QT2 Development Kit	DVK1906QT2-1-A1

SOLOMON SYSTECH

MCU Interface Sockets

The SSD1906QT2 should be interfaced with external microcontroller to control it and input image data. The connection is done through MCU interface sockets P2, P3, P4 and P5. The pin assignment and function of these sockets are described in the following tables.

Table 1: Socket P2 Pin Descriptions

Pin No.	Pin Name	Function
1	COREVDD	No connection is required
2	AB3	System address bus A3
3	AB2	System address bus A2
4	AB1	System address bus A1
5	AB0	This input pin has multiple functions. • For Generic #1, this pin is not used and should be connected to V_{SS}. • For Generic #2, this is an input of system address bit 0 (A0). • For SH-3/SH-4, this pin is not used and should be connected to Vss. • For MC68K, this is an input of the lower data strobe (LDS#). • For DragonBall, this pin is not used and should be connected to V_{SS}.
6	CS#	Chip select input
7	M/R#	M/R# pin is set high to access display buffer and low to access control registers of SSD1906QT2.
8	BS#	Bus status pin has different functions for particular MCU interface: ▪ For Generic #1, this pin must be tied to IOVDD. ▪ For Generic #2, this pin must be tied to IOVDD. ▪ For SH-3/SH-4, this pin is the input of bus start signal (BS#). ▪ For MC68K, this is the input of address strobe (AS#). ▪ For Dragonball, this pin must be tied to IOVDD.
9	RD#	This pin has different functions for particular MCU interface: • For Generic #1, this is an input of the read command for the lower data byte (RD0#). • For Generic #2, this is an input of the read command (RD#). • For SH-3/SH-4, this pin is the input of read signal (RD#). • For MC68K, this pin must be tied to IOVDD. • For DragonBall, this is an input of the output enable (OE#).
10	WE0#	This pin has different functions for particular MCU interface: • For Generic #1, this is an input of the write enable signal for the lower data byte (WE0#). • For Generic #2, this is an input of the write enable signal (WE#). • For SH-3/SH-4, this pin is the input of write enable signal for low byte (WE0#). • For MC68K, this pin must be tied to IOV_{DD}. • For DragonBall, this is an input of the byte enable signal for the D[7:0] data byte (LWE#).
11	WE1#	This pin has different functions for particular MCU interface: • For Generic #1, this is an input of the write enable signal for the upper data byte (WE1#). • For Generic #2, this is an input of the byte enable signal for the high data byte (BHE#). • For SH-3/SH-4, this pin is the input of write enable signal for high byte (WE1#). • For MC68K, this is an input of the upper data strobe (UDS#). • For DragonBall, this is an input of the byte enable signal for the D[15:8] data byte (UWE#).
12	RD/WR#	This pin has different functions for particular MCU interface:

		• For Generic #1, this is an input of the read command for the upper data byte (RD1#). • For Generic #2, this pin must be tied to IOV_{DD} . • For SH-3/SH-4, this pin is the input of read/write signal (RD/WR#). • For MC68K, this is an input of the R/W# signal. • For DragonBall, this pin must be tied to IOV_{DD} .
13	RESET#	Active low input to set all internal registers to the default state and to force all signals to their inactive states.
14	GND	Ground
15	CLKI	Used as input clock source for bus clock and memory clock
16	IOVDD	Power supply pin
17	WAIT#	During a data transfer, this output pin is driven active to force the system to insert wait states. It is driven inactive to indicate the completion of a data transfer. WAIT# is released to the high impedance state after the data transfer is complete. Its active polarity is configurable. A pull-up or pull-down resistor should be used to resolve any data contention issues. ▪ For Generic #1, this pin outputs the wait signal (WAIT#). • For Generic #2, this pin outputs the wait signal (WAIT#). • For SH-3, this pin outputs the wait request signal (WAIT#). • For SH-4, this pin outputs the device ready signal (RDY#). • For MC68K, this pin outputs the data transfer acknowledge signal (DTACK#). • For DragonBall, this pin outputs the data transfer acknowledge signal (DTACK#).
18	DB15	System data bus D15
19	DB14	System data bus D14
20	DB13	System data bus D13
21	DB12	System data bus D12
22	DB11	System data bus D11
23	DB10	System data bus D10
24	DB9	System data bus D9
25	GND	Ground
26	GND	Ground

Table 2: Socket P3 Pin Descriptions

Pin No.	Pin Name	Function
1	IOVDD	Power supply pin
2	DB8	System data bus D8
3	DB7	System data bus D7
4	DB6	System data bus D6
5	DB5	System data bus D5
6	DB4	System data bus D4
7	DB3	System data bus D3
8	DB2	System data bus D2
9	DB1	System data bus D1
10	DB0	System data bus D0
11	GND	Ground
12	IOVDD	Power supply pin
13	LPWMOUT	Connect to LPWMOUT pin of SSD1906QT2. This output pin has multiple functions. • PWM clock output • General purpose output
14	GPIO6	General purpose IO pin 6 (GPIO6)
15	GPIO5	General purpose IO pin 5 (GPIO5)

16	GPIO4	General purpose IO pin 4 (GPIO4)
17	GPIO3	This pin has multiple functions. • SPL for Sharp HR-TFT • General purpose IO pin 3 (GPIO3)
18	GPIO2	This pin has multiple functions. • REV for Sharp HR-TFT • General purpose IO pin 2 (GPIO2)
19	GPIO1	This pin has multiple functions. • CLS for Sharp HR-TFT • General purpose IO pin 1 (GPIO1)
20	GPIO0	This pin has multiple functions. • PS for Sharp HR-TFT • General purpose IO pin 0 (GPIO0) • Hardware Color Invert input pin
21	LCVOUT	Connect to LCVOUT pin of SSD1906QT2. This output pin has multiple functions. • CV Pulse Output • General purpose output
22	GPO	General purpose output (possibly used for controlling the LCD power)
23	LDEN	Connect to LDEN pin of SSD1906QT2. This output pin has multiple functions. • Display enable (DRDY) for TFT panels • LCD backplane bias signal (MOD) for all other LCD panels
24	IOVDD	Power input pin
25	GND	Ground
26	GND	Ground

Table 3: Socket P4 Pin Descriptions

Pin No.	Pin Name	Function
1	COREVDD	No connection is required
2	LFRAME	Connect to LFRAME pin of SSD1906QT2. This output pin has multiple functions. • Frame Pulse • SPS for Sharp HR-TFT
3	LLINE	Connect to LLINE pin of SSD1906QT2. This output pin has multiple functions. • Line Pulse • LP for Sharp HR-TFT
4	LSHIFT	Connect to LSHIFT pin of SSD1906QT2. This output pin has multiple functions. • Shift Clock • CLK for Sharp HR-TFT
5	LDATA0	LCD panel data LDATA0
6	LDATA1	LCD panel data LDATA1
7	LDATA2	LCD panel data LDATA2
8	LDATA3	LCD panel data LDATA3
9	LDATA4	LCD panel data LDATA4
10	LDATA5	LCD panel data LDATA5
11	LDATA6	LCD panel data LDATA6
12	GND	Ground
13	IOVDD	Power supply pin
14	LDATA7	LCD panel data LDATA7
15	LDATA8	LCD panel data LDATA8
16	LDATA9	LCD panel data LDATA9
17	LDATA10	LCD panel data LDATA10
18	LDATA11	LCD panel data LDATA11
19	LDATA12	LCD panel data LDATA 12

20	LDATA13	LCD panel data LDATA13
21	LDATA14	LCD panel data LDATA14
22	LDATA15	LCD panel data LDATA15
23	LDATA16	LCD panel data LDATA16
24	LDATA17	LCD panel data LDATA17
25	GND	Ground
26	GND	Ground

Table 4: Socket P5 Pin Descriptions

Pin No.	Pin Name	Function
1	IOVDD	Power supply pin
2	AUXCLK	Connect to AUXCLK of SSD1906. This pin may be used as input clock source for pixel clock.
3	CF7	Configuration pin CF7. No connection is required.
4	CF6	Configuration pin CF6. No connection is required.
5	CF5	Configuration pin CF5. No connection is required.
6	CF4	Configuration pin CF4. No connection is required.
7	CF3	Configuration pin CF3. No connection is required.
8	CF2	Configuration pin CF2. No connection is required.
9	CF1	Configuration pin CF1. No connection is required.
10	CF0	Configuration pin CF0. No connection is required.
11	AB17	System address bus A17
12	AB16	System address bus A16
13	AB15	System address bus A15
14	AB14	System address bus A14
15	AB13	System address bus A13
16	AB12	System address bus A12
17	AB11	System address bus A11
18	AB10	System address bus A10
19	AB9	System address bus A9
20	AB8	System address bus A8
21	AB7	System address bus A7
22	AB6	System address bus A6
23	AB5	System address bus A5
24	AB4	System address bus A4
25	GND	Ground
26	GND	Ground

LCD interface header

The socket header H1 and H2 is designed to be interfaced with external LCD panel module. The pin assignment and description of these two sockets are show in the following tables.

Table 5: Header H1 Pin Descriptions

Pin No.	Pin Name	Function
1	LDATA0	LCD panel data LDATA0
2	GND	Ground
3	LDATA1	LCD panel data LDATA1
4	LDATA16	LCD panel data LDATA16
5	LDATA2	LCD panel data LDATA2
6	LDATA17	LCD panel data LDATA17
7	LDATA3	LCD panel data LDATA3
8	GND	Ground
9	LDATA4	LCD panel data LDATA4
10	GND	Ground
11	LDATA5	LCD panel data LDATA5
12	GND	Ground
13	LDATA6	LCD panel data LDATA6
14	GND	Ground
15	LDATA7	LCD panel data LDATA7
16	GND	Ground
17	LDATA8	LCD panel data LDATA8
18	GND	Ground
19	LDATA9	LCD panel data LDATA9
20	GND	Ground
21	LDATA10	LCD panel data LDATA10
22	GND	Ground
23	LDATA11	LCD panel data LDATA11
24	GND	Ground
25	LDATA12	LCD panel data LDATA12
26	GND	Ground
27	LDATA13	LCD panel data LDATA13
28	LPWMOUT	Connect to LPWMOUT pin of SSD1906QT2. This output pin has multiple functions. • PWM Clock output • General purpose output
29	LDATA14	LCD panel data LDATA14
30	NC	No connection
31	LDATA15	LCD panel data LDATA15
32	VOUT	Power supply to drive LCD panel
33	LSHIFT	Connect to LSHIFT pin of SSD1906QT2. This output pin has multiple functions. • Shift Clock • CLK for Sharp HR-TFT
34	VDC	+5V supply
35	LDEN	Connect to LDEN pin of SSD1906QT2. This output pin has multiple functions. • Display enable (DRDY) for TFT panels • LCD backplane bias signal (MOD) for all other LCD panels
36	VLCD	+5V to 15V Power supply for LCD
37	LLINE	Connect to LLINE pin of SSD1906QT2. This output pin has multiple functions. • Line Pulse • LP for Sharp HR-TFT
38	LDEN	This pin is connected to pin 35 LDEN
39	LFRAME	Connect to LFRAME pin of SSD1906QT2. This output pin has multiple functions. • Frame Pulse • SPS for Sharp HR-TFT

40	GPO	General Purpose Output (possibly used for controlling the LCD power).

Table 6: Header H2 Pin Descriptions

Pin No.	Pin Name	Function
1	GPIO0	This pin has multiple functions. • PS for Sharp HR-TFT • General purpose IO pin 0 (GPIO0) • Hardware Color Invert input pin
2	GND	Ground
3	GPIO1	This pin has multiple functions. • CLS for Sharp HR-TFT • General purpose IO pin 1 (GPIO1)
4	GND	Ground
5	GPIO2	This pin has multiple functions. • REV for Sharp HR-TFT • General purpose IO pin 2 (GPIO2)
6	GND	Ground
7	GPIO3	This pin has multiple functions. • SPL for Sharp HR-TFT • General purpose IO pin 3 (GPIO3)
8	GND	Ground
9	GPIO4	General purpose IO pin 4 (GPIO4)
10	GND	Ground
11	GPIO5	General purpose IO pin 5 (GPIO5)
12	GND	Ground
13	GPIO6	General purpose IO pin 6 (GPIO6)
14	GND	Ground
15	LCVOUT	Connect to LCVOUT pin of SSD1906QT2. This output pin has multiple functions. • CV Pulse Output • General purpose output
16	GND	Ground

Dip switch setting

In DVK1906QT2-1-A1 development board, there are two dip switches S1, S2 for different configuration purpose. Their functions are described in the tables below.

