TI Designs

RGB LED Signal Tower With Wi-Fi® Interface Reference Design

Description

The objective of the TIDA-00994 reference design is to develop an embedded firmware of a multi-segment RGB signal tower used in factory floor and industrial process automation of greater complexity. A Wi-Fi® interface is implemented to control the stack light and read back status information.

The CC3200 offers a Wi-Fi access point as well as an embedded web server for reading status information and controlling the stack light.

Resources

TIDA-00994	Design Folder
TIDA-00979	Design Folder
CC3200	Product Folder
CC3200 LaunchPad™	Tool Folder
TLC5971	Product Folder
LMZ35003	Product Folder
MSP430F5528	Product Folder
HDC1080	Product Folder
TPD2E001	Product Folder

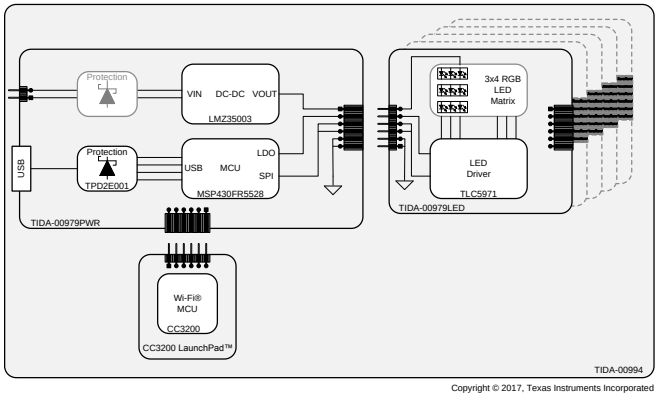
TI E2E™ Community — ASK Our E2E Experts

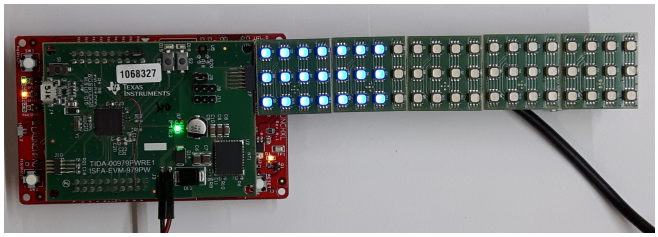
Features

- Flexible and Easy-to-Control RGB LED Tower Light Design
- Different Switchable Modes: Demo, Temperature, Humidity, and Stack Light
- Controllable Through Wi-Fi Interface
- One to Five RGB LED Segments With Four Individual Channels Each
- Adjustable Maximum LED Current up to 60 mA per Channel

Applications

- Factory Automation and Control
- Building Automation





An IMPORTANT NOTICE at the end of this TI reference design addresses authorized use, intellectual property matters and other important disclaimers and information.

1 System Description

Industrial signal lights indicate the status of manufacturing equipment or the status of processes in industrial environments. These lights are often called stack lights, tower lights, or indicator lights, and contain one to five difference colors. The IEC 60073 standard shows how the colors correspond to different states (see Table 1).

Table 1. IEC 60073 International Color Usage

COLOR	SAFETY MEANING	CONDITION OF A PROCESS
RED	Danger	Emergency or fault
AMBER	Warning	Abnormal
GREEN	Safe	Normal
BLUE	Mandatory significance	
WHITE	No specific meaning assigned	

This TI Design provides a flexible solution based on RGB light-emitting diodes (LEDs) and a modular approach. The solution has several printed-circuit boards (PCBs), as Figure 1 shows:

- Power and control board (TIDA-00979PW)
- LED driver with RGB LEDs (TIDA-00979LD)
- CC3200 LaunchPad™ Development Kit

Five of the mentioned LED boards are daisy chained and connected to one control board. This control board runs a stack light control software and communicates through a serial peripheral interface (SPI) with the CC3200 LaunchPad.

The LED driver requires an external power supply of 18 V to 36 V. The microcontroller (MCU) is externally powered through the CC3200 LaunchPad. The stack light can be controlled through Wi-Fi using an embedded web server or a simple command line interface.

This design can be configured to be:

- A simple one-color stack light
- A multicolor stack light
- A tower light with single or multicolor

1.1 Key System Specifications

Table 2 shows the key system specifications.

Table 2. Key System Specifications

PARAMETER	SPECIFICATION
Power supply	V_{IN_MIN} = 18 V
	V_{IN_TYP} = 24 V
	V_{IN_MAX} = 36 V
Interfaces	Wi-Fi®
LED	RGB
	Max 60 mA for each channel
	Brightness control
	Configured as 3×20 array

