Getting Started with GT-CP: Connect to Microcontroller

Scope

This document is intended to provide general instructions on connecting a GT-CP series module to a microcontroller using any of the available interfaces (UART, SPI, and I^2C). Time-wasting troubleshooting areas will also be addressed to help accelerate evaluation/development time. Texas Instrument's MSP430™ (MSP-EXP430F5529LP) microcontroller will be used as the host microcontroller as it is relatively low cost and easy to translate concepts and code to other microcontrollers. Other host boards can be used to interface with GT-CP as long as they support the necessary serial interfaces.

Table of Contents

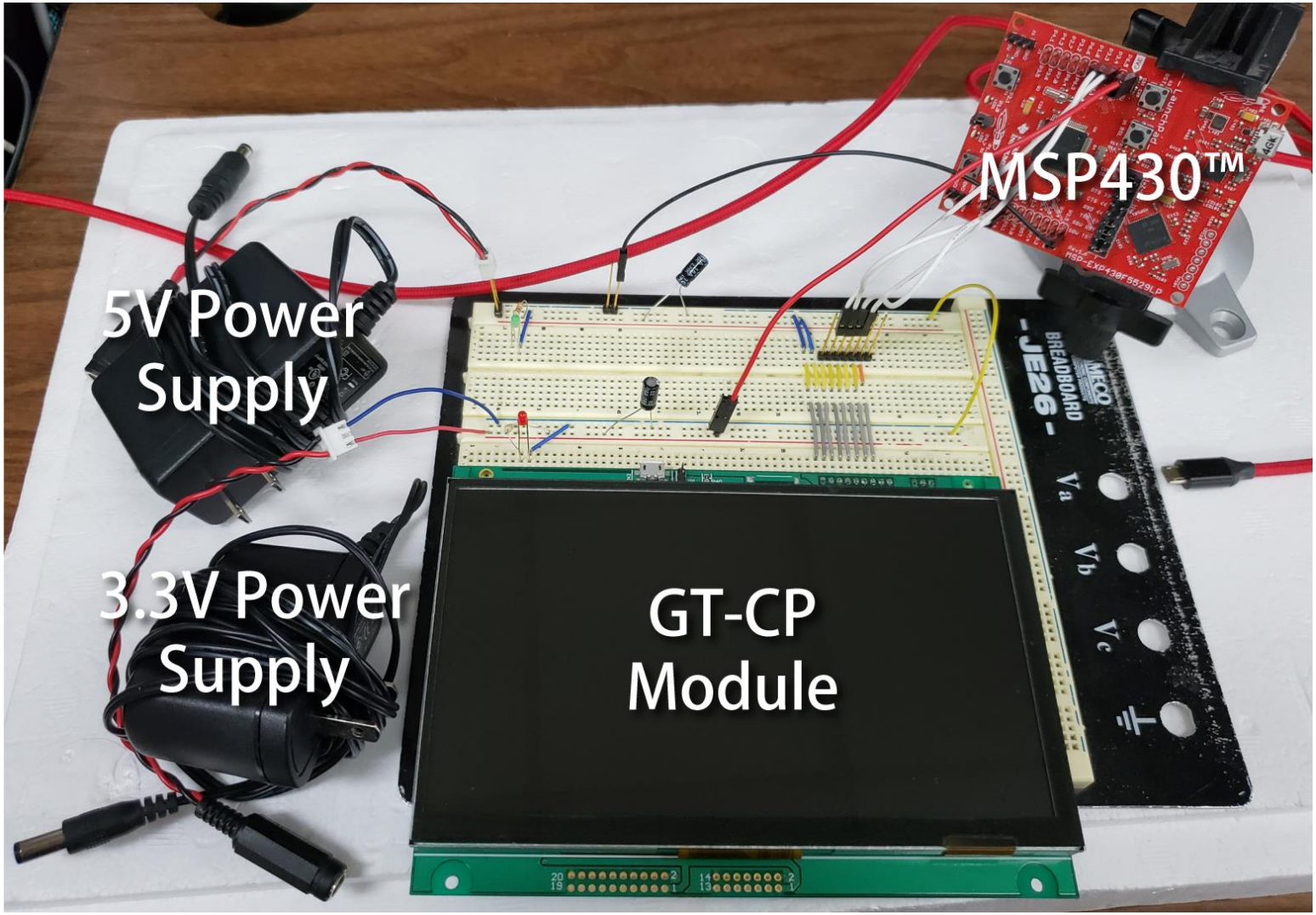
Materials

- MSP-EXP430F5529LP
  - Micro USB cable (included in Texas Instrument's MSP430™ kit)
- Breadboard
- A GT-CP module
  - GTWV070C3A00PA
  - GTWV050C3A00PA
  - GTWQ043C3A00PA
- Pin headers (2.54mm pitch)
- 5V 1.5A 7.5W AC/DC power adapter
- 3.3V 1A AC/DC power adapter
- PC with at least Windows 7 installed
- Female/female cable jumpers
- Jumper wire kit
- 330Ω Resistors
- 10KΩ Resistors (for I^2C interface)
- 10uF 50V electrolytic capacitors
- LEDs
  - Red 19mcd - 3mm 1.85Vf @ 10mA (B4303F1)
  - Green 19mcd – 3mm 2Vf @10mA (B4303F5)

NOTE: Power adapters must be regulated within the respective module's hardware specification.

Material Setup Image

This image shows all electronic materials (besides a Windows PC and pullup resistors for I^2C) using the UART interface with the power supplies unplugged.



Command Set Overview

The GT-CP series of modules operate based on a pre-set group of commands. These commands tell the module to do different actions, display text/graphics, adjust touch sensitivity, change internal settings, etc. This helps reduce the amount of instructions needed to operate the display by letting the display perform graphical calculations, store images, adjust touch options, etc. Typical ASCII commands are compatible as well. Commands must be sent serially via UART, SPI, or I^2C (or through USB).

Concept

Each command is constructed with a group of header bytes that determine the command that is going to be used. For more complex commands, additional bytes are used to select specific operations/parameters within the command. The entire command set can be seen in the GT-CP software spec.

Command Examples

The hexadecimal notation seen in the GT-CP software specification is “00h, 01h, 02h, etc.” This document gives programming examples in C and will use the C-style hexadecimal notation “0x00, 0x01, 0x02, etc.”.

Simple Command

Brightness level change command:

This command changes the display brightness.
Code: *0x1f, 0x58, n*
n = brightness level

$$Brightness\ percentage = \frac{n}{255} * 100$$

So, a brightness percentage of 25% can be obtained by sending: *0x1f, 0x58, 0x40.*

Complex Command

Downloaded image display:

This command recalls an image from module memory.
Code: *0x1f, 0x28, 0x66, 0x10, m, aL, aH, aE, xSL, xSH, xL, xH, yL, yH, fmt*

m = Memory location select
aL = Image address, lower byte
aH = Image address, upper byte
aE = Image address, extension byte
xSL = Image defined width, lower byte
xSH = Image defined width, upper byte
xL = Image display width, lower byte
xH = Image display width, upper byte
yL = Image display height, lower byte
yH = Image display height, upper byte
fmt = Image format

So, if we want to display an 800x480 BMP image that is stored in FROM at address 0x01000000, the following bytes should be sent: *0x1f, 0x28, 0x66, 0x10, 0x11, 0x00, 0x00, 0x00, 0x20, 0x03, 0x20, 0x03, 0xe0, 0x01, 0xf0*.