Table 7: Configuration of dip switch S1

Dip switch S1	Close (On/0)	Open (Off/1)
1	GPIO0 pull down (if J1 is set to 2-3)	GPIO0 pull up (if J1 is set to 2-3)
2	GPIO1 pull down (if J2 is set to 2-3)	GPIO1 pull up (if J2 is set to 2-3)
3	GPIO2 pull down (if J3 is set to 2-3)	GPIO2 pull up (if J3 is set to 2-3)
4	GPIO3 pull down (if J4 is set to 2-3)	GPIO3 pull up (if J4 is set to 2-3)
5	GPIO4 pull down (if J5 is set to 2-3)	GPIO4 pull up (if J5 is set to 2-3)
6	GPIO5 pull down (if J6 is set to 2-3)	GPIO5 pull up (if J6 is set to 2-3)
7	GPIO6 pull down (if J7 is set to 2-3)	GPIO6 pull up (if J7 is set to 2-3)
8	Reserved	Reserved

For dip switches S2, they are used to set configuration the logic level of inputs CF[7:0] of SSD1906QT2. These input values are read on the rising edge of reset signal. The meanings of setting of CF[7:0] with respect to dip switches S2 is described below.

Table 8: Configuration of dip switch S2

Dip switch S2	Close (On/Logic 0)		Open (Off/Logic 1)	
1-3 (CF[2:0])	Switch 3 (CF2)	Switch 2 (CF1)	Switch 1 (CF0)	Host bus interface
	0	0	0	SH-3/SH-4
	0	0	1	MC68K
	0	1	0	Reserved
	0	1	1	Generic #1
	1	0	0	Generic #2
	1	0	1	Reserved
	1	1	0	Dragonball
	1	1	1	Reserved
4 (CF3)	Configure GPIO pins as outputs at power-on (for use by HR-TFT when selected)		Configure GPIO pins as inputs at power-on	
5 (CF4)	Little endian bus interface		Big endian bus interface	
6 (CF5)	WAIT# is active low		WAIT# is active high	
7-8 (CF[7:6])	Switch 4 (CF7)	Switch (CF6)	CLKI to BCLK divide ratio	
	0	0	1:1	
	0	1	2:1	
	1	0	3:1	
	1	1	4:1	

Jumper setting

The functions of each jumper positions are listed in the following table.

Table 9: Configuration of jumper

Jumper	Function	Position 1-2	Position 2-3	No Jumper
J1	GPIO0 connection	Connect to LCD output header H2	Either pull up and down by dip switch S1 setting	N/A
J2	GPIO1 connection	Connect to LCD output header H2	Either pull up and down by dip switch S1 setting	N/A
J3	GPIO2 connection	Connect to LCD output header H2	Either pull up and down by dip switch S1 setting	N/A
J4	GPIO3 connection	Connect to LCD	Either pull up and down	N/A

		output header H2	by dip switch S1 setting	
J5	GPIO4 connection	Connect to LCD output header H2	Either pull up and down by dip switch S1 setting	N/A
J6	GPIO5 connection	Connect to LCD output header H2	Either pull up and down by dip switch S1 setting	N/A
J7	GPIO6 connection	Connect to LCD output header H2	Either pull up and down by dip switch S1 setting	N/A
J8	COREVDD connection	COREVDD is connected to +2.5V from H4 (for no regulator version of SSD1906QT2 only)	N/A	COREVDD is regulated by SSD1906QT2 itself
J9	IOVDD connection	IOVDD is connected to +3.3V from H3	IOVDD is connected to VCC from H3	No change on connection (Note 1)
J10	VOUT (LCD panel power) connection	VOUT is connected to VDC	VOUT is connected to VCC from H3 (Note 2)	N/A
J11	A0 connection	N/A	Connect to ground	No change on connection
J12	BS# connection	BS# is connected to IOVDD	BS# is connected to CS#	No change in connection
J13	RD# connection	RD# is connected to IOVDD	N/A	No change in connection
J14	WE0# connection	WE0# is connected to IOVDD	N/A	No change in connection
J15	WE1# connection	N/A	N/A	No change in connection
J16	Reset# connection	N/A	Reset generated from SW1 reset switch	No change in connection
J18	TESTEN connection	N/A	Connect to GND	N/A
J19	M/R connection	N/A	N/A	N/A

Jumper	**Function**	**Connection**	
		Short	**Open**
J20	VDC connection	VDC = +5V	floated
J21	NC	N/A	N/A
J22	VLCD connection	VLCD = +5 to 15V (controlled by VR1)	floated

Jumper	**Function**	**Position 1-2**	**Position 3-4**	**Position 5-6**	**No Jumper**
J17	RD/WR# connection	Connect to IOVDD	N/A	N/A	No change in connection

Note 1: Since DVK1906QT2-A0 board is usually connected to external microcontroller through sockets P2, P3, P4 and P5, IOVDD is normally supplied from these sockets also. There is no need to give IOVDD from socket H3.
Note 2: If VOUT (LCD panel signal power) is the same as IOVDD, VOUT can get power from IOVDD by configuring the jumper connection like this: J9: 2-3 and J10: 2-3. If VOUT (LCD panel signal power) is +5V, VOUT can get power from VDC by configuring the jumper connection like this: J10: 1-2 and J20: Short. Otherwise, users have to supply VOUT power by wrapping an external power cable to pin 2 of J10.

AUXCLK Input

If AUXCLK is enabled and input from AUXCLK pin of SSD1906QT2, users can plug an oscillator with appropriate frequency into socket X1 to provide AUXCLK clock signal.

RESET Switch

If jumper J16 is connected to 2-3 position, the active low reset signal to SSD1906QT2 is generated when SW1 switch is pressed down.

Power supply socket H3 and H4

Since DVK1906QT2-A1 board is usually connected to external microcontroller through sockets P2, P3, P4 and P5, IOVDD is normally supplied from these sockets also. There is no need to give IOVDD from socket H3 through +3.3V or VCC pins. In addition, SSD1906QT2 normally includes an internal regulator to provide +2.5V COREVDD supply. Hence it's not required to supply COREVDD from socket H4 through +2.5V pin.

VLCD Power adjustment

DVK1906QT2-A1 board provided a convenient power for most kind of LCD panels require a positive high voltage VLCD power. The VLCD power signal exists in the connector from DVK1906QT2-A1 to LCD panel. VR1 is used to adjust VLCD voltage from +5V to +15V.

Extension socket

The extension sockets P6, P7, P8 and P9 are designed for debugging use when the development board is manufactured. These sockets are connected to Motorola Dragonball MC68VZ328. If the target system is Dragonball VZ MCU, these 4 sockets can be considered as connection interface. The pin assignment can be found at development board schematic section.

SSD1906 Application Program Interface

The application program interface (API) is designed to ease programming effort by providing standard routines to perform features of SSD1906. They are originally designed for NEC VR4181 and Motorola Dragonball M68VZ328 platforms. However, it is easy to plot the APIs into other platforms as they are written in C language.

The APIs described below are high-level functions for programmers to learn how to program SSD1906 quickly and expedite the design process. If there is any specific function requirement that cannot be fulfilled by these APIs, users are advised to study the source codes of these APIs and modify them to suit the need.

The File Structure of SSD1906 API

The APIs are constructed by several C programs and header files. The brief introduction of each file is as follows.

Table 10: SSD1906 API program structure

Program File	Description
Bitmap.c	Contains functions for display and finding the properties of bitmaps
Cursor.c	Cursor 1 and 2 display routines
Mc68vz328.c	Program for initialization and low level operations of Motorola Dragonball MC68VZ328 MCU.
VR4181.c	Program for initialization and low level operations of NEC VR4181 MIPS MCU.
lcddrv.c	Initialize SSD1906 registers to enable its functions
Main.c	The C program contains main loop
Memory.c	Initialize, allocate and free display memory routines
Rotate.c	Functions supporting display rotate mode with 0, 90, 180, 270 degree counter-clockwise hardware rotation for display image.
SSD1906.c	Low level routines to read/write registers and memory buffer of SSD1906 graphics controller
Virtual.c	Routine provided for virtual display feature
Bitmap.h	Bitmap file format structures definition
Mc68vz328.h	Dragonball MCU registers address definition
VR4181.h	VR4181 MCU registers address definition
Lcd.h	LCD panels size definition
Lcdinfo.h	SSD1906 control registers preset value definition
SSD1906.h	SSD1906 control registers address definition

Summary of SSD1906 API

In the table below, these APIs are classified into several categories according to their functionalities.

Table 11: SSD1906 API summary

API name	Function
Initialization	
McuInit	To initialize the MCU to make it able to communicate with SSD1906
LcdInit	To initialize the registers and display memory of SSD1906
MCU Register Operation	

RdMcuByteReg	Read MCU register value with size of one byte
RdMcuWordReg	Read MCU register value with size of one word
RdMcuDWordReg	Read MCU register value with size of one double word
WtMcuByteReg	Write value into MCU register with size of one byte
WtMcuWordReg	Write value into MCU register with size of one word
WtMcuDWordReg	Write value into MCU register with size of one double word
SSD1906 Register Operation	
ReadRegByte	Read SSD1906 register value with size of one byte
ReadRegWord	Read SSD1906 register value with size of one word
ReadRegDword	Read SSD1906 register value with size of one double word
WriteRegByte	Write value into SSD1906 register with size of one byte
WriteRegWord	Write value into SSD1906 register with size of one word
WriteRegDword	Write value into SSD1906 register with size of one double word
Disp1906AllReg	Display the content of all SSD1906 registers on debugger console window
Display Memory Operation	
ReadDisplayByte	Read SSD1906 display memory value with size of one byte
ReadDisplayWord	Read SSD1906 display memory value with size of one word
ReadDisplayDword	Read SSD1906 display memory value with size of one double word
WriteDisplayBytes	Write value into SSD1906 display memory with size of one byte and certain number of times
WriteDisplayWords	Write value into SSD1906 display memory with size of one word and certain number of times
WriteDisplayDwords	Write value into SSD1906 display memory with size of one double word and certain number of times
Disp1906Mem	Display the content of SSD1906 memory with the specified start and end address on debugger console window
Bitmap Operation	
MainWinDispOn	Display bitmap on main window of SSD1906
MainWinDispFree	Free the allocated memory occupied by bitmap displayed on main window of SSD1906
FloatWinDispOn	Display bitmap on floating window of SSD1906
FloatWinDispOff	Turn off floating window and free the allocated memory occupied by bitmap displayed on floating window of SSD1906
ReadBMPInfo	Read the width, height and bit-per-pixel of bitmap
WriteBmpLUT	Read LUT of bitmap and write into LUT entries of SSD1906
Display Rotation	
Rot0MainBmp	Rotate the main bitmap window by 0 degree counter-clockwise
Rot90MainBmp	Rotate the main bitmap window by 90 degree counter-clockwise
Rot180MainBmp	Rotate the main bitmap window by 180 degree counter-clockwise
Rot270MainBmp	Rotate the main bitmap window by 270 degree counter-clockwise
Rot0FloatBmp	Rotate the floating bitmap window by 0 degree counter-clockwise
Rot90FloatBmp	Rotate the floating bitmap window by 90 degree counter-clockwise
Rot180FloatBmp	Rotate the floating bitmap window by 180 degree counter-clockwise
Rot270FloatBmp	Rotate the floating bitmap window by 270 degree counter-clockwise
Virtual Display	
VirtMovePic	Move the virtual main bitmap window
Cursor Operation	
Cursor1Blink	Set the blinking period of cursor 1
Cursor2Blink	Set the blinking period of cursor 2
Cursor1Color	Define the RGB color value for cursor 1
Cursor2Color	Define the RGB color value for cursor 2
Cursor1DispOn	Display cursor 1 on LCD panel
Cursor2DispOn	Display cursor 2 on LCD panel

Cursor1DispOff	Turn off cursor 1
Cursor2DispOff	Turn off cursor 2
Memory Operation	
MemRemainSize	Find the remaining size of memory available to be further used
MemUsedSize	Find the already allocated size of displayed memory
Miscellaneous	
Disp1906LUT	Display SSD1906 color look-up table content on debugger console window
SSD1906Delay	Create a delay time in seconds by SSD1906
CheckEndian	Check if the MCU system is big or little endian
DispBlank	Enable/disable blanking of the screen
DispMainNxNChecker	Display a n by n checker board image

SSD1906 API description

Initialization

McuInit

Prototype: void McuInit(void)
Description: Initialize the microcontroller system such that it can communicate with SSD1906. For different type of microcontrollers, the initialization procedures are difference. If the system is not MC68VZ328 or VR4181, user should make following changes in this routine.
1. Preserve memory space to map SSD1906 registers and display buffer into the system.
2. Configure MCU memory control pins such that it can interface with SSD1906.