1.2 Block Diagram

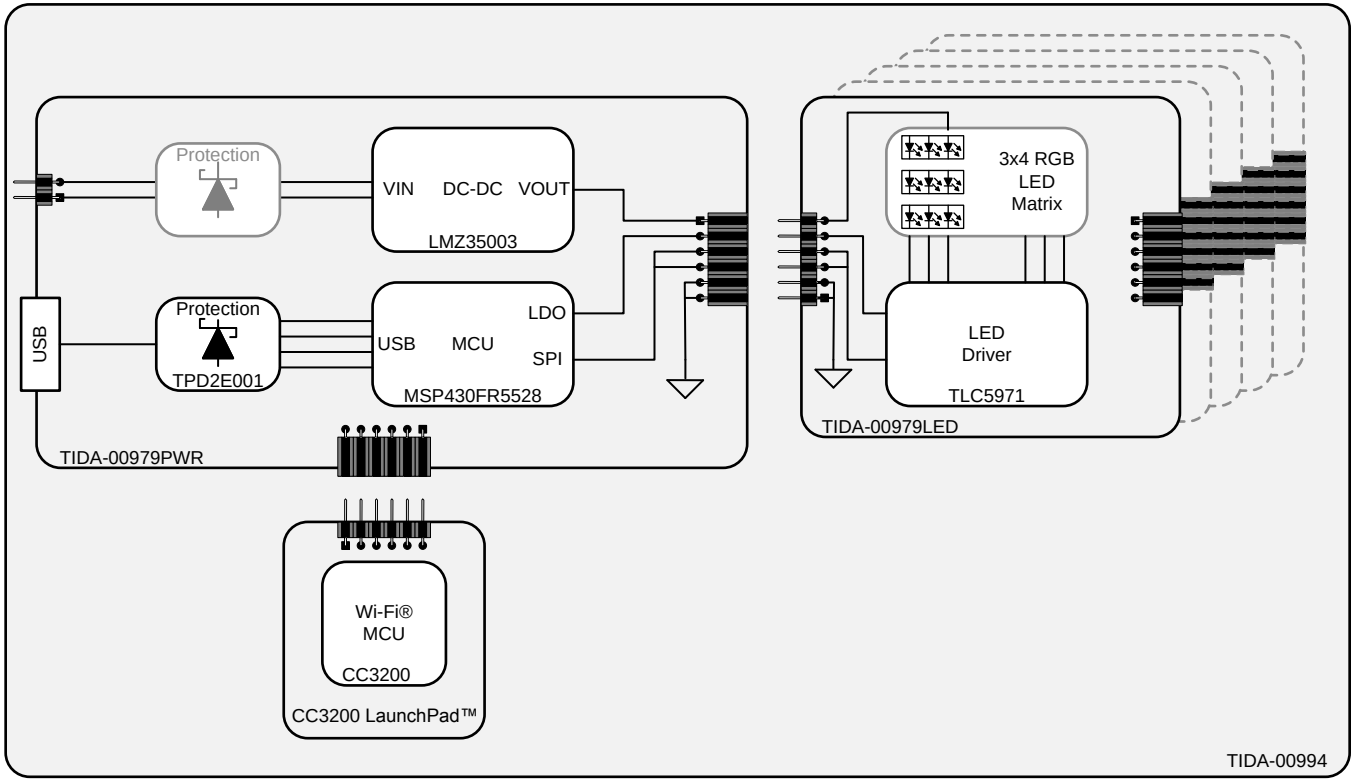


Figure 1. TIDA-00994 Block Diagram

1.3 Highlighted Products

The TIDA-00994 reference design features the TLC5971, LMZ35003, MSP430F5528, HDC1080, TIPD2E001, and CC3200.

For more information on each of these devices, see their respective product folders at www.ti.com.

1.3.1 TLC5971

The TLC5971 device is a 12-channel, constant-current sink driver. Each output channel has individually adjustable currents with 65536 pulse-width modulation (PWM) grayscale (GS) steps. Each color group can be controlled by 128 constant-current sink steps with the global brightness control (BC) function. GS control and BC are accessible through a two-wire signal interface. The maximum current value for each channel is set by a single external resistor. All constant-current outputs are turned off when the integrated circuit (IC) is in an overtemperature condition. Figure 2 shows the functional block diagram of the TLC5971 device.