Code Library

Noritake offers a free GT-CP code library for the M430F5529 microcontroller launch pad development board (MSP-EXP430F5529LP). This library configures the UART, SPI, I^2C interfaces on the microcontroller to properly interface with a GT-CP module. It also has functions for each available GT-CP command that can be used with any of the three serial interfaces.

Download

This GT-CP code library and example code can be downloaded here.

Interfacing

The code library contains a library file for each available interface:

- GTCP_UART.c
- GTCP_I2C.c
- GTCP_SPI.c

Each interface file utilizes the microcontroller's built-in circuitry to communicate using UART, SPI and I^2C. Each file changes the necessary registers to properly configure the communication protocol to conform to the GT-CP's' requirements.

Interface.c is also included for common interface elements like RESET.

Commands

The code library contains a library file for all GT-CP commands:

- GTCP.c

This file simplifies each display command into its own function call.

Interfacing Details

The GT-CP modules can be controlled serially via UART, I^2C, and SPI. The core interfacing details are the same across the currently available GT-CP modules.

GT-CP Connector Pinout (CN9)

Pin No.	Signal Name			Function	Direction
	UART	SPI	I^2C		
1	/RESET	/RESET	/RESET	Reset	Input
2	-	SCK	SCL	Serial Clock	Input
3	RXD	MOSI	SDA	Data Receive	Input/ Output
4	DTR	SBUSY	SBUSY	Display Busy	Output
5	DSR	SSEL	-	Host Busy	Input
6	TXD	MISO	-	Data Send	Output
7	TRDY	TRDY	TRDY	Target Ready	Output
8	VCC	VCC	VCC	Power Supply (5V)	-
9	GND	GND	GND	Ground	-

Jumper Configuration

The module's serial interfaces are selected via hardware jumpers (J5 and J6) located on the back of the display module as seen here. These jumpers must be soldered in an OPEN or SHORT state in order to change the selected interface.

	UART	SPI	I^2C
J5	SHORT	OPEN	SHORT
J6	SHORT	OPEN	OPEN

Buffer Capacity

The GT-CP modules have a limited buffer capacity that must be monitored when transmitting a large number of bytes to the module.

Receive Buffer	2047 bytes
Transmit Buffer	127 bytes

SBUSY/DTR Signal Change

The SBUSY/DTR signal can be used to monitor the receive buffer status as it will go high when the buffer has 128 bytes or more and go back to low if it has 62 bytes or less. It is good practice to monitor this signal when transmitting any number of bytes to the module.

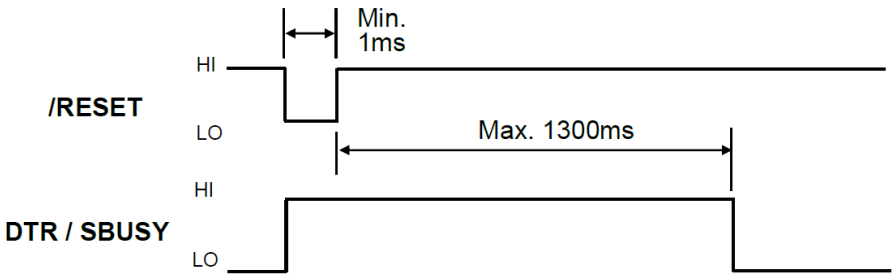
SBUSY Change	'L' (READY) -> 'H' (BUSY)	'H' (BUSY) -> 'L' (READY)
Receive buffer space	62 bytes or less	128 bytes or more

/RESET Signal

The /RESET signal can be used to perform a hardware reset. This is a 3.3V active-low signal.

If connected to a host board, the /RESET pin must be in an open-drain output configuration. If an open-drain output configuration is not available, then set the /RESET pin as an input until a reset pulse is required. If unused, it may be left floating and disconnected.

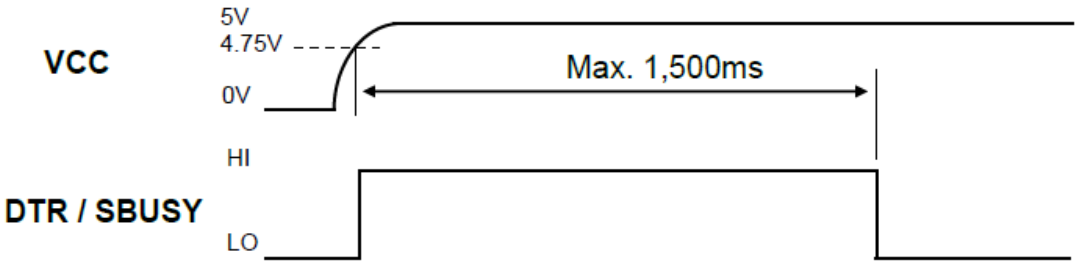
When performing a hardware reset, adhere to the following timing diagram:



After a hardware reset is performed, wait for 1300ms to ensure the module is fully initialized and ready to receive data.

Power-On Reset

When the GT-CP module receives power, wait for 1500ms to let the module fully initialize, otherwise data may be lost. For complete certainty that the module is ready to receive data, the DTR/SBUSY signal must be monitored. If the signal is LO, then the module is ready to receive data.



TRDY Signal Change

The TRDY signal is useful when reading data from the module. It signifies if data is present in the module's transmit buffer and shows when the last byte of data is going to be sent. Typically, touch data is read from the module so it is very important to know when touch data is ready to be read and analyzed. TRDY does not have the exact same behavior across the module's three interfaces. In I^2C, TRDY can only change in-between byte sequences. This makes the signal able to signify where the end of the data stream is. In UART and SPI, TRDY can change at any time and lessens the usefulness of the TRDY signal.

TRDY State	READY ('H')	EMPTY ('L')
Condition	Data in transmit buffer	Transmit buffer is empty

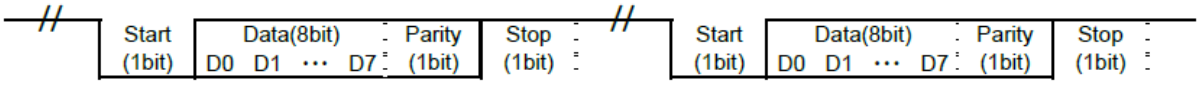
Type of Interface	**When does the TRDY signal change?**		**Note**
	Low to High	**High to Low**	
UART	The transmit buffer was empty, but one or more bytes have been placed in the transmit buffer, queued for transmission.	The final byte that was in the transmit buffer has been transferred to the interface hardware for transmission to the host. The transmit buffer is now empty.	Signal will change at any time depending on the transmit buffer state regardless of data transmit timing.
SPI			Signal will change at any time depending on the transmit buffer state regardless of data transmit timing.
I^2C	The transmit buffer was empty, but: - One or more bytes have been placed in the transmit buffer, queued for transmission, while the I^2C interface was idle or in receive state, or - The I^2C interface has exited the transmit state and there is at least one byte queued in the transmit buffer.		Signal will only change in-between bytes.

UART

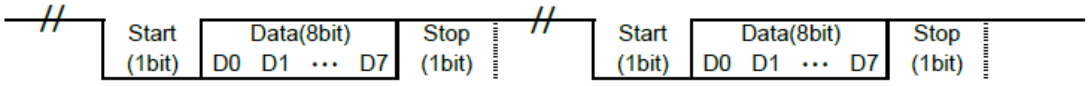
The UART interface is very straightforward to implement at the module's default values. As long as the host controller is communicating using 38400bps, no parity, and one start and stop bit, basic communication should be no problem.