This function should be called first before any other SSD1906 APIs.
Parameters: None
Return value: None

LcdInit

Prototype: void LcdInit(void)
Description: This function initializes SSD1906 LCD graphics controller by setting the control registers with values defined in “LcdInfo.h” file. Moreover, it initializes display memory and clears its contents.
Parameters: None
Return value: None

MCU Register Operation

These functions are used to access the control registers of microcontroller. Since these routines are written such that the real address of register is calculated as “base address + reg”, where reg is the offset address of register. Users have to define the correct “base address” value for their microcontroller system.

RdMcuByteReg

Prototype: BYTE RdMcuByteReg(DWORD reg)
Description: Read the value of register from MCU with size of one byte at address "reg".
Parameters: reg The offset address of register to be read
Return value: Read value is returned in BYTE size

RdMcuWordReg

Prototype: WORD RdMcuWordReg(DWORD reg)
Description: Read the value of register from MCU with size of one word at address "reg".

Parameters: reg The offset address of register to be read
Return value: Read value is returned in WORD size

RdMcuDWordReg

Prototype: DWORD RdMcuDWordReg(DWORD reg)
Description: Read the value of register from MCU with size of one double word at address "reg".
Parameters: reg The offset address of register to be read
Return value: Read value is returned in DWORD (double word) size

WtMcuByteReg

Prototype: void WtMcuByteReg(DWORD reg, BYTE val)
Description: Write the value into register with size of one byte at address "reg".
Parameters: reg The offset address of register to be written
val The byte value to be written
Return value: None

WtMcuWordReg

Prototype: void WtMcuWordReg(DWORD reg, WORD val)
Description: Write the value into register with size of one word at address "reg".
Parameters: reg The offset address of register to be written
val The word value to be written
Return value: None

WtMcuDWordReg

Prototype: void WtMcuDWordReg(DWORD reg, DWORD val)
Description: Write the value into register with size of one double word at address "reg".
Parameters: reg The offset address of register to be written
val The double word value to be written
Return value: None

SSD1906 Register Operation

These routines provide function to access the control register of SSD1906. As the real address of the registers are found by “RegAddress + index”, users have to define the value of “RegAddress” according to the mapping address of control registers into MCU system memory space.

ReadRegByte

Prototype: BYTE ReadRegByte(DWORD index)
Description: Read the value of register with size of one byte at address "index".
Parameters: index The offset address of register to be read
Return value: Read value is returned in BYTE size

ReadRegWord

Prototype: WORD ReadRegWord(DWORD index)
Description: Read the value of register with size of one word at address "index".
Parameters: index The offset address of register to be read
Return value: Read value is returned in WORD size

ReadRegDword

Prototype: DWORD ReadRegDword(DWORD index)
Description: Read the value of register with size of one double word at address "index".
Parameters: index The offset address of register to be read
Return value: Read value is returned in DWORD (double word) size

WriteRegByte

Prototype: void WriteRegByte(DWORD index, BYTE value)
Description: Write the value into register with size of one byte at address "index".
Parameters: index The offset address of register to be written
value The byte value to be written
Return value: None

WriteRegWord

Prototype: void WriteRegWord(DWORD index, WORD value)
Description: Write the value into register with size of one word at address "index".
Parameters: index The offset address of register to be written
value The word value to be written
Return value: None

WriteRegDword

Prototype: void WriteRegDword(DWORD index, DWORD value)
Description: Write the value into register with size of one double word at address "index".
Parameters: index The offset address of register to be written
value The double word value to be written
Return value: None

Disp1906AllReg

Prototype: void Disp1906AllReg(void)
Description: Display all control registers value of SSD1906 on debugger console window.
Parameters: None
Return value: None

Display Memory Operation

These routines are used to access the display memory of SSD1906. As the real address of the display memory are calculated by “MemAddress + adrOff”, users have to define the value of “MemAddress” according to the mapping address of display memory into MCU system memory space.

ReadDisplayByte

Prototype: BYTE ReadDisplayByte(DWORD adrOff)
Description: Read the value of display memory with size of one byte at address "adrOff".
Parameters: adrOff The offset address of display memory to be read
Return value: Read value is returned in BYTE size

ReadDisplayWord

Prototype: WORD ReadDisplayWord(DWORD adrOff)
Description: Read the value of display memory with size of one word at address "adrOff".
Parameters: adrOff The offset address of display memory to be read
Return value: Read value is returned in WORD size

ReadDisplayDword

Prototype: DWORD ReadDisplayDword(DWORD adrOff)
Description: Read the value of display memory with size of one double word at address "adrOff".
Parameters: adrOff The offset address of display memory to be read
Return value: Read value is returned in DWORD size

WriteDisplayBytes

Prototype: void WriteDisplayBytes(DWORD adrOff, BYTE value, DWORD count)

Description: Write the value into register with size of one byte at address "adrOff" with repeat number of times "count".
Parameters: adrOff The offset address of display memory to be written
value The byte value to be written
count Number of the same byte value to be written
Return value: None

WriteDisplayWords

Prototype: void WriteDisplayWords(DWORD adrOff, WORD value, DWORD count)
Description: Write the value into register with size of one word at address "adrOff" with repeat number of times "count".
Parameters: adrOff The offset address of display memory to be written
value The word value to be written
count Number of the same word value to be written
Return value: None

WriteDisplayDwords

Prototype: void WriteDisplayDwords(DWORD adrOff, DWORD value, DWORD count)
Description: Write the value into register with size of one double word at address "adrOff" with repeat number of times "count".
Parameters: adrOff The offset address of display memory to be written
value the double word value to be written
count number of the same double word value to be written
Return value: None

Disp1906Mem

Prototype: void Disp1906Mem(int startAdr, int endAdr)
Description: Show display memory values of SSD1906 on debugger console window.
Parameters: startAdr the start offset address of display memory content to be shown
endAdr the end offset address of display memory content to be shown
Return value: None

Bitmap Operation

SSD1906 API is able to display bitmap on LCD panel. However, the bitmap file should be converted into a specified format in C program first. A program utility “bmpconv.exe” provides such service. The procedure to use this utility to change a bitmap file into C program is as follows.

1. Suppose there is a bitmap file called “abc.bmp” in the same directory as bmpconv.exe.
2. Type the command: bmpconv.exe c abc.bmp to convert bitmap file.
3. If the command is success, the C program file abc.c is generated.
4. The C program abc.c can be converted back to bitmap file by typing the command: bmpconv.exe b abc.c.

The content of abc.c program is an array with the name abc[]. Afterwards, this bitmap can be displayed by calling appropriate routine which reads the image content at starting address “abc”.

Care should be taken on the physical dimension of bitmap file. The horizontal size of bitmap should be multiple of value 32/bit-per-pixel. For instance, if the image of bitmap is 8bpp, the horizontal size should be multiple of 4 (32/8 = 4).

Before displaying a bitmap on screen, it has to ensure that enough display buffer is available for the bitmap. Users can call memory operation APIs to find out the remaining display buffer size in SSD1906. The memory size required by a bitmap can be calculated as the equation below.

Required buffer size = Width x Height x (bit-per-pixel / 8)

It is difficult to find 16 bit-per-pixel bitmaps. A 24-bit, i.e. true color, bitmap can be used and SSD1906 APIs translate it into 16 bit-per-pixel format automatically.

MainWinDispOn

Prototype: BOOL MainWinDispOn(const unsigned char *Image)
Description: Allocate display buffer memory and display bitmap on main window of SSD1906. In 1, 2, 4, 8 bit-per-pixel situations, the Lookup Table (LUT) is read from bitmap and written into LUT entries of SSD1906.
Parameters: Image the starting address of bitmap array in system
Return value: TRUE success to display bitmap on main window
FALSE fail to allocate memory to display bitmap

MainWinDispFree

Prototype: void MainWinDispFree(void)
Description: Free the allocated display buffer memory occupied by bitmap on main window of SSD1906. The sequence of calling MainWinDispOn and this routine should be taken care such that the memory is properly freed.
Parameters: None
Return value: None

FloatWinDispOn

Prototype: BOOL FloatWinDispOn (const unsigned char *Image, int startx, int starty)
Description: Allocate display buffer memory to display bitmap on floating window of SSD1906 and define the display position on LCD panel. The origin of the position (0, 0) is at upper left corner of the screen.
In 1, 2, 4, 8 bit-per-pixel situations, the color of image in floating window is referenced to the LUT of main window. Therefore, it has to ensure that the LUT of bitmap of floating window is the same as the LUT of main window for correct color display. Moreover, the bit-per-pixel of bitmap in main and floating windows should be the same.
Parameters: Image the starting address of bitmap array in system
startx upper left x-coordinate of floating window
starty upper left y-coordinate of floating window
Return value: TRUE success to display bitmap on floating window
FALSE fail to allocate memory to display bitmap

FloatWinDispOff

Prototype: void FloatWinDispOff (void)
Description: Free the allocated display buffer memory occupied by bitmap on floating window of SSD1906. Besides, the floating window is turned off. The sequence of calling FloatWinDispOn and this routine should be taken care such that the memory is properly freed.
Parameters: None
Return value: None

ReadBMPInfo

Prototype: void ReadBMPInfo (const unsigned char *BmpPtr, DWORD *bWidth, DWORD *bHeight, WORD *bmpBpp)
Description: Read the width, height and bit-per-pixel of a bitmap
Parameters: BmpPtr the starting address of bitmap array in system
bWidth width of bitmap file to be read back
bHeight Height of bitmap file to be read back
bmpBpp Bit-per-pixel of bitmap file to be read back
Return value: None

WriteBmpLUT

Prototype: void WriteBmpLUT (const unsigned char *BmpPtr)
Description: Read the Lookup table (LUT) of bitmap and write into LUT entries of SSD1906. If the bit-per-pixel of bitmap is greater than 8-bit, no action in this routine is performed.
Parameters: BmpPtr the starting address of bitmap array in system
Return value: None

Example of Bitmap Operation

Suppose we want to display a bitmap with array Img1 on main window. This bitmap is converted by Img1.bmp with bmpconv.exe. Moreover, another bitmap with array Img2 is displayed on floating window for a while. Afterwards, the floating window is turned off and main window occupied memory is freed. The program is written as below.

```
MainWinDispOn(Img1);
FloatWinDispOn(Img2, 120, 5);        //Display position(x,y) = (120, 5)
SSD1906Delay(2);                     //Delay loop for 2 seconds
FloatWinDispOff();                   //Pay attention to sequence of calling
MainWinDispFree();                   //these two functions to free memory
```

Display Rotation

This feature provides 0, 90, 180 and 270 degree counter-clockwise rotation of main and floating windows. However, attention should be paid before performing rotation. In 0 and 180 degree cases, the image ***width*** should be a multiple of 32 ÷ bit-per-pixel. For 90 and 270 situations, the ***height*** of image should be a multiple of 32 ÷ bit-per-pixel. These criterions are applied to rotated image in main and floating windows.