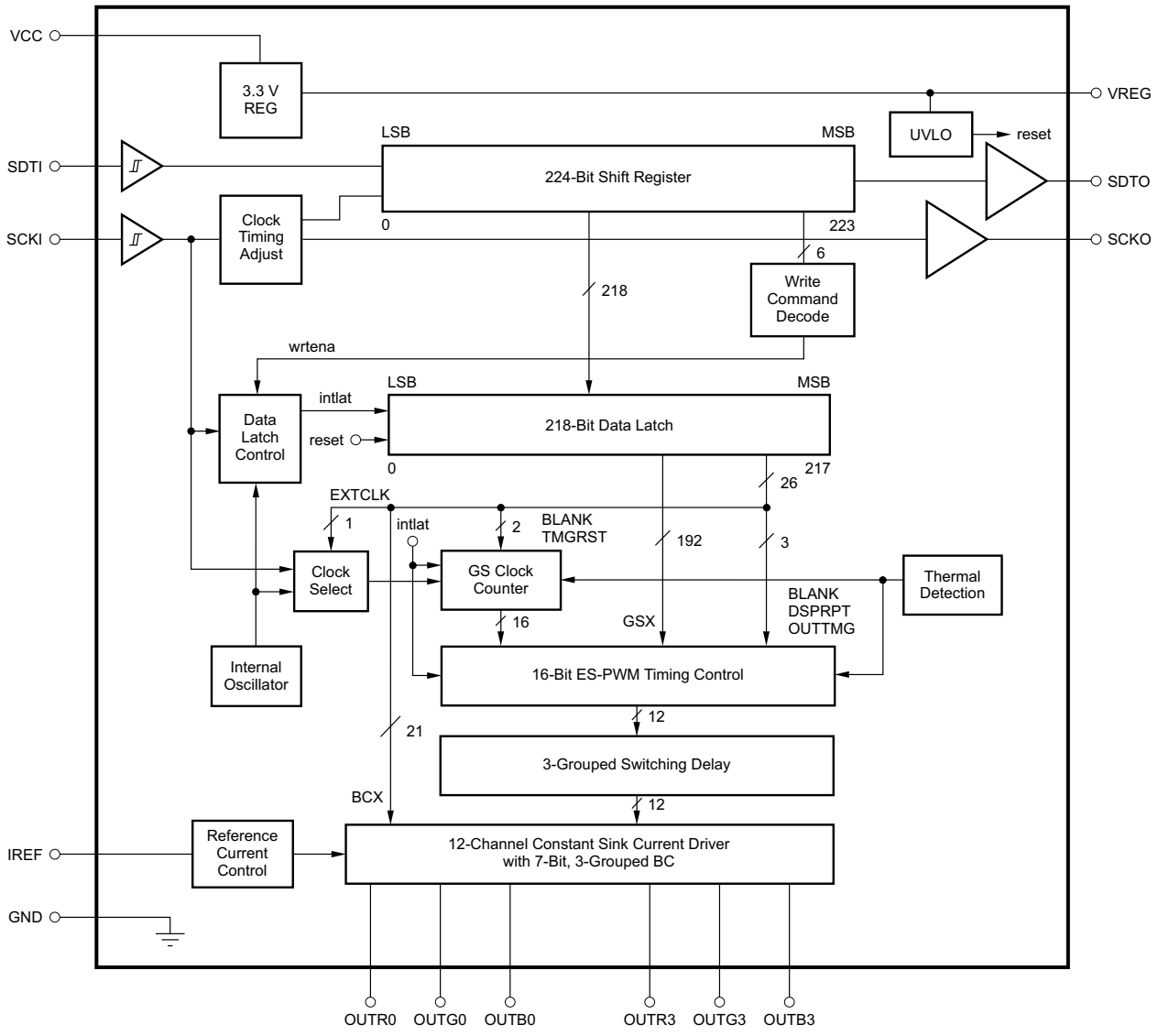


Figure 2. TLC5971 Functional Block Diagram

1.3.2 LMZ35003 SIMPLE SWITCHER®

The LMZ35003 SIMPLE SWITCHER® power module is an easy-to-use integrated power solution that combines a 2.5-A DC-DC converter with a shielded inductor and passives into a low-profile QFN package. This total power solution allows as few as five external components and eliminates the loop compensation and magnetics part-selection process.

The small 9-mm × 11-mm × 2.8-mm QFN package is easy to solder onto a PCB and allows a compact point-of-load design with greater than 90% efficiency and excellent power dissipation capability. The LMZ35003 offers the flexibility and the feature-set of a discrete point-of-load design and is ideal for powering a wide range of ICs and systems. Advanced packaging technology affords a robust and reliable power solution that is compatible with standard QFN mounting and testing techniques. Figure 3 shows the functional block diagram of the LMZ35003 device.