Baudrate	4800 to 115200bps (set via Jumper and/or Memory Switch) **Default setting: 38400bps**
Parity	None, Even, Odd (set via Memory Switch) **Default setting: None**
Format	Start (1 bit) + Data (8 bits) + Parity (0 or 1 bit) + Stop (1 bit)
Communication Control Signal	DSR, DTR, /TRDY

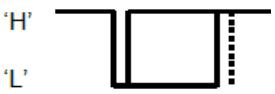
Data Frame (When using parity bit)



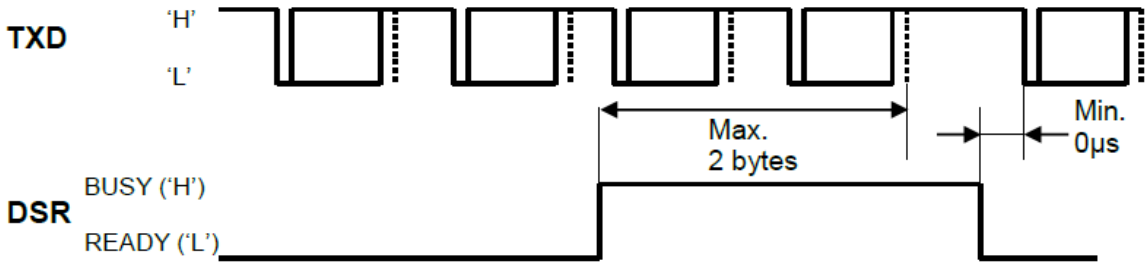
Data Frame (When parity bit is not used)



Each data frame is represented in the following signal diagrams as:

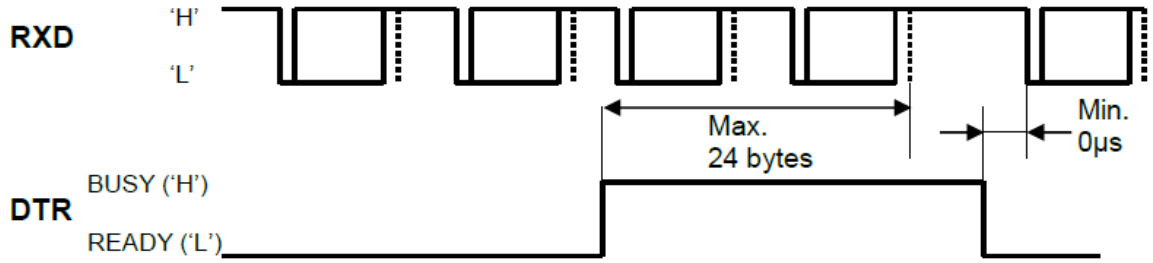


Write Signal Diagram



The HBUSY/DSR signal tells the display module if the host board is busy. If DSR is in its BUSY state, the display module will stop transmitting data until DSR has returned to its READY state. This is necessary to ensure that there is no data loss when reading an excessive amount of data from the display module.

Read Signal Diagram



The SBUSY/DTR signal is very important when sending large amounts of data as it signals (with a high voltage state) when the display module has more than 128 bytes in it's receive buffer. Once it's receive buffer has been reduced below 62 bytes, SBUSY/DTR will change to its low voltage state.

TRDY Signal

The TRDY signal signifies if data is present in the module's transmit buffer. When this signal goes low, there is no data in the transmit buffer. The signal can change anywhere within the data sequence, so it is difficult to determine when the last byte will be transmitted.

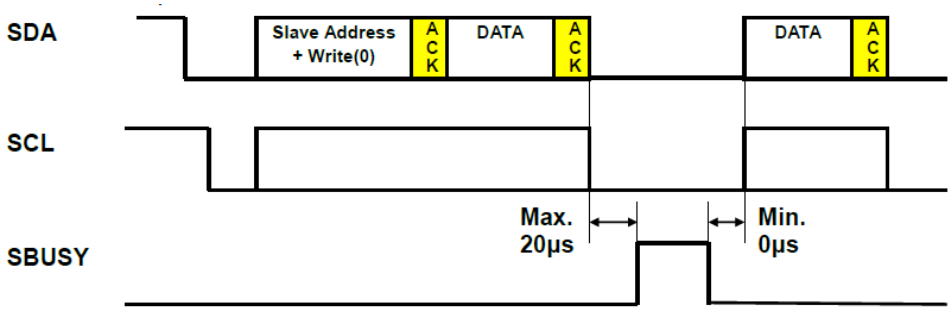
I²C

The GT-CP module can interface with the industry standard I²C protocol with a maximum baudrate of 400kbps. The I²C slave address can be selected via jumpers J0 and J1 and has a default value of 0x50. Additional information on the I²C protocol can be seen in the appendix.

J0	J1	Function
OPEN	OPEN	I²C Slave Address = 0x50
SHORT	OPEN	I²C Slave Address = 0x51
OPEN	SHORT	I²C Slave Address = 0x52
SHORT	SHORT	I²C Slave Address = 0x53

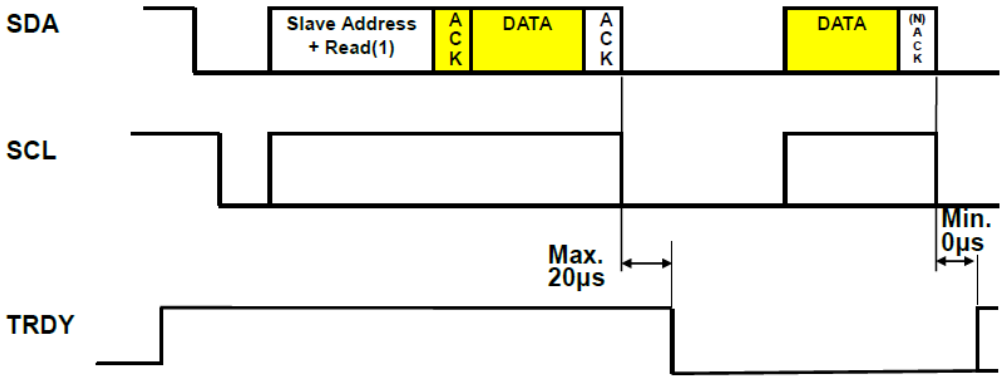
Baudrate	400kbps Maximum
Slave Address	Set by J0 and J1 (Default: 0x50)
Supported Functions	ACK response, clock stretch
Communication Control Signals	SBUSY, TRDY

Data Write Signal Diagram



The SBUSY signal is very important when sending large amounts of data as it signals (with a high voltage state) when the display module has more than 128 bytes in it's receive buffer. Once it's receive buffer has been reduced below 62 bytes, SBUSY will change to its low voltage state.

Data Read Signal Diagram



The TRDY signal indicates when data is available to be read from the module. If TRDY is low, then 0xFF will be transmitted in response to a read sequence. If TRDY is high, data is available to be transmitted and will be transmitted after a read sequence. When TRDY changes from high to low, there is one more byte of data left to be transmitted. This indicates the end of the data stream.

SPI

GT-CP modules have three different modes of operation within SPI. These modes can be seen in the following table:

1st Byte	Operation Mode
0x44	Data Write (Host -> Module)
0x54	Data Read (Host <- Module)
0x58	Status Read (Host <- Module)

Data Write

The data write mode is very straightforward. 0x44 must be sent before any other data is sent to the module.