Whenever there is a main and floating window displaying on LCD panel at the same time, their display orientation should be the same. User has to ensure this in the program.

Rot0MainBmp

Prototype: void Rot0MainBmp (const unsigned char *Image)
Description: Rotate the image to 0-degree orientation in main window. This image should be the one which is displayed by calling the function MainWinDispOn.
Parameters: Image the starting address of bitmap array in system
Return value: None

Rot90MainBmp

Prototype: void Rot90MainBmp (const unsigned char *Image)
Description: Rotate the image to 90-degree counter-clockwise orientation in main window. This image should be the one which is displayed by calling the function MainWinDispOn.
Parameters: Image the starting address of bitmap array in system
Return value: None

Rot180MainBmp

Prototype: void Rot180MainBmp (const unsigned char *Image)
Description: Rotate the image to 180-degree counter-clockwise orientation in main window. This image should be the one which is displayed by calling the function MainWinDispOn.
Parameters: Image the starting address of bitmap array in system
Return value: None

Rot270MainBmp

Prototype: void (const unsigned char *Image)
Description: Rotate the image to 270-degree counter-clockwise orientation in main window. This image should be the one which is displayed by calling the function MainWinDispOn.
Parameters: Image the starting address of bitmap array in system

Return value: None

Rot0FloatBmp

Prototype: void Rot0FloatBmp (const unsigned char *Image, int startx, int starty)
Description: Rotate the image to 0-degree orientation in floating window. The image should be the one which is displayed by calling the function FloatWinDispOn. This routine can also be used to move the position of floating window on the screen.
Parameters: Image the starting address of bitmap array in system
startx Upper left x-coordinate of floating window
starty Upper left y-coordinate of floating window
Return value: None

Rot90FloatBmp

Prototype: void Rot90FloatBmp (const unsigned char *Image, int startx, int starty)
Description: Rotate the image to 90-degree counter-clockwise orientation in floating window. The image should be the one which is displayed by calling the function FloatWinDispOn. This routine can also be used to move the position of floating window on the screen.
Parameters: Image the starting address of bitmap array in system
startx Upper left x-coordinate of floating window
starty Upper left y-coordinate of floating window
Return value: None

Rot180FloatBmp

Prototype: void Rot180FloatBmp (const unsigned char *Image, int startx, int starty)
Description: Rotate the image to 180-degree counter-clockwise orientation in floating window. The image should be the one which is displayed by calling the function FloatWinDispOn. This routine can also be used to move the position of floating window on the screen.
Parameters: Image the starting address of bitmap array in system
startx Upper left x-coordinate of floating window
starty Upper left y-coordinate of floating window
Return value: None

Rot270FloatBmp

Prototype: void (const unsigned char *Image, int startx, int starty)
Description: Rotate the image to 270-degree counter-clockwise orientation in floating window. The image should be the one which is displayed by calling the function FloatWinDispOn. This routine can also be used to move the position of floating window on the screen.
Parameters: Image the starting address of bitmap array in system
startx Upper left x-coordinate of floating window
starty Upper left y-coordinate of floating window
Return value: None

Example of Display Rotation

Now we have Img1 bitmap array displaying on main window and Img2 bitmap array displaying on floating window. Then we rotate both windows from 0 to 90, 180, 270 and back to 0 in step by step. Afterwards, the floating window is moved around the screen. It is noted that whenever the main window rotates to a particular orientation, the floating window has to follow the same orientation and this is done by programming SSD1906 to do so.

```
MainWinDispOn(Img1);
FloatWinDispOn(Img2, 10, 5);          //Display position (x,y) = (10,5)
Rot90MainBmp(Img1);                   //Rotate 90 degree
Rot90FloatBmp(Img2, 10, 5);
SSD1906Delay(2);                      //Delay loop for 2 seconds
Rot180MainBmp(Img1);                  //Rotate 180 degree
Rot180FloatBmp(Img2, 10, 5);
SSD1906Delay(2);                      //Delay loop for 2 seconds
```

```
Rot270MainBmp(Img1);                    //Rotate 270 degree
Rot270FloatBmp(Img2, 10, 5);
SSD1906Delay(2);                        //Delay loop for 2 seconds
Rot0MainBmp(Img1);                      //Rotate back to 0 degree
Rot0FloatBmp(Img2, 10, 5);
Rot0FloatBmp(Img2, 90, 5);              //Move floating window to (x,y) = (90, 5)
SSD1906Delay(2);                        //Delay loop for 2 seconds
Rot0FloatBmp(Img2, 90, 75);             //Move floating window to (x,y) = (90, 75)
FloatWinDispOff();                      //Turn off floating window
MainWinDispFree();                      //Release memory of main window for future use
```

Virtual Display

When either the height or width of bitmap image is larger than the display physical dimension of LCD panel, the remaining part of the image can be shown by panning (moving horizontally) and scrolling (moving vertically) it. This feature is called virtual display.

VirtMovePic

Prototype: void VirtMovePic (DWORD PosX, DWORD PosY, const unsigned char *Image)
Description: Move the virtual image inside main window.
Parameters: PosX x-coordinate offset position of image with reference to its upper left corner. The value should be a multiple of 32 ÷ bit-per-pixel.
PosY y-coordinate offset position of image with reference to its upper left corner.
Image The starting address of bitmap array in system
Return value: None

Example of Virtual Display

A bitmap array Img1 is displayed on main window. The physical size of this bitmap is larger than that of LCD panel screen size. The image is then move around the screen to enable the other part to be seen. Since the image is displayed with its upper left corner attached to the upper left corner of the LCD screen by MainWinDispOn routine, the image can only be moved either in left or up direction with positive offset values.

```
MainWinDispOn(Img1);
VirtMovePic(40, 0, Img1);               //Move image horizontally 40 pixels in left direction
SSD1906Delay(2);                        //Delay loop for 2 seconds
VirtMovePic(40, 36, Img1);              //Move image vertically 36 pixels in up direction
SSD1906Delay(2);                        //Delay loop for 2 seconds
VirtMovePic(0, 0, Img1);                //Move image back to starting position
SSD1906Delay(2);                        //Delay loop for 2 seconds
MainWinDispFree();                      //Release memory of main window for future use
```

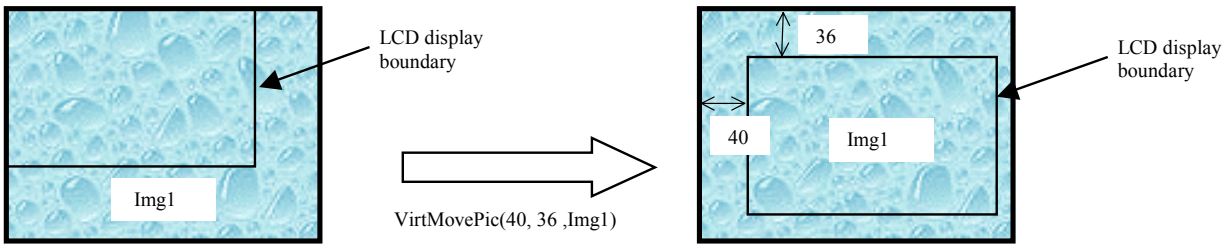


Figure 2: Example of virtual display

Cursor Operation

There are two cursors available in SSD1906 and they can be displayed simultaneously. The maximum size of cursors is 1024x1024 pixels. The available color depth value is 4, 8 and 16 bit-per-pixel only and this value should be the same as that of main window image.

However, the image data architecture of cursor is a little bit different from normal bitmap. For ***any*** color depth setting, the cursor image data structure is the same as 2 bit-per-pixel bitmap, i.e., each pixel occupies two bits of memory space. Hence, there are only four color values (3 colors + transparent) available to be chosen for each pixel.

Table 12: Color setting of cursors

Pixel value	Color
00	Transparent
01	Color 1 defined in color index 1
10	Color 2 defined in color index 2
11	Color 3 defined in color index 3

The color indexes are defined according to different color depth setting. For 4 and 8 bit-per-pixel, the color indexes are actually the LUT entries value of SSD1906. In 16 bit-per-pixel, the colors are defined as direct RGB value with the following format.

Table 13: Color value definition of cursor in 16 bpp

Bit position	15 13	12 8	7 3	2 0
Color component	Green bit 5 to 3	Blue	Red	Green bit 2 to 0

To produce bitmap for cursor, the bitmap is edited with image editor as 4 bit-per-pixel. The maximum available color is 4 only by using the first four color index. Painting the section with color belonging to color index 0 generates transparent. After being saved the changes, the bitmap is converted into C program bitmap array by calling bmpconv.exe.

Cursor1Blink

Prototype: void Cursor1Blink (WORD TotalPeriod, WORD OnPeriod)
Description: Define the blinking period of cursor 1.
Parameters: TotalPeriod Total period used in calculation of blinking period. The unit is in frame.
OnPeriod Cursor turns on period in a “TotalPeriod” time. The unit is in frame.
Return value: None

Cursor2Blink

Prototype: void Cursor2Blink (WORD TotalPeriod, WORD OnPeriod)
Description: Define the blinking period of cursor 2.
Parameters: TotalPeriod Total period used in calculation of blinking period. The unit is in frame.
OnPeriod Cursor turns on period in a “TotalPeriod” time. The unit is in frame.
Return value: None

Cursor1Color

Prototype: void Cursor1Color (WORD Color1, WORD Color2, WORD Color3)
Description: Set the color for cursor 1.
In 4, 8 bit-per-pixel, the arguments are color index pointing to LUT entries of SSD1906.
In 16 bit-per-pixel, the arguments are direct color RGB value definition.

Parameters: Color1 Color 1 index/ RGB value
Color2 Color 2 index/ RGB value
Color3 Color 3 index/ RGB value
Return value: None

Cursor2Color

Prototype: void Cursor2Color (WORD Color1, WORD Color2, WORD Color3)
Description: Set the color for cursor 2.
In 4, 8 bit-per-pixel, the arguments are color index pointing to LUT entries of SSD1906.
In 16 bit-per-pixel, the arguments are direct color RGB value definition.
Parameters: Color1 Color 1 index/ RGB value
Color2 Color 2 index/ RGB value
Color3 Color 3 index/ RGB value
Return value: None

Cursor1DispOn

Prototype: BOOL Cursor1DispOn (const unsigned char *Image, DWORD PosX, DWORD PosY)
Description: Allocate space in display buffer and display cursor 1 on the screen. This function can be used to move the position of cursor. Whenever a rotation occurs in main window, this routine should be called then to update the display rotation of cursor.
Parameters: Image the starting address of bitmap array in system
PosX x-coordinate of cursor with origin at upper left corner of screen.
PosY y-coordinate of cursor with origin at upper left corner of screen.
Return value: TRUE success to display cursor 1 on the screen
FALSE fail to allocate memory to display cursor 1

Cursor2DispOn

Prototype: BOOL Cursor2DispOn (const unsigned char *Image, DWORD PosX, DWORD PosY)
Description: Allocate space in display buffer and display cursor 2 on the screen. This function can be used to move the position of cursor. Whenever a rotation occurs in main window, this routine should be called then to update the display rotation of cursor.
Parameters: Image the starting address of bitmap array in system
PosX x-coordinate of cursor with origin at upper left corner of screen.
PosY y-coordinate of cursor with origin at upper left corner of screen.
Return value: TRUE success to display cursor 2 on the screen
FALSE fail to allocate memory to display cursor 2

Cursor1DispOff

Prototype: void Cursor1DispOff (void)
Description: Turn off cursor 1 and free the memory occupied by it. The sequence of calling Cursor1DispOn and this routine should be taken care such that the memory is properly freed.
Parameters: None
Return value: None

Cursor2DispOff

Prototype: void Cursor2DispOff (void)
Description: Turn off cursor 2 and free the memory occupied by it. The sequence of calling Cursor2DispOn and this routine should be taken care such that the memory is properly freed.
Parameters: None
Return value: None

Example of Cursor Operation

There are two bitmap arrays available for cursors: Cur1 and Cur2, and a bitmap array Img1 for main window display. After displaying these two cursors, we move cursor 2 around the screen. Then the main window is rotated 90 degree. Note that the two cursors have to be updated their positions at this time by program to keep the same orientation as the main window.

```
MainWinDispOn(Img1);                //Display bitmap on main window
Cursor1Blink(100, 80);              //In 100 frames, cursor 1 turn on at 80 frames period
Cursor1Color(38,213,251);           //Set color indexes to LUT entries 38, 213, 251
Cursor1DispOn(Cur1, 64, 80);        //Display cursor 1 at position (x,y) = (64, 80)

Cursor2Blink(20, 20);               //No blinking on cursor 2
Cursor2Color(0,251,3);              //Set color indexes to LUT entries 0, 251, 3
Cursor2DispOn(Cur2, 32, 0);         //Display cursor 2 at position (x,y) = (32, 0)

SSD1906Delay(2);                    //Delay loop for 2 seconds
Cursor2DispOn(Cur2, 32, 90);        //Cursor 2 is moved to position (x,y) = (32, 90)
SSD1906Delay(2);                    //Delay loop for 2 seconds
Cursor2DispOn(Cur2, 120, 90);       //Cursor 2 is moved to position (x,y) = (120, 90)
SSD1906Delay(2);                    //Delay loop for 2 seconds

Rot90MainBmp(Img1);                 //Main window is rotated 90 degree
Cursor1DispOn(Cur1, 64, 80);        //Cursor 1 will be rotated 90 degree automatically
Cursor2DispOn(Cur2, 120, 90);       //Cursor 2 will be rotated 90 degree automatically

SSD1906Delay(2);                    //Delay loop for 2 seconds

Cursor2DispOff();                   //Turn off cursor 2, note the calling sequence
Cursor1DispOff();                   //Turn off cursor 1
MainWinDispFree();                  //Release memory of main window for future use
```

Memory Operation

The memory operations APIs are targeted on 256K bytes display buffer of SSD1906.