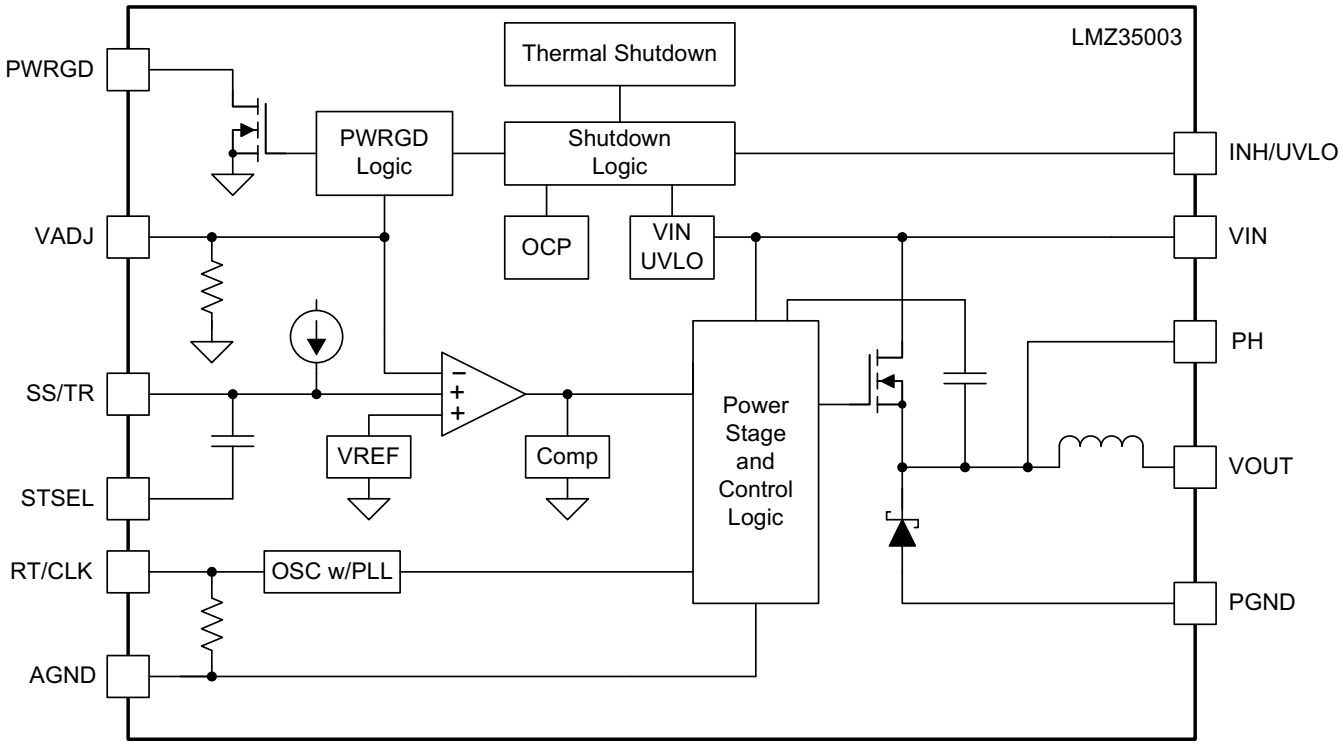


Figure 3. LMZ35003 Functional Block Diagram

1.3.3 MSP430F5528

The TI MSP430™ family of ultra-low-power MCUs consists of several devices featuring peripheral sets targeted for a variety of applications. The architecture, combined with low-power modes, is optimized to achieve extended battery life in portable measurement applications. The MCU features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency. The digitally controlled oscillator (DCO) allows the devices to wake up from low-power modes to activate in 3.5 µs (typical).

The MSP430F5529, MSP430F5517, MSP430F5525, and MSP420F5521 MCUs have integrated USB and PHY, supporting USB 2.0, four 16-bit timers, a high-performance 12-bit analog-to-digital converter (ADC), two universal serial communication interfaces (USCI), a hardware multiplier, DMA, a real-time clock (RTC) module with alarm capabilities, and 63 I/O pins. The MSP430F5528, MSP420F5526, MSP430F5524, and MSP430F5522 MCUs have all of these peripherals but have 47 I/O pins.

The MSP420F5519, MSP430F5517, and MSP430F5515 MCUs have integrated USB and PHY supporting USB 2.0, four 16-bit timers, two USCIs, a hardware multiplier, DMA, an RTM module with alarm capabilities, and 63 I/O pins. The MSP430F5514 and MSP430F5513 MCUs include all of these peripherals but have 47 I/O pins.

Typical applications include analog and digital sensor systems, data loggers, and others that require connectivity to various USB hosts. Figure 4 shows the functional block diagram of the MSP430F5528 device.

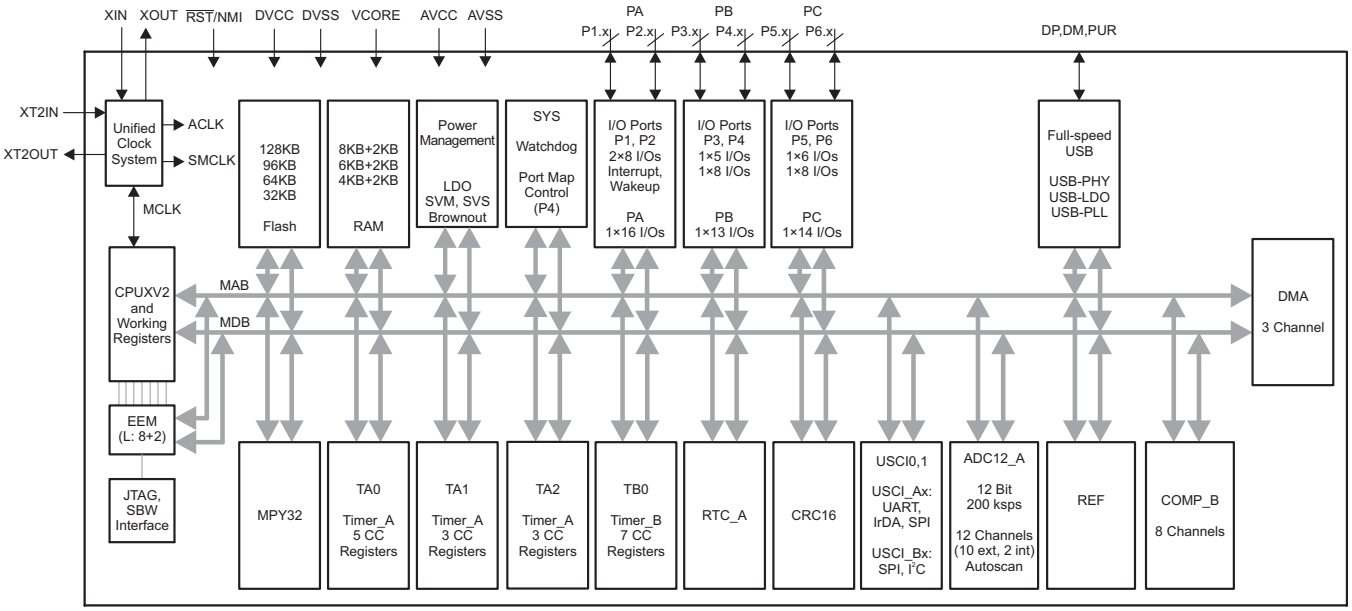


Figure 4. MSP430F5528 Functional Block Diagram

1.3.4 HDC1080

The HDC1080 is a digital humidity sensor with an integrated temperature sensor that provides excellent measurement accuracy at low power. The HDC1080 operates over a wide supply range and is a low-cost, low-power alternative to competitive solutions in a wide range of common applications. The humidity and temperature sensors are factory calibrated.