	1st Byte	2nd Byte	3rd Byte	---	n Byte
MOSI	0x44	Data (1)	Data (2)	---	Data (n – 1)
MISO	-	-	-	---	-

Status Read

In order to read data from the module, a status read must be performed first. 0x58 must be sent to the module first, then the module will respond with a status byte indicating the number of bytes to be read along with a valid indication and the state of SBUSY. A byte of 0x00 might need to be sent after the 1st byte to generate the SPI clock.

	1st Byte	2nd Byte	---	n Byte
MOSI	0x58	-	---	-
MISO	-	Status	Status	Status

Status Data Bit Assignments

b7	b6	b5	b4	b3	b2	b1	b0
SBUSY	/Valid	TL (b5)	TL (b4)	TL (b3)	TL (b2)	TL (b1)	TL (b0)

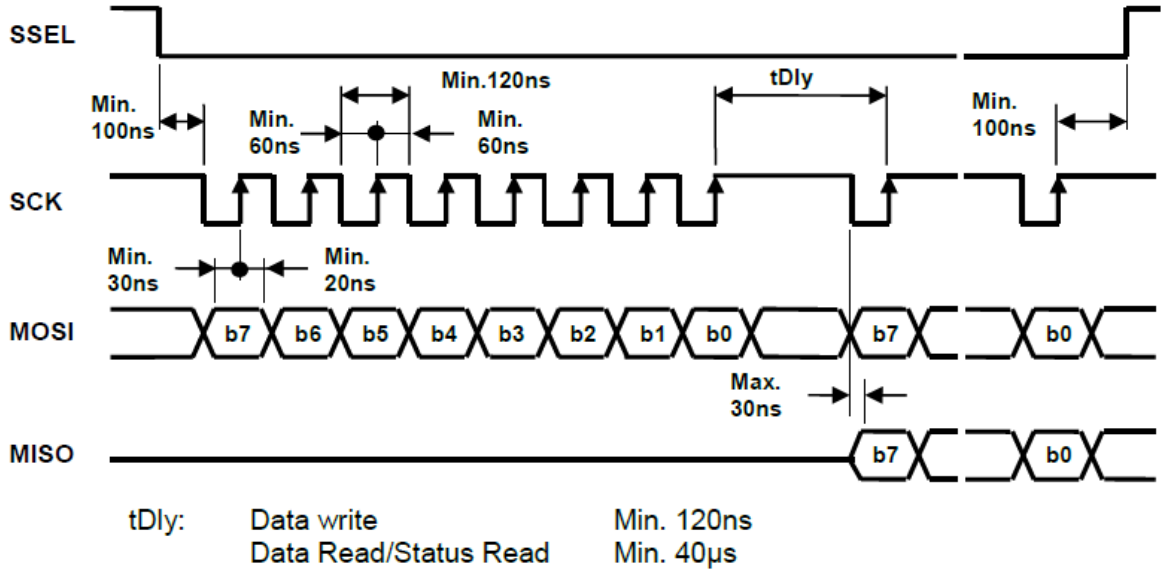
SBUSY = SBUSY signal state
TL = Number of data bytes available in the transmit buffer (63 bytes max.).
/Valid = Status byte valid bit. If b6 = 1, then the Status byte is invalid.

Data Read

Immediately after the status is read from the module, a data read can be performed. The number of bytes reported by the status read will be transmitted and any unread bytes will be discarded.

	1st Byte	2nd Byte	3rd Byte	---	n Byte
MOSI	0x54	-	-	---	-
MISO	-	0x00	Data (1)	---	Data (n – 2)

Signal Diagram



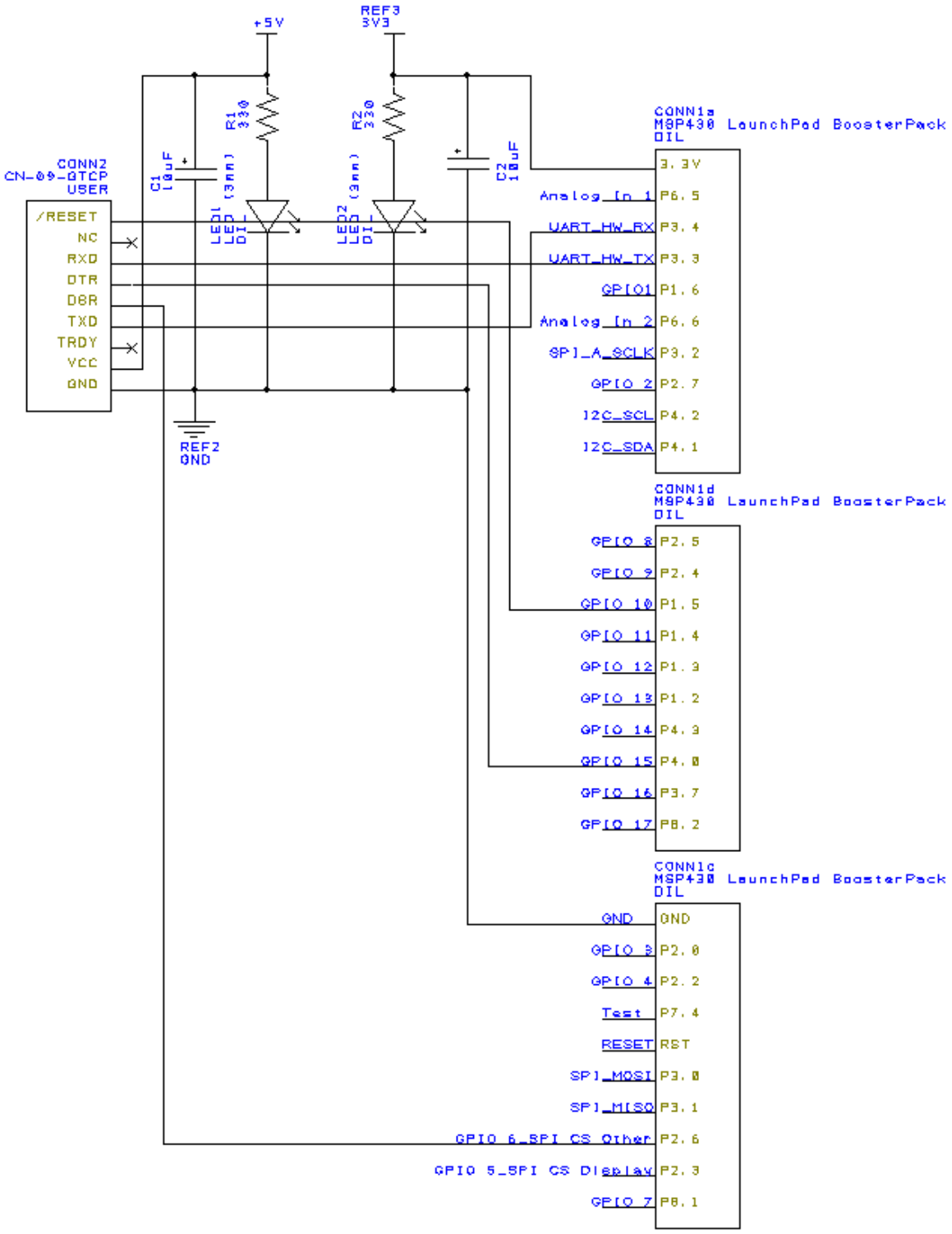
tDly:	Data write	Min. 120ns
	Data Read/Status Read	Min. 40µs