MemRemainSize

Prototype: DWORD MemRemainSize (void)
Description: Find out the remain size of display buffer
Parameters: None
Return value: Remain size of available display memory in the unit of byte

MemUsedSize

Prototype: DWORD MemUsedSize (void)
Description: Find out the size of display buffer already allocated
Parameters: None
Return value: Size of total allocated display memory in the unit of byte

Miscellaneous

Disp1906LUT

Prototype: VOID Disp1906LUT (void)
Description: Read 256 LUT entries of SSD1906 and display them on debug console.
Parameters: None
Return value: None

SSD1906Delay

Prototype: VOID SSD1906Delay (int sec)
Description: Create a delay time in seconds by polling vertical non-display status bit
Parameters: sec Number of second of time delay
Return value: None

CheckBigEndian

Prototype: BOOL CheckBigEndian (void)
Description: Check if the microcontroller is a big or little endian system
Parameters: None
Return value: TRUE it is a big endian system.
FALSE it is a little endian system.

DispBlank

Prototype: void DispBlank(BOOL ENABLE)
Description: Either blank or not blank the LCD screen
Parameters: ENABLE TRUE for blank screen
FALSE for not blank screen
Return value: None

DispMainNxNChecker

Prototype: void DispMainNxNChecker(int n, BYTE *LUT1, BYTE *LUT2)
Description: Display a checker board with size of each checker to be N by N
Parameters: n Length of each square checker in pixel
*LUT1 Address of an array which contain red, green and blue color value of LUT for defining the color of first checker of first row.
*LUT2 Address of an array which contain red, green and blue color value of LUT for defining the color of second checker of first row.
Return value: None
Note: The content of LUT1 or LUT2 array contains
LUTX[0] = red color, LUTX[1] = green color, LUTX[2] = blue color, where X = 1 or 2

Procedure to port SSD1906 API to a system

SSD1906 API is written in C language. Hence, it is not difficult to port it to another platform that supports C compiler. This section describes the steps to change the program to serve for a new system.

Microcontroller Register Access

The McuInit routine is used to initialize the microcontroller to able to interface with SSD1906. The first thing to do is enable the program to read/write the control registers of MCU so as to configure it to communicate with SSD1906. The example program m68vz328.c and vr4181.c provide sample routines on how to perform these functions. For most MCU platform, the only variable required to be changed in order to make these routines work is "_BaseAdr". This is the base address used to calculate the final address of control registers of MCU.

For example, all the registers have address of 0xFFFFF000 + index in Dragonball MC68VZ328. "_BaseAdr" is set to 0xFFFFF000 in m68vz328.h file. Afterwards, we can use the API "RdMcuWordReg (0x112)" to read the register CSB which has the address of 0xFFFFF112.

Data Size Setting

SSD1906 API requires the definitions of BYTE (8-bit byte), WORD (16-bit word) and DWORD (32-bit double word) for size declaration of program variables. The example program header m68vz328.h and vr4181.h demonstrate how they are defined. Each MCU platform has its own meaning of the length of char, int, long which are used to set the above definitions and it can be found in MCU or program compiler user menu.

Host bus interface

The MCU is configured to deploy one of the available CPU interfaces of SSD1906, namely, Generic #1, Generic #2, Motorola MC68K or Motorola DragonBall MC68EZ/VZ/SZ328. In SSD1906, this is done by setting the CF[5:0] pins for the appropriate host bus. Please refer to Table 8 Configuration of dip switch S2.

In microcontroller side, the dedicated I/O function pins are configured to reserve for interface with SSD1906. These I/O pins name varies for different systems. However, the main functionalities of them are the same. In below, the different host interface configuration is introduced.

Generic #1

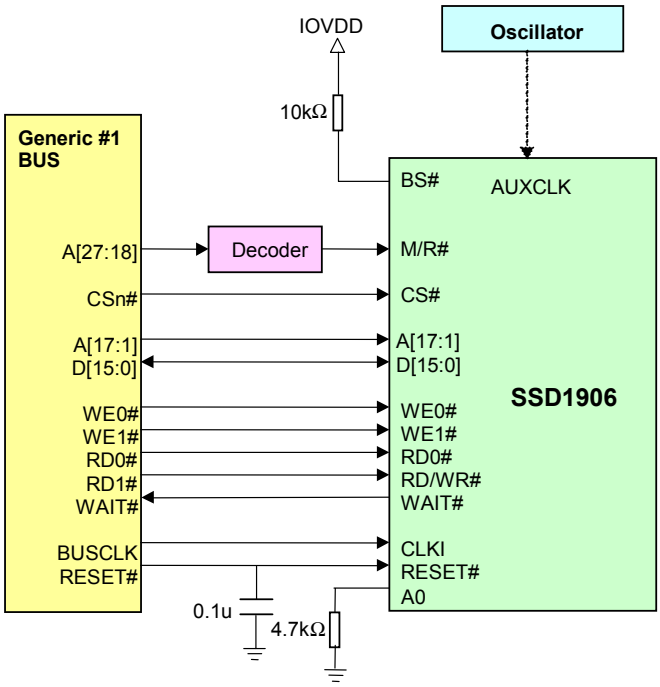


Figure 3: Generic #1 interface connection diagram

Generic #1 is a SRAM type interface. It uses control pins and communication protocol similar to SRAM one. Therefore, if the microcontroller supports SRAM, it can interface with SSD1906 by using Generic #1 configuration.

Here are some properties of Generic #1 host bus interface:

- WE0# is driven low for low byte write cycle
- WE1# is driven low for high byte write cycle
- RD0# is driven low for low byte read cycle
- RD/WR# is driven low for high byte read cycle
- WAIT# is driven low to inform MCU to wait until data is ready (read cycle) or accepted (write cycle) and WAIT# is driven high then

The following table shows the setting of CF[5:0] logic for Generic #1 MCU interface.

Table 14: CF[5:0] setting for Generic #1 interface

SSD1906 pin	1 (High)	0 (Low)
CF[2:0]	011\b => Generic #1 interface	
CF3	GPIO pins set as inputs at power on	GPIO pins set as HR-TFT output signals.
CF4	Big Endian bus interface	Little Endian bus interface
CF5	Active high WAIT#	Active low WAIT#

Generic #2

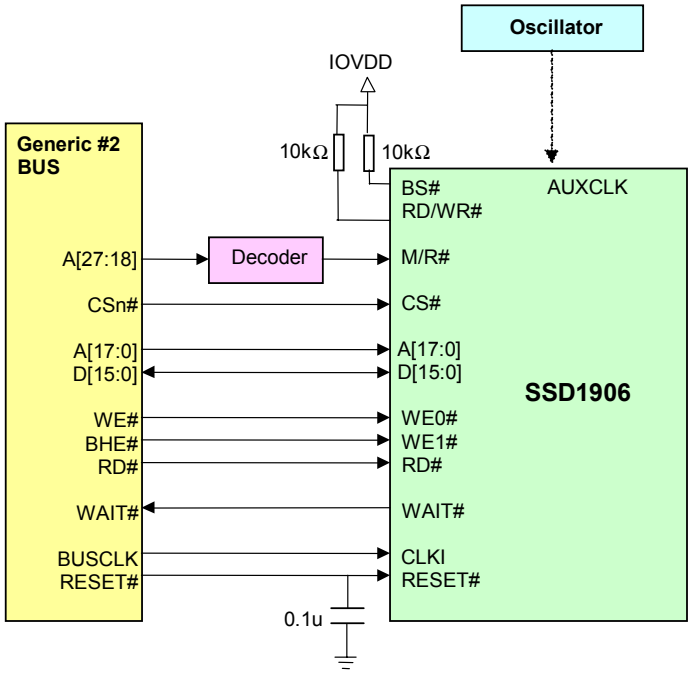


Figure 4: Generic #2 interface connection diagram

In Generic #2, the interface pins and communication protocol is similar to an ISA type bus interface. Hence, if the microcontroller supports ISA bus device, it can interface SSD1906 in Generic #2 mode.

Properties of Generic #2 host bus interface are:
- WE0# is driven low for every write cycle
- RD# is driven low for every read cycle
- WE1# is driven low for high byte *read* or *write* cycle.
- Selection of either read/write high byte or word is done by controlling WE1# and A0 pins according to the following table.

Table 15: High byte and word read/write signals for Generic #2 interface

	WE1#	**A0**
High byte read/write	0	1
Word read/write	0	0

The following table shows the setting of CF[5:0] logic for Generic #2 MCU interface.

Table 16: CF[5:0] setting for Generic #2 interface

SSD1906 pin	**1 (High)**	**0 (Low)**
CF[2:0]	100\b => Generic #2 interface	
CF3	GPIO pins set as inputs at power on	GPIO pins set as HR-TFT output signals.
CF4	Big Endian bus interface	Little Endian bus interface
CF5	Active high WAIT#	Active low WAIT#

Hitachi SH-3/SH-4

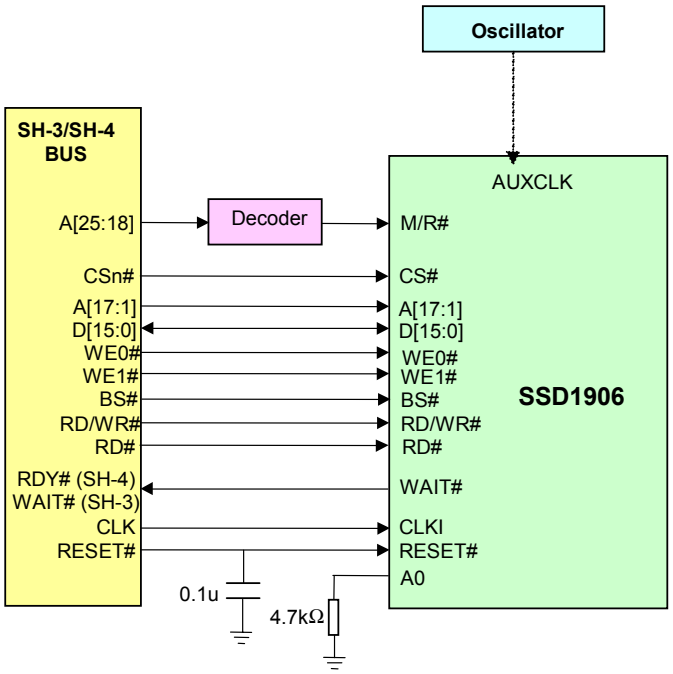


Figure 5: Hitachi SH-3/SH-4 interface connection diagram

SH-3/SH-4 host bus interface supports Hitachi SuperH SH-3 and SH-4 series microprocessor. Following are the properties of this communication protocol.