Figure 5 shows the functional block diagram of the HDC1080 device.

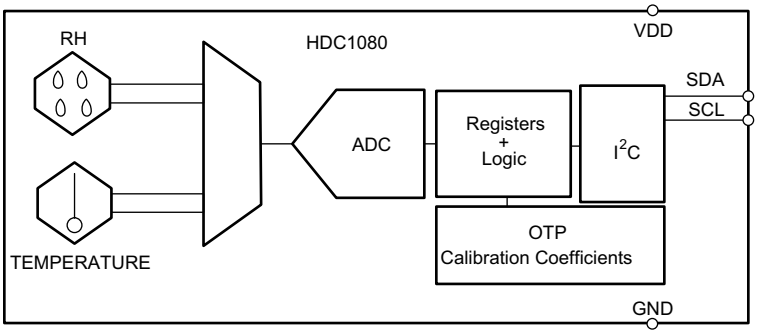


Figure 5. HDC1080 Functional Block Diagram

1.3.5 TPD2E001

The TPD2E001 is a two-channel, transient voltage suppressor (TVS) based, electrostatic discharge (ESD) protection diode array (see Figure 6). The TPD2E001 is rated to dissipate ESD strikes at the maximum level that is specified in the IEC 6100-42 Level 4 international standard.

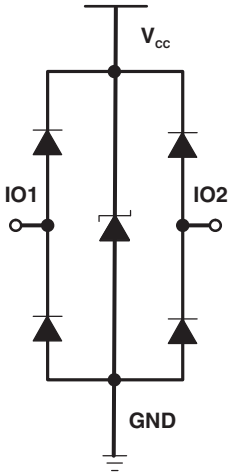


Figure 6. TPD2E001 Diagram

1.3.6 CC3200

The CC3200 is a powerful single-chip MCU with built-in Wi-Fi connectivity. The device has an integrated high-performance ARM® Cortex®-M4 MCU, allowing customers to develop an entire application with a single IC. With on-chip Wi-Fi, Internet, and robust security protocols, no prior Wi-Fi experience is required for faster development.

The device has an integrated 80-MHz Cortex M4 CPU, 256 kB of RAM, and external SPI flash. The device includes a wide variety of peripherals, including a fast parallel camera interface, I^2S, SD/MMC, universal asynchronous receiver/transmitter (UART), SPI, I^2C, and four-channel ADC. The CC3200 family includes flexible embedded RAM for code and data and ROM with an external serial flash bootloader and peripheral drivers.

The Wi-Fi network processor subsystem features a Wi-Fi Internet-on-a-Chip™ and contains an additional dedicated ARM MCU that completely offloads the applications MCU. This subsystem includes an 802.11 b/g/n radio, baseband, and MAC with a powerful crypto engine for fast, secure Internet connections with 256-bit encryption. The CC3200 device supports station, access point, and Wi-Fi direct modes. The device also supports WPA2 personal and enterprise security and WPS 2.0. The Wi-Fi Internet-on-a-chip includes embedded TCP/IP and TLS/SSL stacks, HTTP server, and multiple Internet protocols.

The power-management subsystem includes integrated DC-DC converters supporting a wide range of supply voltages. This subsystem enables low-power consumption modes, such as the hibernate with RTC mode, which requires less than 4 μA of current.

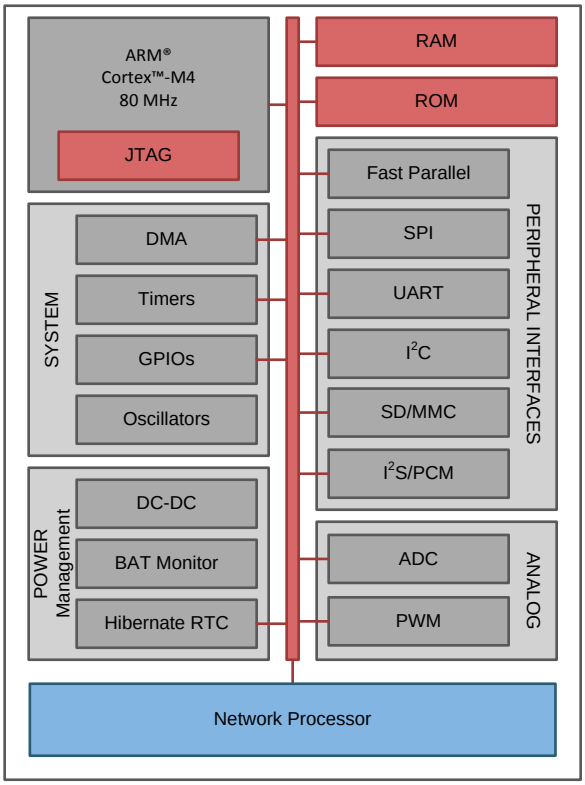


Figure 7. CC3200 Hardware Overview

2 System Design Theory

This TI Design consists of two different boards: one is the TIDA-00979 for controlling the LEDs and the other one is the CC3200 LaunchPad as a Wi-Fi interface. Use of the pins is chosen in such a way that makes stacking the PCBs possible.