TRDY Signal

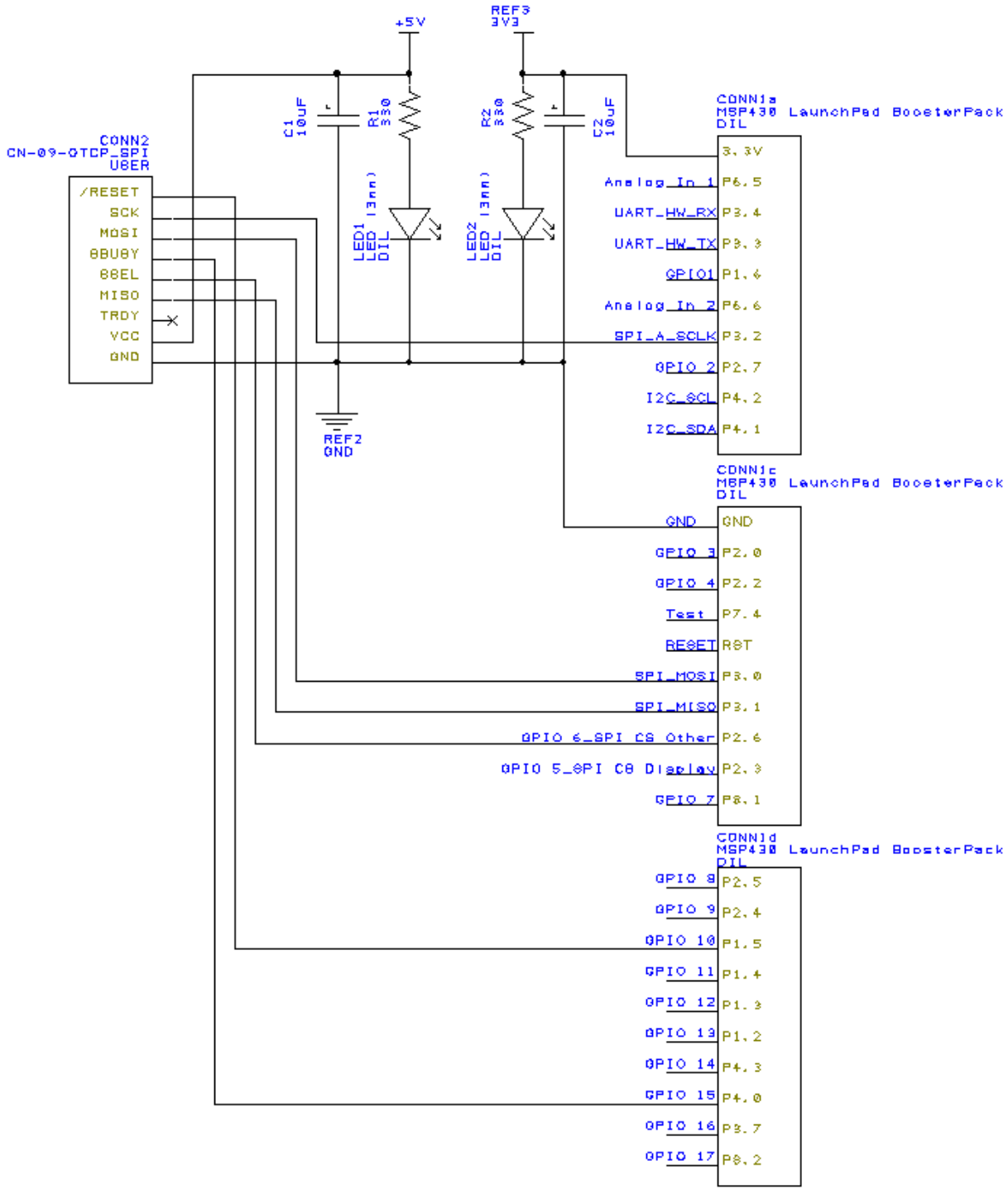
The TRDY signal signifies if data is present in the module's transmit buffer. When this signal goes low, there is no data in the transmit buffer. The signal can change anywhere within the data sequence, so it is difficult to determine when the last byte will be transmitted.