- WE0# is driven low for every low byte (D7 – D0) write cycle
- WE1# is driven low for every high byte (D15 – D8) write cycle
- BS# is driven low at every start of bus cycle
- RD/WR# is driven low to indicate write and driven high for read data
- RD# is driven low in read cycle
- For SH-3, WAIT# is driven low in wait cycle
- For SH-4, WAIT# is driven high in wait cycle

The following table shows the setting of CF[5:0] logic for SH-3/SH-4 interface.

Table 17: CF[5:0] setting for Hitachi SH-3/SH-4 interface

SSD1906 pin	1 (High)	0 (Low)
CF[2:0]	000\b => Hitachi SH-3/SH-4 interface	
CF3	GPIO pins set as inputs at power on	GPIO pins set as HR-TFT output signals.
CF4	Big Endian bus interface	Little Endian bus interface
CF5	Active high WAIT# (for SH-4)	Active low WAIT# (for SH-3)

Motorola MC68K

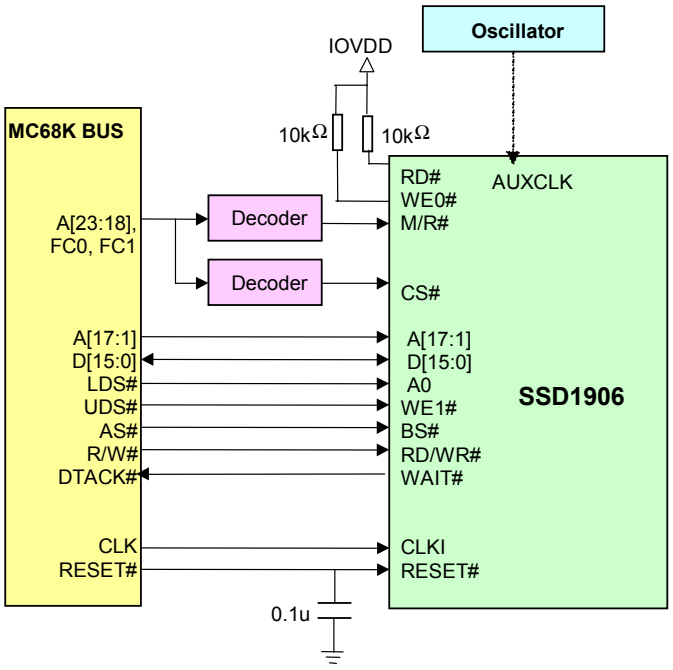


Figure 6: Motorola MC68K interface connection diagram

Motorola MC68K host bus interface supports Motorola M68000 series CPU. Following are the properties of this communication protocol.

- A0# is driven low for every low byte read/write cycle
- WE1# is driven low for every high byte read/write cycle
- BS# is driven low when address on address bus is valid
- RD/WR# is driven low in write cycle and driven high in read cycle

The following table shows the setting of CF[5:0] logic for MC68K interface.

Table 18: CF[5:0] setting for Motorola MC68K interface

SSD1906 pin	1 (High)	0 (Low)
CF[2:0]	001\b => MC68K interface	
CF3	GPIO pins set as inputs at power on	GPIO pins set as HR-TFT output signals.
CF4	1 => Big Endian bus interface	
CF5	Active high WAIT#	Active low WAIT#

Motorola Dragonball MC68EZ/VZ/SZ328

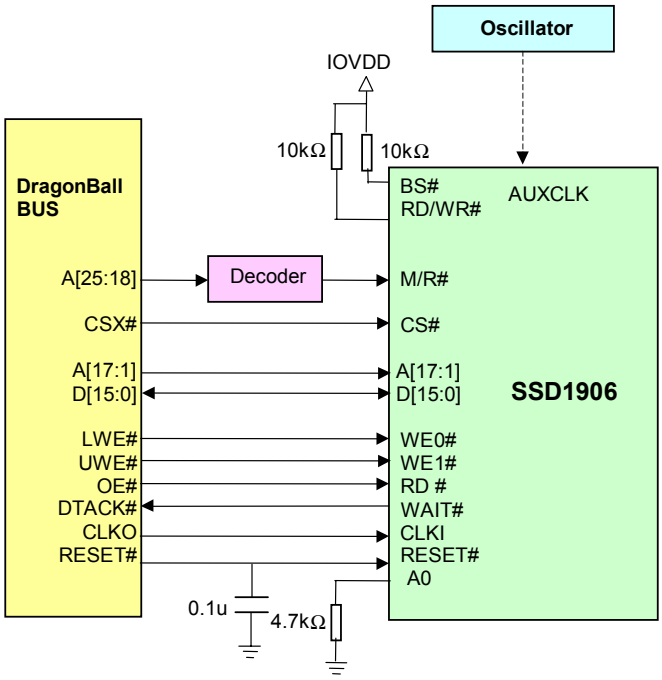


Figure 7: Motorola MC68EZ/VZ/SZ328 interface diagram

This interface is dedicated designed for Dragonball MC68EZ/VZ/SZ328 microcontroller. The properties of this interface are:

- WE0# is driven low for low byte write cycle. Data is available at D[15:8] bus.
- WE1# is driven low for high byte write cycle. Data is available at D[7:0] bus.
- RD# is driven low for every 16-bit word read cycle. In DragonBall interface, there is no byte read.

The following table shows the setting of CF[5:0] logic for Dragonball interface.

Table 19: CF[5:0] setting for Motorola MC68EZ/VZ/SZ328 interface

SSD1906 pin	1 (High)	0 (Low)
CF[2:0]	110\b => Dragonball interface	
CF3	GPIO pins set as inputs at power on	GPIO pins set as HR-TFT output signals.
CF4	1 => Big Endian bus interface	
CF5	1 => Active high WAIT#	

Registers & Display Memory Mapping

Since SSD1906 control registers and display buffer are memory mapped, MCU system is required to reserve sections of its memory space to achieve this purpose. Normally, a microcontroller has several chip select pins to interface to different peripheral devices. Each chip select occupies a certain memory space. Once a slice of space is allocated for SSD1906, a chip select (CS#) pin of MCU is assigned for this address space and it is used to select SSD1906 for communication.

In the reserved MCU memory space for SSD1906, it is divided into control registers and display buffer area. Therefore, address pin(s) can be used to connect to M/R# pin of SSD1906 to select between memory and register address space in every access. In below, an example of how M/R# connected to address pin is shown.

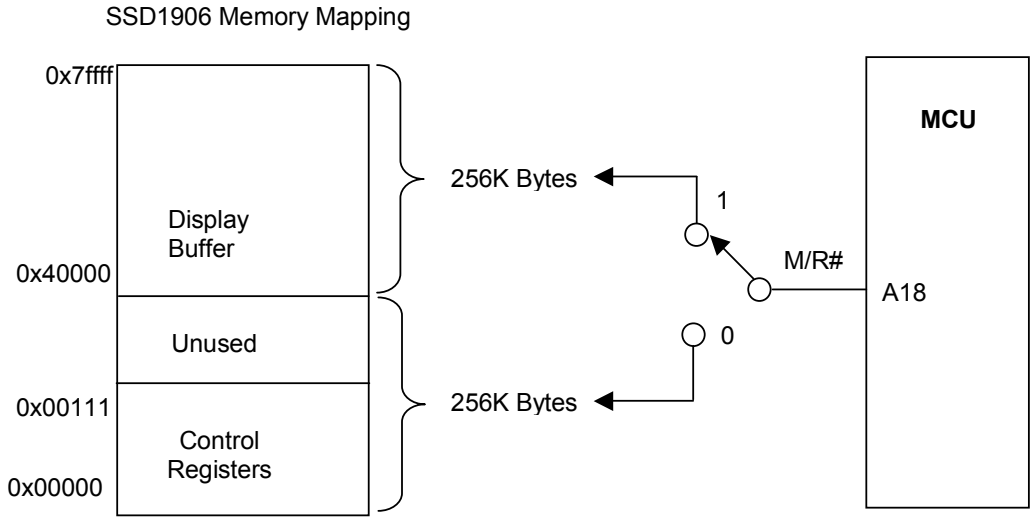


Figure 8: SSD1906 Memory Mapping Example

Actually, the total memory space occupied by SSD1906 is found by
Size of control registers space + Size of display buffer space
= 0x112 + 256k bytes = 262418 bytes = 256.27k bytes

For the simplicity of address pin connection to M/R# pin without address decoder, the address of the space occupied by control registers and display buffer are **aligned** such that only one address pin is connected to M/R# pin. In the above example, control registers of SSD1906 are allocated with lower 256K bytes while display buffer occupies upper 256K bytes of MCU memory address space.

If control registers takes up address 0x00000, display memory starts at 0x40000 and range to 0x7FFFF to occupy 256K bytes display buffer space. When accessing display memory, address bit 18 should be set to 1 since value 0x40000 is equivalent to bit 18 to be 1. Therefore, MCU can use address pin A18 connecting to M/R# pin for selecting either accessing control registers or display memory. This configuration is suitable for microprocessor with available continuous address space of 256K bytes.

This setting is made effective by defining the addresses in SSD1906 API inside the program header ssd1906.h. Let take the above example, the starting address of control registers is 0x00000 and the constant "RegAddress" should be set with this value. Another constant "MemAddress" is set with value 0x40000 like below.

```
#define RegAddress    0x00000     // Registers starting address
#define MemAddress    0x40000     // Display buffer starting address
```

For various type of MCU platform, the available memory space and its starting addresses are different and it'd better to consult the MCU user menu for possible settings.

LCD Interface

SSD1906 can interface to different types of LCD panels include STN, CSTN, TFT, HR-TFT in various data widths. There are several changes to be done in SSD1906 API program for interface a new LCD panel.

In program header lcd.h, the new panel is defined by specifying its panel width and panel height with constant values PANEL_W and PANEL_H respectively. This program header allows more than one type of LCD panel to be defined and this arrangement facilitates SSD1906 API to be changed to interface with other type of panels easily.

Another program header file to be changed is lcdinfo.h. It contains the control registers values which are written into SSD1906 by LcdInit( ) routine during start up of the SSD1906 API program. In this header, it is allowed to define more than one type of LCD panel control registers settings. These settings are enabled by the LCD panel definition in program header lcd.h.