2.1 System Overview

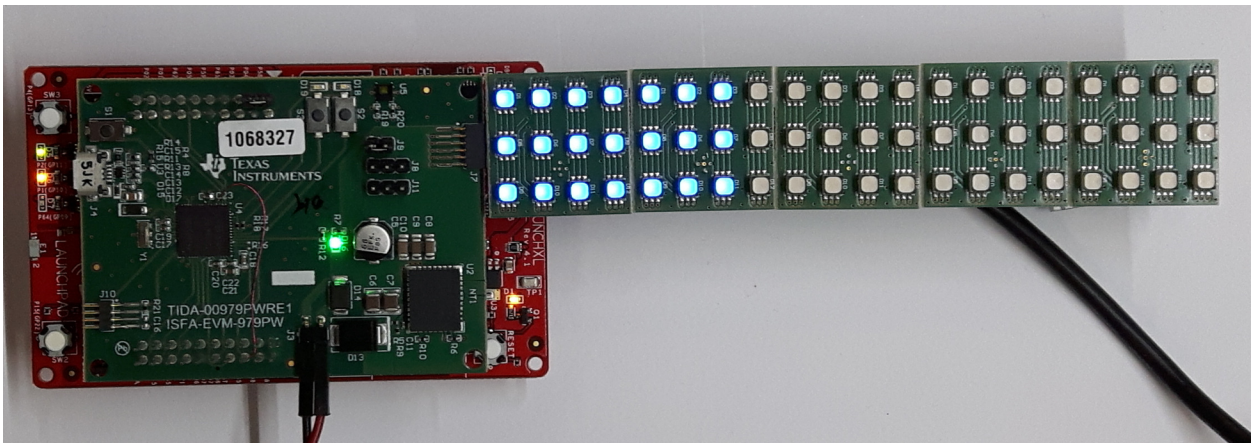


Figure 8. Setup Image

The two stacked PCBs that Figure 8 shows communicate using SPI. The CC3200 device is configured as master in mode one and the MSP430F5528 on the TIDA-00979 is configured as slave. A bidirectional communication is implemented between the two boards. Table 3 shows the connections that are necessary for the communication.

Table 3. Connection Between TIDA-00979 and CC3200 LaunchPad™

SIGNAL	CONNECTOR - PIN TIDA-00979	PIN CC3200
nReset	J4 - 20	15
SPI nCS	J4 - 6	8
SPI MOSI	J4 - 12	7
SPI MISO	J4 - 14	6
SPI CLK	J3 -13	5
GND	J4 - 2	GND
3.3 V	J3 - 1	VCC

2.2 Software

The software of the MSP430F5528 is adapted from TIDA-00980 with some changes in the SPI. For details about the SPI see Section 2.2.1.

The SimpleLink™ platform web server runs on the CC3200 device. The web server is extended to offer different demo sites for controlling the LED pattern by sending the appropriate SPI commands; moreover, a simple server is listening on port 1234 TCP for sending custom SPI commands. This feature can simplify automated control of different stack lights.

2.2.1 TIDA-00979 Stack Light Controller

As previously mentioned, a SPI is used for controlling the TIDA-00979 stack light controller. The device acts as a SPI slave in mode one. Each transaction consist of 32 bits: 1 bit to distinguish between read and write, 7 bits for the register address, and the remaining 24 bits are data. Table 4 explains the communication.

Table 4. SPI Communication

BIT	FIELD	DESCRIPTION
31	R/W	0: SPI write 1: SPI read
30 - 24	Register	Select the register to read/write
23 - 0	Data	Contains data written to or read from register

Different registers are implemented to control the operation of the stack light, see Table 5 for a list and explanation.

Table 5. Register Description

REGISTER	NAME	TYPE	RESET	DESCRIPTION
0x01	Mode	RW	0	0: Demo mode 1: Temperature mode 2: Humidity mode 3: Stack light mode
0x02	Pattern	RW	0	0: Off 1: All on 2: Blinking 3: Stack light 4: Temperature visualization 5: Run light 6: level Light
0x03	Color	RW	0xFFFFFF	Bit 23 - 16: Red Bit 15 - 8: Green Bit 7 - 0: Blue
0x04	Brightness	RW	0x01	0 - 127: Brightness
0x05	Interval	RW	0x14	Interval for blinking
0x06	Data	RW	0x14	Data for different Pattern (meaning dependant on selected pattern, see Table 6)
0x07	Temp min	RW	0x0F	Minimum value for temperature visualization
0x08	Temp max	RW	0x1E	Maximum value for temperature visualization
0x09	Level min	RW	0	Minimum value for level visualization
0x0A	Level max	RW	0x64	Maximum value for level visualization
0x0B	Temperature	R	—	Current ambient temperature in ºC
0x0C	Humidity	R	—	Current ambient humidity in %

The value of the data register has a different meaning depending on the selected pattern. Table 6 provides an overview of the different functions.

Table 6. Data Register Usage Depending on Pattern

PATTERN	DESCRIPTION OF DATA REGISTER
1: All on	Number of LEDs to turn on
2: Blinking	Number of LEDs to blink
3: Stack Light	Segment to turn on
4: Temperature visualization	Temperature value to scale and display
5: Run light	Number of LEDs the run light should cover
6: Level light	Level to display (gets scaled in the configured range)

2.2.1.1 SPI Slave Operation

An SPI slave USCIA0 of the MSP is used, which is configured in three-wire mode and to generate an interrupt after every eight bits of data.

```
GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P3, GPIO_PIN4); // UCA0 SOMI P3.4
GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P3, GPIO_PIN3); // SIMO P3.3
GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P2, GPIO_PIN7); // CLK P2.7

USCI_A_SPI_initSlave(USCI_A0_BASE, USCI_A_SPI_MSB_FIRST,
USCI_A_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT, USCI_A_SPI_CLOCKPOLARITY_INACTIVITY_LOW);
USCI_A_SPI_enable(USCI_A0_BASE);
USCI_A_SPI_enableInterrupt(USCI_A0_BASE, UCRXIE);
```

The received data is handled in the interrupt request (IRQ) function of the SPI module. In case of a write instruction, the data can be parsed after receiving four bytes. For any read instructions, it is necessary to react after receiving the first byte and load the corresponding value to the UCA0TXBUF register for transmission with the next eight clock cycles.