Connection Diagrams

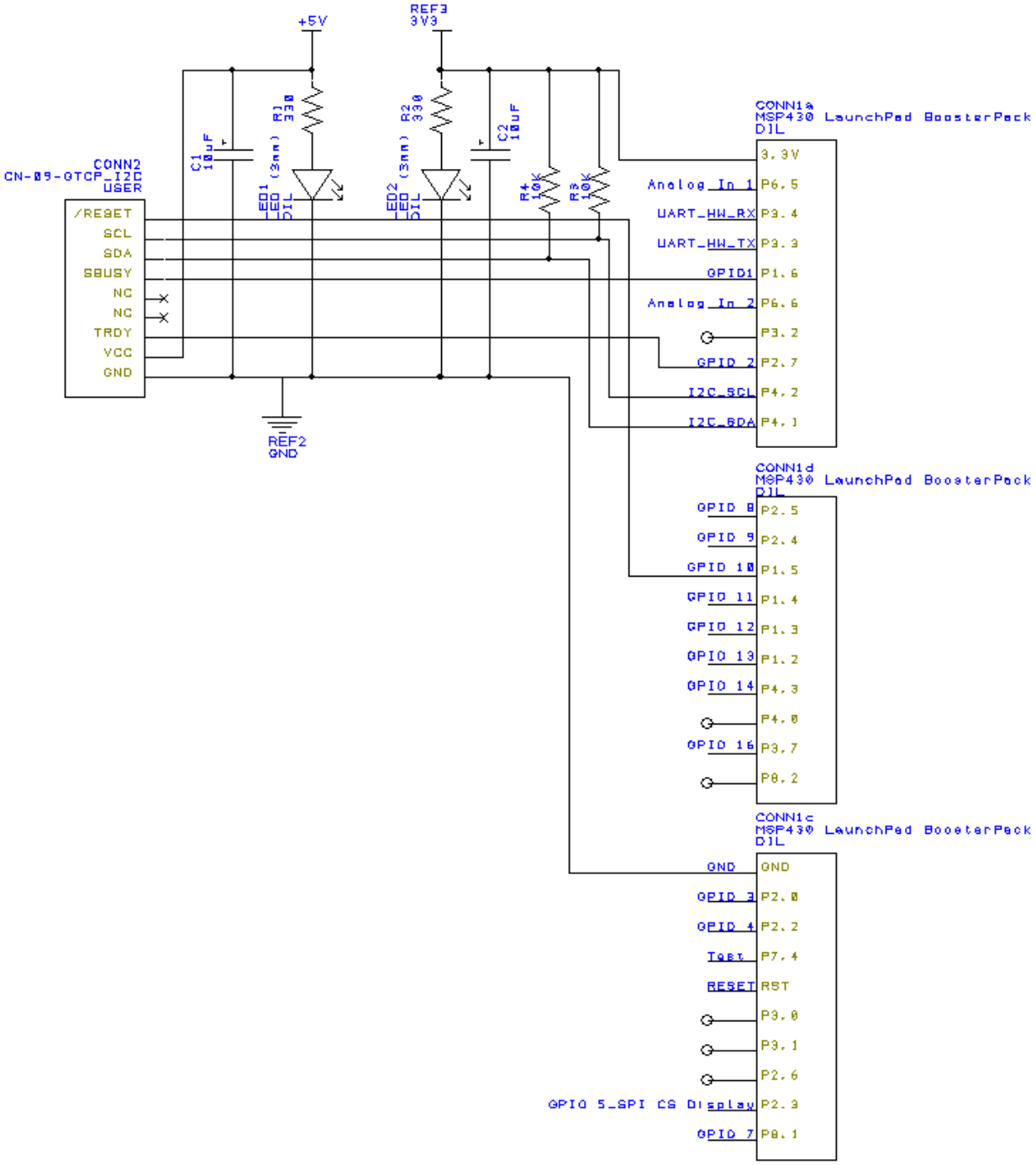
UART



SPI


+5V
REF3
3V3
C1
10uF
R1
330
R2
330
C2
10uF
LED1
LED
DIL
LED2
LED
DIL
CONN2
CN-09-QTCP_SPI
USER
/RESET
SCK
MOSI
SBUSY
SSEL
MISO
TRDY
VCC
GND
REF2
GND
CONN1a
MSP430 LaunchPad BoosterPack
DIL
3.3V
Analog In 1 P6.5
UART_HW_RX P3.4
UART_HW_TX P3.3
GPIO1 P1.6
Analog In 2 P6.6
SPI_A_SCLK P3.2
GPIO 2 P2.7
I2C_SCL P4.2
I2C_SDA P4.1
CONN1c
MSP430 LaunchPad BoosterPack
DIL
GND GND
GPIO 3 P2.0
GPIO 4 P2.2
Test P7.4
RESET RST
SPI_MOSI P3.0
SPI_MISO P3.1
GPIO 6_SPI CS Other P2.6
GPIO 5_SPI CS Display P2.3
GPIO 7 P8.1
CONN1d
MSP430 LaunchPad BoosterPack
DIL
GPIO 8 P2.5
GPIO 9 P2.4
GPIO 10 P1.5
GPIO 11 P1.4
GPIO 12 P1.3
GPIO 13 P1.2
GPIO 14 P4.3
GPIO 15 P4.0
GPIO 16 P3.7
GPIO 17 P8.2

I²C


+5V
REF3
3V3
CONN2
CN-09-GTCP_I2C
USER
/RESET
SCL
SDA
SBUSY
NC
NC
TRDY
VCC
GND
C1
10uF
R1
330
LED1 (3mm)
LED DIL
R2
330
LED2 (3mm)
LED DIL
C2
10uF
R4
10K
R5
10K
REF2
GND
CONN1a
MSP430 LaunchPad BoosterPack
DIL
3.3V
Analog In 1
P6.5
UART_HW_RX
P3.4
UART_HW_TX
P3.3
GPIO1
P1.6
Analog In 2
P6.6
P3.2
GPIO 2
P2.7
I2C_SCL
P4.2
I2C_SDA
P4.1
CONN1d
MSP430 LaunchPad BoosterPack
DIL
GPIO 8
P2.5
GPIO 9
P2.4
GPIO 10
P1.5
GPIO 11
P1.4
GPIO 12
P1.3
GPIO 13
P1.2
GPIO 14
P4.3
P4.0
GPIO 16
P3.7
P8.2
CONN1c
MSP430 LaunchPad BoosterPack
DIL
GND
GND
GPIO 3
P2.0
GPIO 4
P2.2
Test
P7.4
RESET
RST
P3.0
P3.1
P2.6
GPIO 5_SPI CS Display
P2.3
GPIO 7
P8.1

Instructions

This set of instructions will provide guidance on how to wire the host and display up together, display “Hello World” on the display, and recall the image that was saved in the previous “Getting Started with GT-CP” support document. The GTWV070C3A00PA module is used in these instructions, but the GTWV050C3A00PA or GTWQ043C3A00PA modules can be used as well.

1. **Download** the **code library** here.
2. **Download** and **install** Code Composer Studio here.
3. **Connect** MSP430™ to PC via micro USB cable.

4. **Create** and **configure** a **project** in Code Composer Studio**.**
    a. Within your workspace, create a new project by going **to File > New > CCS Project**.
    b. In the “New CCS Project” dialog box, **choose** “MSP430F5529” as the **target**.
    c. **Use** the most recent compiler version and **choose** an empty project (with main.c).
    d. In the workspace window, **expand** the “main.c” node.
    e. Right click on your project node in the project explorer window and choose **Add Files…** .
    f. **Add** the following **include statements** to main.c:
        i. #include <msp430.h>
        ii. #include “GTCP.h”
    g. Right-click on the project node in your project explorer and choose **Properties**.
    h. Go to **MSP430 Compiler**.
    i. Click on “Edit Flags” and add “--c99” to the flags list.
    j. Click “OK”.
    k. Click “Apply and Close”.
5. **Follow** the **“Display Text”** or **“Display an Image”** **instructions**.
6. **Disconnect** the MSP430™ from the PC.
7. **Make sure** the GT-CP’s jumper settings are **correct** for the desired interface.
8. **Wire** the GT-CP to the MSP430™.

a. **Solder** a 9 pin **male pin header** to **CN9** on the GT-CP module so that the pins face away from the display.

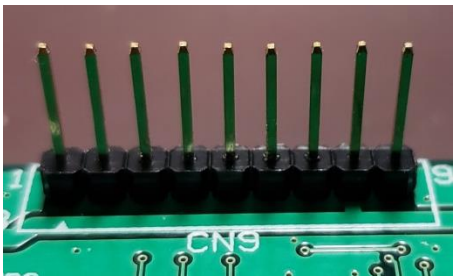


Figure 1: CN9 Pin Header

b. **Set up** the **desired interface** using the diagrams in the Connection Diagrams section.

9. **Apply power** to the circuit. It is recommended to apply power to the GT-CP module first before applying power to the MSP430™.
10. The **selected program** will **run**.
    a. If the **"Display Text"** instructions were followed earlier, then the top left of the display should look like this:

Figure 2: Display Test Result on GT800X480A-C903PA

b. If the **"Display an Image"** instructions were followed earlier, then the top left corner of the display should look like this:

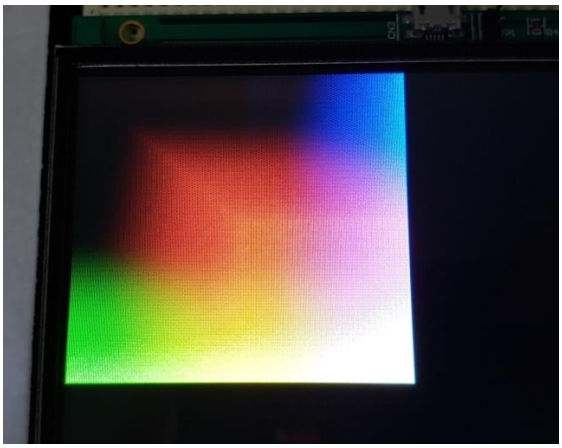


Figure 3: Display Image Result for GT800X480A-C903PA

Display Text

This section will describe how to display “Hello World!” on the display using the easy-to-read built-in outline font. Since the GT-CP’s default font is 5x7 pixels, a larger, easier-to-read font is more practical to display. The source code for this section is available in the appendix.

1. **Add code** in main.
    a. **Initialize** the GT-CP **module.**
        i. GTCP_start(interface);

        This function initializes the GT-CP display, reads and stores necessary information from the display, and initializes the desired serial interface. A delay about 500ms is performed before commands can be sent to the display.