Below is an example of the definition of LCD panel control registers values in lcdinfo.h file.

```
#ifdef TFT
        {REG_PCLK_CONFIG               ,0x12},   // Reg    5h

        {REG_PANEL_TYPE                ,0x41},   // Reg   10h
        {REG_MOD_RATE                  ,0x00},   // Reg   11h
        {REG_HORIZ_TOTAL               ,0x22},   // Reg   12h
        {REG_HDP                       ,0x1d},   // Reg   14h

        {REG_HDP_START_POS0            ,0x0c},   // Reg   16h
        {REG_HDP_START_POS1            ,0x00},   // Reg   17h

        {REG_VERT_TOTAL0               ,0xc7},   // Reg   18h
        {REG_VERT_TOTAL1               ,0x00},   // Reg   19h
        {REG_VDP0                      ,0x9f},   // Reg   1ch
        {REG_VDP1                      ,0x00},   // Reg   1dh

        {REG_VDP_START_POS0            ,0x14},   // Reg   1eh
        {REG_VDP_START_POS1            ,0x00},   // Reg   1fh
        {REG_HSYNC_PULSE_WIDTH         ,0x87},   // Reg   20h
        {REG_HSYNC_PULSE_START_POS0    ,0x00},   // Reg   22h
        {REG_HSYNC_PULSE_START_POS1    ,0x00},   // Reg   23h
        {REG_VSYNC_PULSE_WIDTH         ,0x80},   // Reg   24h

        {REG_VSYNC_PULSE_START_POS0    ,0x00},   // Reg   26h
        {REG_VSYNC_PULSE_START_POS1    ,0x00},   // Reg   27h

#endif
```

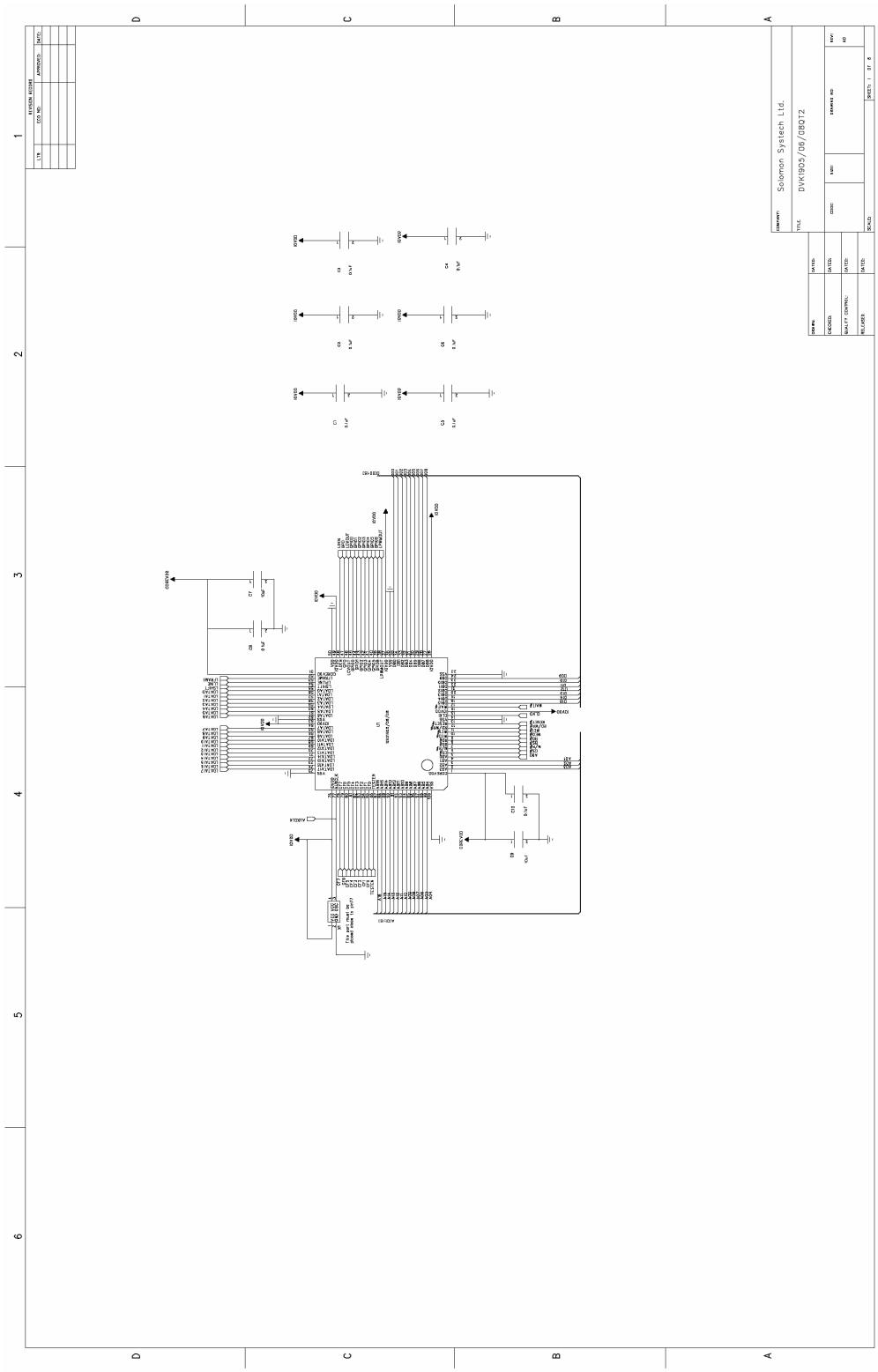
In each bracket, the first parameter is the control register address. For example, `REG_PCLK_CONFIG` is the address of pixel clock configuration register. These constant addresses are defined in ssd1906.h header file. The second parameter inside the bracket is the value to be set in this register. All the registers embraced by #ifdef TFT to #endif are related to LCD panels properties settings. Programmers should change all the above registers setting according to timing diagram and specification inside the datasheet of the LCD panel. The detailed description of these control registers can be found in section 7 Registers of SSD1906 specification.

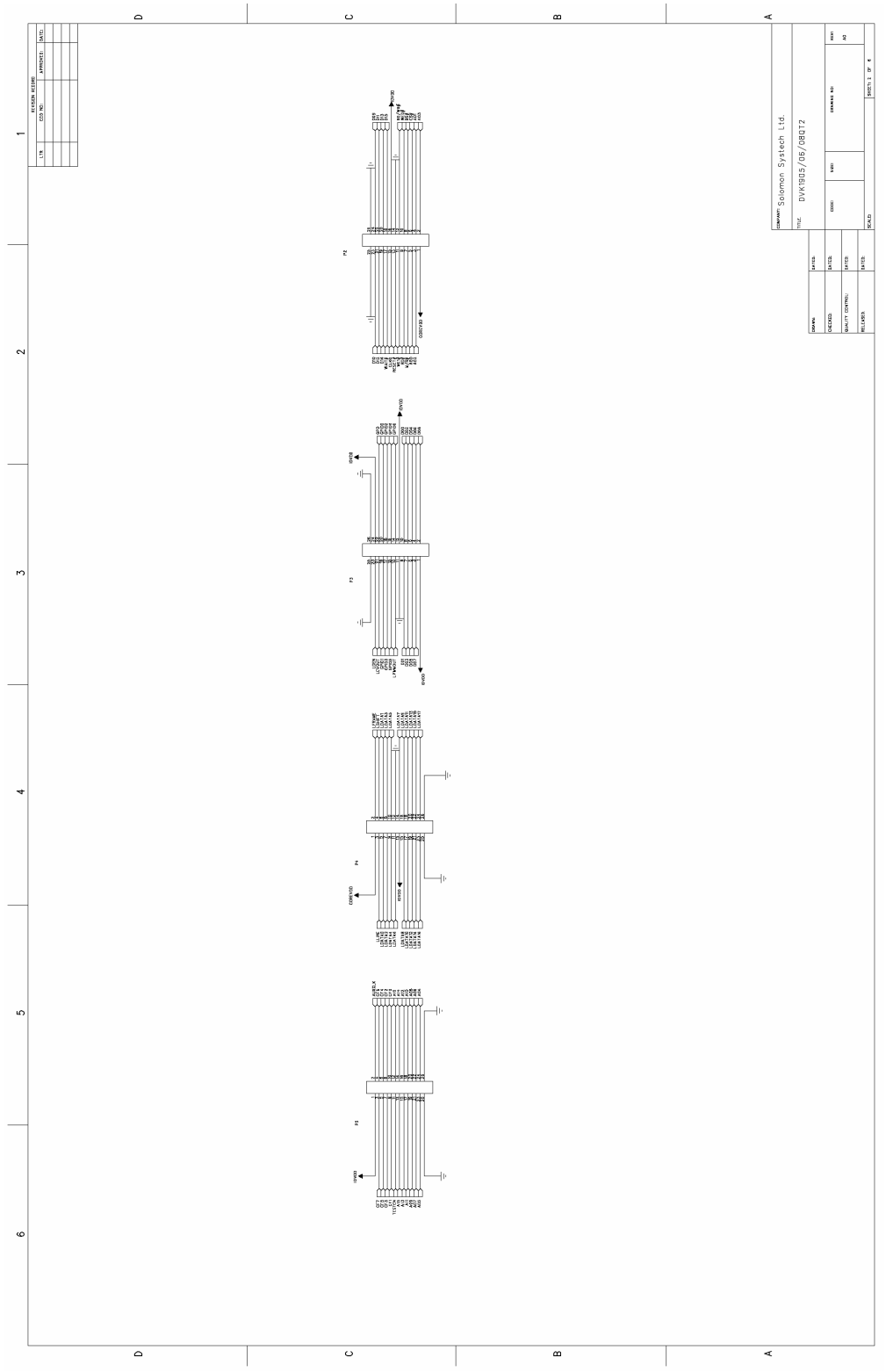
At the last section of lcdinfo.h header file, there are two items deserved concern.

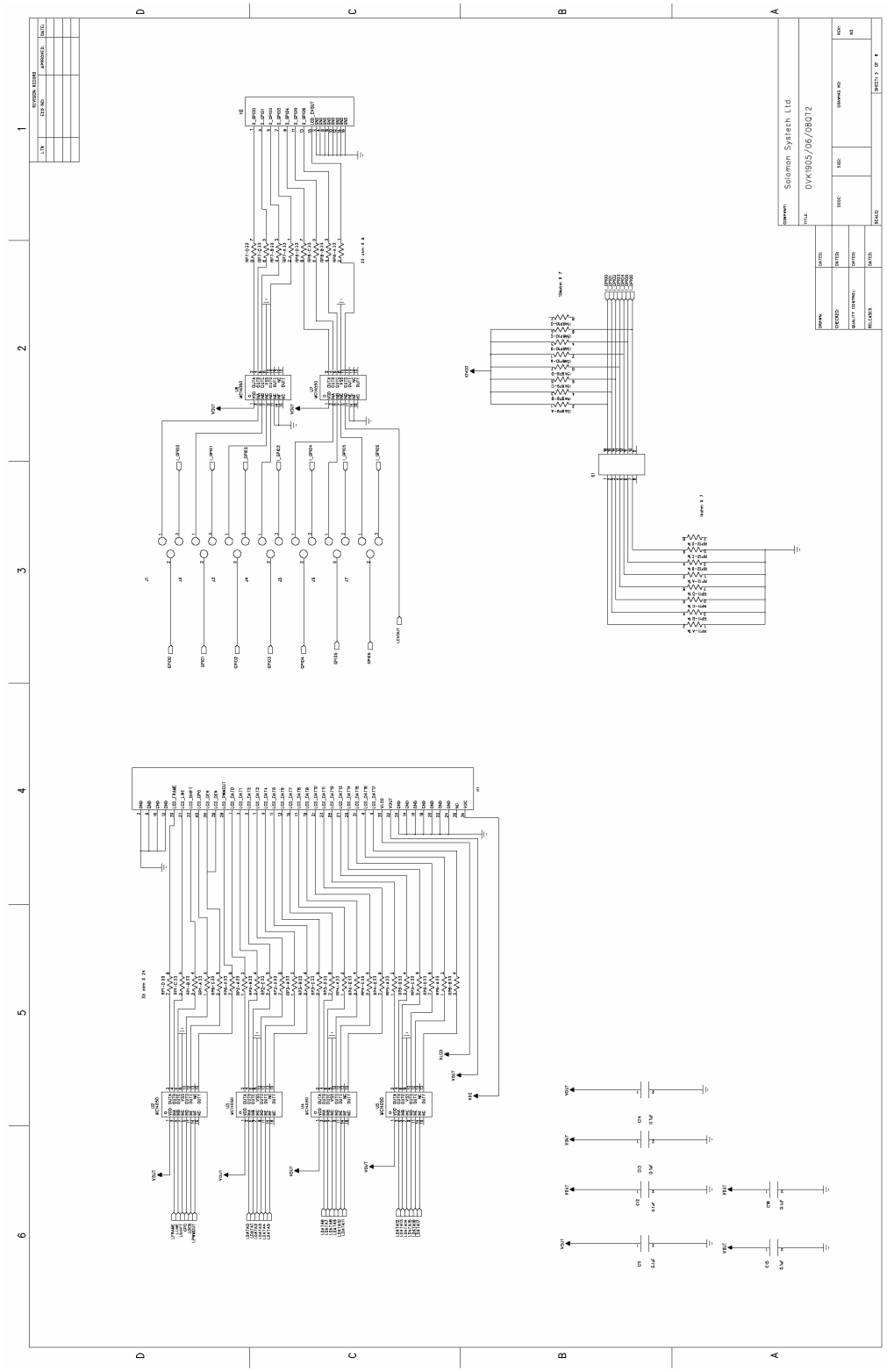
```
16000,                  // ClkI (kHz)
6000,                   // AUXCLK (kHz)
```