To avoid disturbance when only one byte is shifted or any other event occurs on the SPI interface, an additional pin P1.1 is used as a SPI chip select to reset the state machine when a deselect is registered (low-to-high transition on chip select). This action is done by configuring an IRQ on the corresponding IO pin on the rising edge.

```
P1IES &= ~(1<<1); // low to high IRQ
P1IFG &= ~(1<<1); // reset any pending IRQs
P1IE |= (1<<1); // enable IRQ
```

The position counter of the receiving data can then be reset in the ISR.

```
#pragma vector=PORT1_VECTOR
__interrupt void PORT1_ISR(void)
{
    P1IFG &= ~(1<<1);
    spi_pos = 0; // reset the position counter
}
```

In this implementation, different SPI devices can not be connected in parallel with different chip selects because the SOMI pin does not get configured as high impedance and remains as output. This issue can either be resolved by using the SPI four-wire mode, or through software by switching this pin between input and output, depending on the state of the chip select signal.

To solve this in software, both edges of the chip select must be detected. The interrupt is initially configured to detect the falling edge and then changed depending on the detected edge.

```
P1IES |= (1<<1); // high to low IRQ
P1IFG &= ~(1<<1); // reset any pending IRQs
P1IE |= (1<<1); // enable IRQ
```

The ISR now changes the configuration of the edge detecting IRQ and reconfigures the IO pin depending on the edge.

```
#pragma vector=PORT1_VECTOR
__interrupt void PORT1_ISR(void)
{
    if(P1IES & 1<<1){ // falling edge
        P3SEL |= (1<<4); // configure pin for spi module
        P3DIR |= (1<<4);
        P1IES &= ~(1<<1); // low to high IRQ
    }else{ // rising edge
        spi_pos = 0; // reset the position counter
        P3SEL &= ~(1<<4); // configure pin as input
        P3DIR &= ~(1<<4);
        P1IES |= (1<<1); // high to low IRQ
    }
    P1IFG &= ~(1<<1);
    P6OUT |= (1<<4);
    P6OUT &= ~(1<<4);
}
```

Reacting as fast as possible on the edge is necessary to have the pin configured before any data is sent by the master; therefore, any function calls are avoided. A clock frequency of 100 kHz is used here and the chip select is asserted 5 µs before the first clock edge because the data is sampled on the second (falling) edge; therefore, the pin must be reconfigured within 10 µs.

Figure 9 shows the transmission. Channel D4 signals the exit of the IRQ for the purpose of analysis. To see the switching between output and high impedance, the SOMI signal is connected with high resistance to a function generator. The user can observe from the figure that it takes about 5 µs to switch from input to SOMI mode and it is fast enough to be able to receive the data at the falling edge.

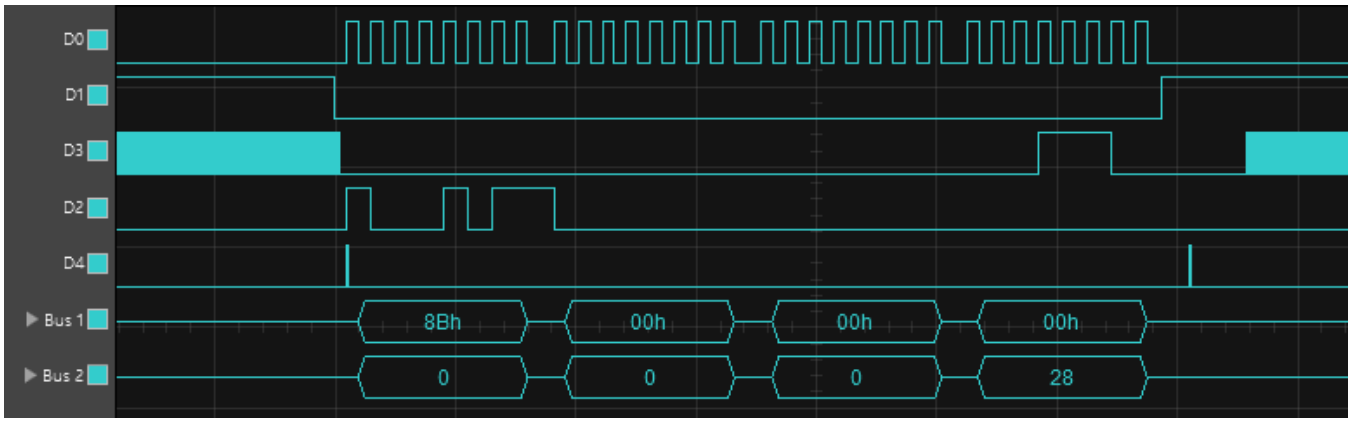


Figure 9. SPI Communication

2.2.1.2 SPI Usage

Not all registers are used in every mode. Only brightness, temperature, and humidity as well as the mode register have an influence in every state. The remaining registers are used depending on the mode register.