        The **function** will be **written** as **follows**:

        ```
        GTCP_start(I2C); // For I2C

        GTCP_start(SPI); // For SPI

        GTCP_start(UART); // For UART
        ```

    b. **Select** the desired outline **font.**
        i. GTCP_setOutlineFontType(type);

        This function selects the desired outline font type and must be set in order to use outline fonts. In this example, we will use the Japanese font type so the type value will be 0.

        The **function** will be **written** as **follows**:

        ```
        GTCP_setOutlineFontType(0x00);
        ```

        ii. GTCP_setFontSize(size);

        This function selects the desired font size for the display. In order to use an outline font, the font size setting must be set to “outline font” with a value of 0.

        The **function** will be **written** as **follows**:

        ```
        GTCP_setFontSize(0x00);
        ```

        iii. GTCP_setOutlineFontSize(lineHeight, nominalY, nominalX, offset);

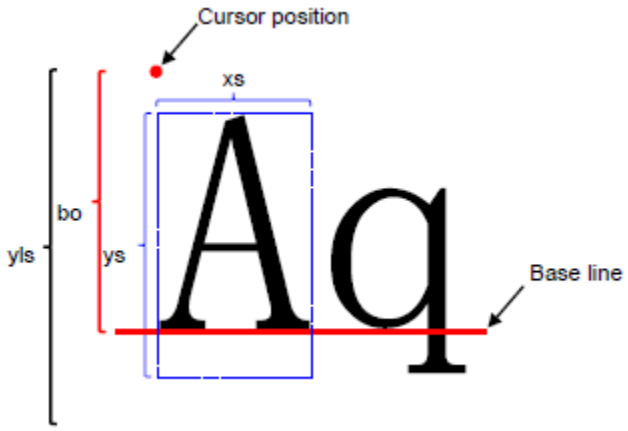
        This function adjusts outline font size parameters:

        lineHeight (yls) dictates the height of the entire line of text.

nominallY (ys) is normally less than lineHeight and is the nominal character height.

nominalX (xs) is the nominal character width.

offset (bo) is the number of pixels between the cursor and the character base line.



For this example, we will use a lineHeight of 40, nominalX and nominalY values of 32 and an offset of 0. Please note that the nominalX and nominalY are relative values and a value of 32 will not create square characters. The firmware will automatically adjust the text to be properly proportioned based on the font design.

The **function** will be **written** as **follows**:

```
GTCP_setOutlineFontSize(40, 32, 32, 0);
```

c. **Send** “Hello World!” to module. Since an outline font is selected, this text will appear as an outline font. Any text afterwards will appear as an outline font as well until the font settings are changed.
   i. GTCP_printString(“Hello World!”);

2. **Load program** onto host board.
3. **Go** to step 6 of the base instructions.

Display an Image

This section will describe how to recall and display the image that was saved in the previous "Getting Started with GT-CP" support document. Please follow the instructions on this document to save the image to the module's FROM2 before continuing with the instructions below. The source code for this section is available in the appendix.

1. **Add code** in main.
    a. **Initialize** the GT-CP **module.**
        i. GTCP_init(width, height);

        This function initializes the GT-CP display, reads and stores necessary information from the display, and initlalizes the desired serial interface. A delay about 500ms is performed before commands can be sent to the display.

        The function will be written as follows:

        ```
        GTCP_start(I2C); // For I2C

        GTCP_start(SPI); // For SPI

        GTCP_start(UART); // For UART
        ```

    b. **Recall** the **saved image**.
        i. GTCP_storedBitImageDraw(memory, address, extension, xDefine, xSize, ySize, format);

        This function sends the necessary byte data to use the downloaded image recall command. This command is described in more detail in the complex command section. For this example, the previously stored image as the following properties:

        Memory Type: FROM2 starting at 0x00000000

        Address: 0x00000000

        Size: 256 x 256

        Format: 16-bit high speed (0x91)

        So, the function will be written as follows:

        ```
        GTCP_storedBitImageDraw(0x10, 0x0000, 0x00, 256, 256,
        256, IMAGE_FORMAT_16BIT_HS);
        ```

2. **Load program** onto host board.
3. **Go** to step 6 of the base instructions.

Troubleshooting

Module does not turn on or display backlight is blinking when power is applied.

- Make sure the power supply can generate at least 5V 1.5A 7.5W.
- Unplug all major components (GT-CP, host board, capacitors) and plug them back in. This will help discharge any capacitors.
- Make sure that one common ground is established.
- Make sure that JP1 is open.

The program is not displaying the expected information.

- Make sure that the jumper configuration is correct for the desired interface.
- Check the continuity of each wire used in the circuit.
- *[UART]* Make sure that the baudrate on the display module and host board match.
- *[I^2C]* Make sure that the address being sent to the module matches the module's address configuration. (The default address is 0x50)
- *[SPI]* Make sure the leading byte being used for each read/write mode is correct.

Recalled image is not displaying correctly.

- Make sure that you followed the directions correctly from the Getting Started with GT-CP: Connect to PC support document.

If you are experiencing any issues that are not answered, please contact support at support.ele@noritake.com or post your issue on our forum.

Appendix

Text Display Source Code

UART Text Display

```
#include <msp430.h>
#include "GTCP.h"

int main(void)
{
	WDTCTL = WDTPW | WDTHOLD;	// stop watchdog timer

	GTCP_start(UART);

	GTCP_setOutlineFontType(0x00);
	GTCP_setFontSize(0x00);
	GTCP_setOutlineFontSize(40, 32, 32, 0);
	GTCP_printString("Hello World!");

	return 0;
}
```

SPI Text Display

```
#include <msp430.h>
#include "GTCP.h"

int main(void)
{
	WDTCTL = WDTPW | WDTHOLD;	// stop watchdog timer

	GTCP_start(SPI);

	GTCP_setOutlineFontType(0x00);
	GTCP_setFontSize(0x00);
	GTCP_setOutlineFontSize(40, 32, 32, 0);
	GTCP_printString("Hello World!");

	return 0;
}
```

I²C Text Display

```
#include <msp430.h>
#include "GTCP.h"

int main(void)
{
	WDTCTL = WDTPW | WDTHOLD;	// stop watchdog timer

	GTCP_start(I2C);

	GTCP_setOutlineFontType(0x00);
	GTCP_setFontSize(0x00);
	GTCP_setOutlineFontSize(40, 32, 32, 0);
	GTCP_printString("Hello World!");

	return 0;
}
```

Image Display Source Code

UART Image Display

```
#include <msp430.h>
#include "GTCP.h"

int main(void)
{
	WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

	GTCP_start(UART);

	GTCP_storedBitImageDraw(0x10, 0, 256, 256, 256, IMAGE_FORMAT_16BIT_HS);

	return 0;
}
```