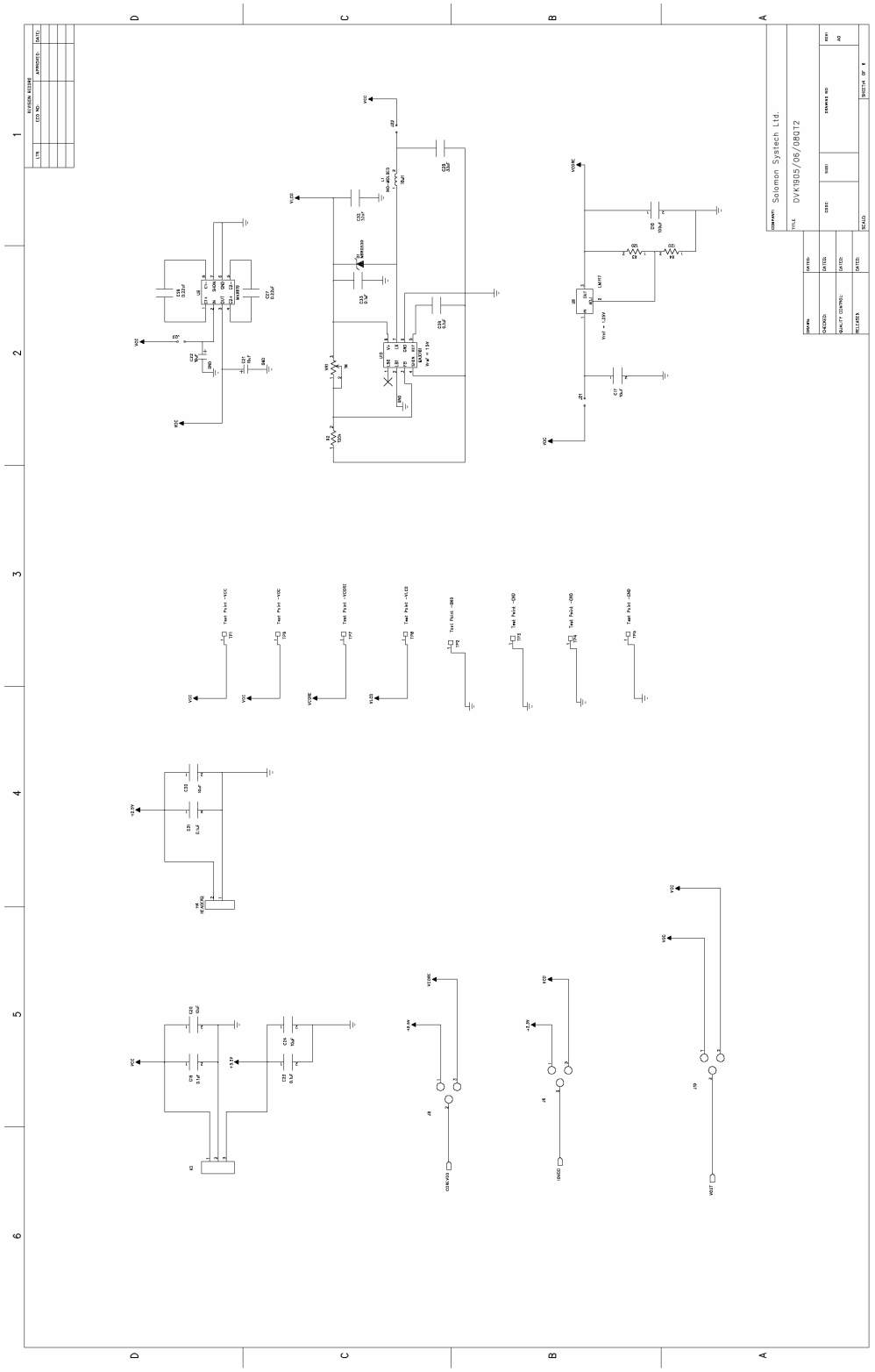
It defines the CLKI and AUXCLK frequency in unit of kHz. These values are set for the calculation of timing in SSD1906 API routine like SSD1906Delay( ).

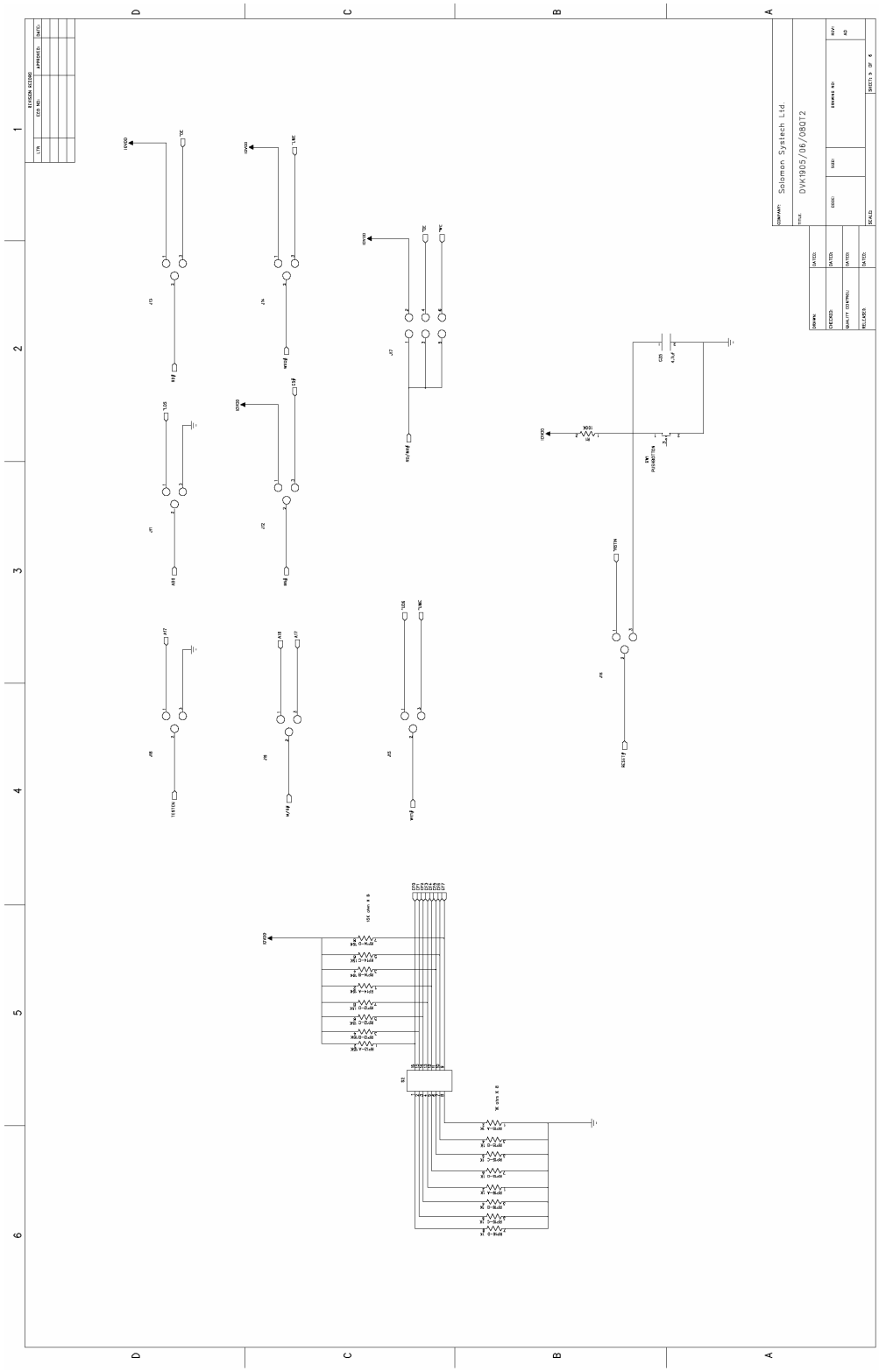
Schematic of DVK1906QT2-1-A1 Development board

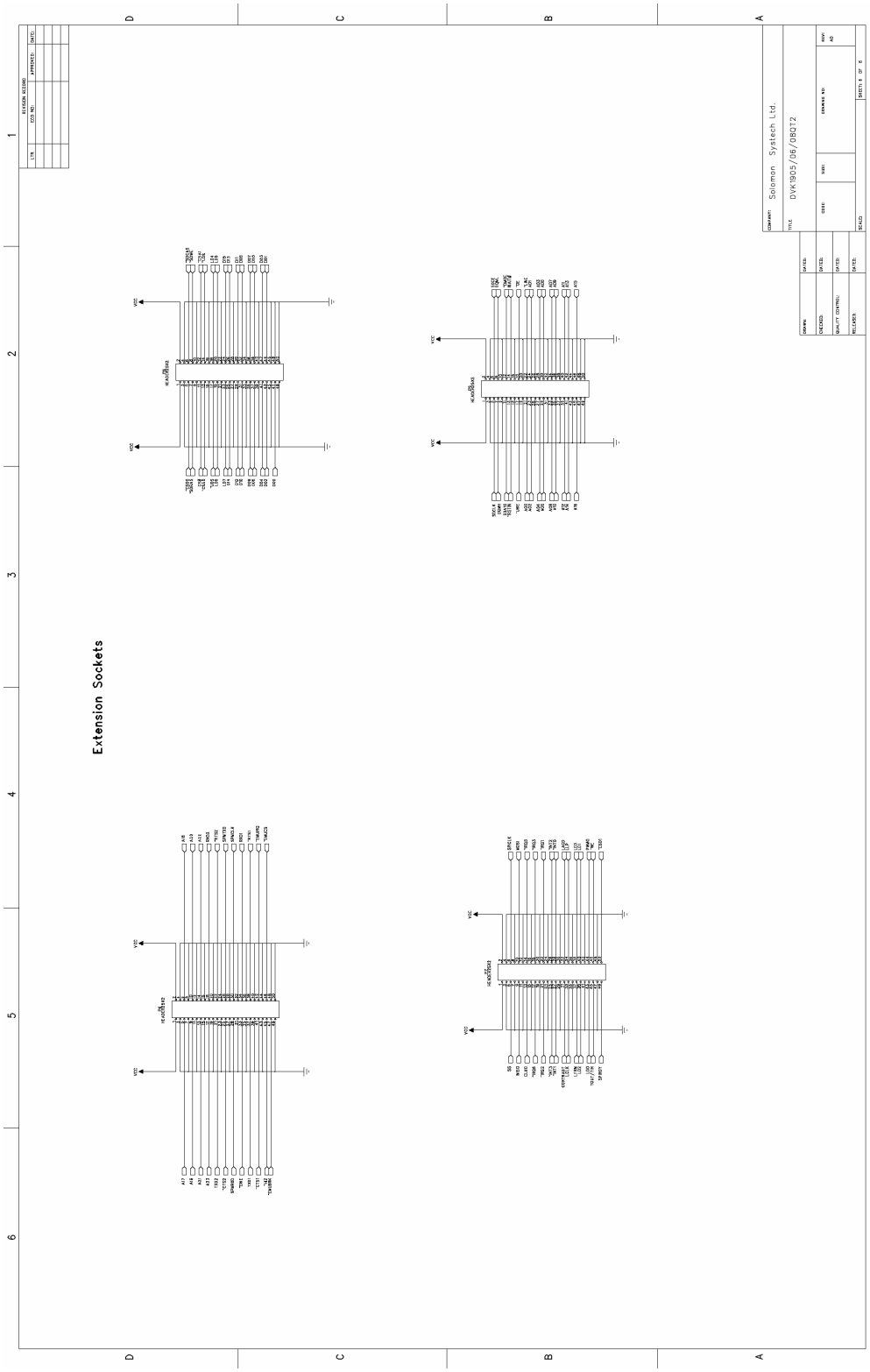





Solomon Systech Ltd.
DVK1905/06/08QT2




Solomon Systech Ltd
DVK1905/06/08T2


Extension Sockets
Solomon Systech Ltd
DVK1905/06/08Q72