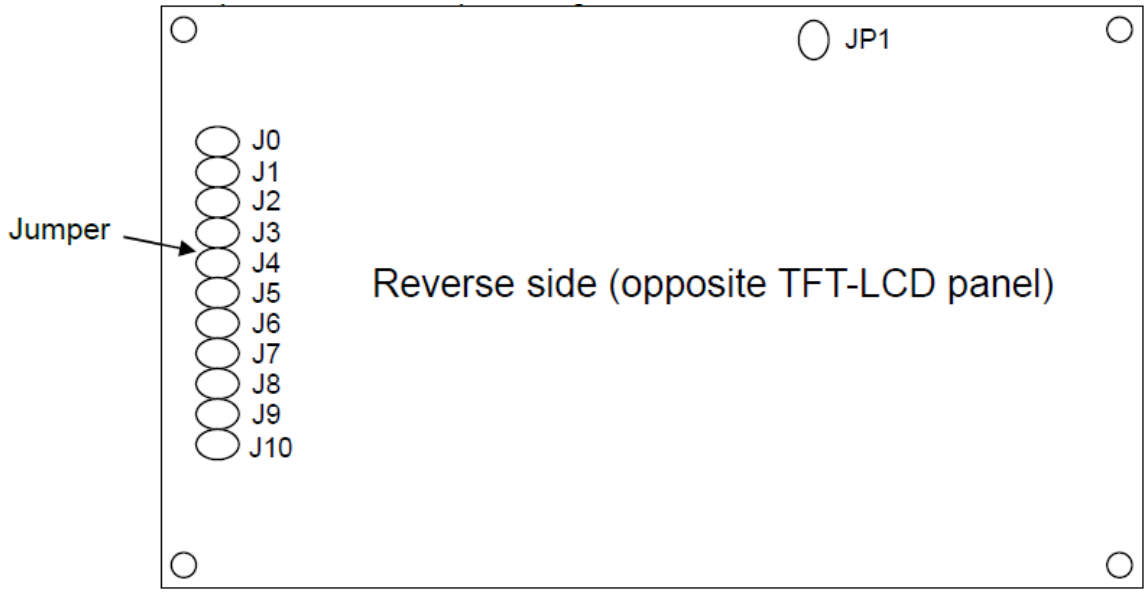
SPI Image Display

```
#include <msp430.h>
#include "GTCP.h"

int main(void)
{
	WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

	GTCP_start(SPI);

	GTCP_storedBitImageDraw(0x10, 0, 256, 256, 256, IMAGE_FORMAT_16BIT_HS);

	return 0;
}
```

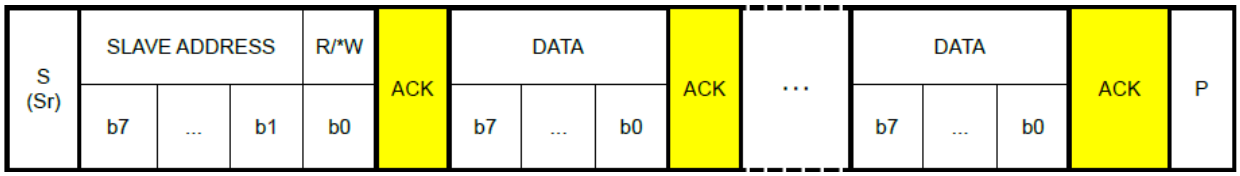
I²C Image Display

```
#include <msp430.h>
#include "GTCP.h"

int main(void)
{
	WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

	GTCP_start(I2C);

	GTCP_storedBitImageDraw(0x10, 0, 256, 256, 256, IMAGE_FORMAT_16BIT_HS);

	return 0;
}
```

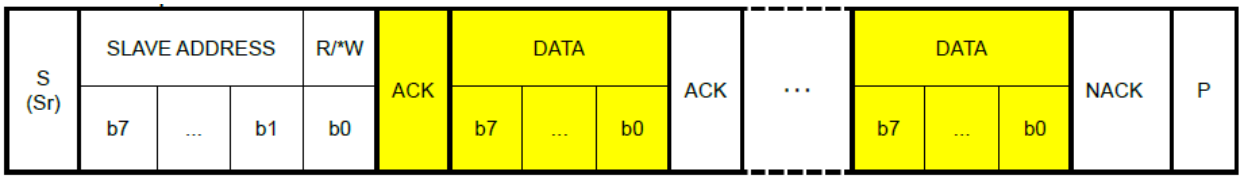
Jumper Location



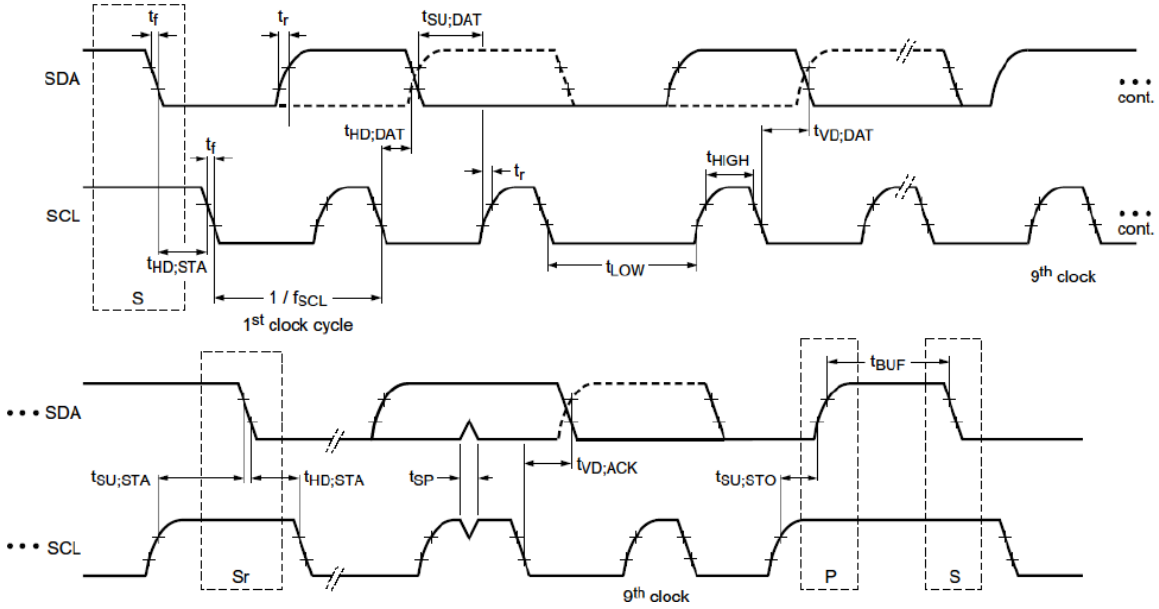
I²C Data Write Sequence



I²C Data Read Sequence



I²C Timing Diagrams



I²C Timing Table

Parameter	Symbol	Condition	Min.	Typ.	Max.	Unit
SCL clock frequency	f_{SCL}	-	0	-	400	kHz
Start condition hold time	$t_{HD;STA}$	-	0.6	-	-	µs
SCL 'L' time	t_{LOW}	-	1.3	-	-	µs
SCL 'H' time	t_{HIGH}	-	0.6	-	-	µs
Start condition setup time	$t_{SU;STA}$	-	0.6	-	-	µs
Data hold time	$t_{HD;DAT}$	-	0	-	-	µs
Data setup time	$t_{SU;DAT}$	-	100	-	-	ns
SCL, SDA rise time	t_r	-	20	-	300	ns
SCL, SDA fall time	t_f	-		-	300	ns
Stop condition setup time	$t_{SU;STO}$	-	0.6	-	-	µs
Stop condition – start condition bus idle time	t_{BUF}	-	20	-	-	µs

Disclaimer

THIS DOCUMENT AND ITS CONTENTS ARE PROVIDED BY NORITAKE CO., INC. "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS DOCUMENT (AND ITS CONTENTS), EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Revision History

Document Number	Date	Revision Note
E-M-0199-00	12/20/2018	Initial Release
E-M-0199-01	06/17/2020	Replaced old GT series PNs with new PNs Added reset information Added more UART communication information
E-M-0199-02	04/23/2021	Implemented all code library update changes.