When the mode is set to temperature, the temp min and temp max registers also have an influence. These registers make it possible to configure the displayed temperature range.

When using humidity mode instead of the temp min and temp max registers, the level min and level max registers influence the scaling.

In the forth mode, stack light mode, full control is possible as previously mentioned. All registers are used and full customization is possible as explained in the preceding Table 5.

After power on, the software starts in demo mode. In this example, the following sequence must be set to switch to temperature visualization and to set the temperature through SPI:

- Set to stack light mode (0x01 00 00 03)
- Select temperature visualization (0x02 00 00 04)
- Select desired brightness if default is too dim (0x04 00 00 7F)
- Configure range according to desired specifications (0x07 00 00 00 and 0x08 00 00 64)
- Write the value to be displayed into data register (0x06 00 00 14)

2.2.2 CC3200 LaunchPad™

A CC3200 device is used as SPI master. The device is programmed to act as an access point and provides a web server and a simple socket server. The web server offers a website which enables easy configuration and changing between the different modes. The socket server provides a sort of command line, which can be used for register access through SPI.

2.2.3 Web Server

A website is generated and stored in the flash of the CC3200 device to enable a simple user interface which is independent from the client that is used.

The integrated SimpleLink web server is used for hosting a static website. The files that can be accessed through the server are transferred to flash by using the UniFlash standalone tool. To implement dynamic content, such as values that should be set by software or those that can be set by the user on a webpage, the web server offers different tokens that cause a callback to the software.

2.2.3.1 Retrieve Values From Software

For retrieving data generated by software, it is necessary to replace the particular part in the HTML file by a token such as __SL_G_BRI, whereas the last three characters can be user defined. Whenever the server reads the HTML file to deliver it and such a token is found, the SimpleLinkHttpServerCallback is called.

Now the user can handle what is displayed in the software. The following code snippet shows an example for dynamic data.

```
unsigned char BRIGHTNESS_GET_token[] = "__SL_G_BRI";

void SimpleLinkHttpServerCallback(SlHttpServerEvent_t *pSlHttpServerEvent, SlHttpServerResponse_t
*pSlHttpServerResponse) {
    char spi_cmd[4];
    memset(spi_cmd, 0, sizeof(spi_cmd));
    switch (pSlHttpServerEvent->Event) {
        case SL_NETAPP_HTTPGETTOKENVALUE_EVENT:
            unsigned char *ptr;

            ptr = pSlHttpServerResponse->ResponseData.token_value.data;
            pSlHttpServerResponse->ResponseData.token_value.len = 0;
            if (memcmp(pSlHttpServerEvent-
>EventData.httpTokenName.data, BRIGHTNESS_GET_token, strlen((const char *) BRIGHTNESS_GET_token))
== 0) {
                spi_cmd[0] = 0x84;
                pSlHttpServerResponse-
>ResponseData.token_value.len += sprintf((char*) ptr, "%d", send_spi(spi_cmd, 4)&0xff);
            }
        break;
    }
}
```

Another interesting feature is the ability to include an entry that is automatically refreshed, for example, displaying the current temperature measured inside the device. Therefore, jQuery can be used. In this case, a file called *temp* is placed on the server, which contains the corresponding token for replacement as well as the unit: __SL_G_TEM °C. In the main HTML file, it then becomes necessary to have a entry with an id:

```
<tr><td class="label">
        Temperature:
    </td>
    <td id="temp" class="value">
        __SL_G_TEM &deg;C
    </td>
</tr>
```

Having an entry with an id enables the possibility to automatically reload the content of the table entry:

```
<script src="jquery.min.js" type="text/javascript"></script>
<script type="text/javascript">
$(document).ready(function () {
    var auto_refresh = setInterval(
        function(){
            $('#temp').load('temp');
        }, 500);
});
</script>
```

With this code, the content of the table entry is automatically refreshed every 500 ms. The same procedure can be applied to any value that requires a regular refresh.

2.2.3.2 Setting Values on Server

Three major parts are necessary In order to be able to set values on the server side: an input item (such as a text box), a javascript part that sends the data as POST request, and the server side that handles this post request.

As an input item (such as a text box, for example), a slider or a color select can be used. In this example, a text box is used:

```
<input id="brightness_text" type="number" min="1" max="127" value="__SL_G_BRI"
oninput="SelectBrightness(brightness_text.value)">
```

This HTML code creates a text box that is limited to enter numbers in the range of 1 to 127. After loading the page, the text box is then filled with the value that the server replaces from the token __SL_G_BRI. Entering a value causes a call to the function SelectBrightness with the value of this input item as the parameter.

The following code snippet shows an example implementation in JavaScript for sending a POST request.

```
function SelectBrightness(which){
    var params = "__SL_P_BRI="+which;
    HTTPrequest.open("POST","No_content", true);
    HTTPrequest.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    HTTPrequest.setRequestHeader("Content-length", params.length);
    HTTPrequest.setRequestHeader("Connection", "close");
    HTTPrequest.onreadystatechange = function()
    {
        if (HTTPrequest.readyState==4 && HTTPrequest.status==200)
        {
        }
    }
    HTTPrequest.send(params);
}
```

This piece of code sends a token ("__SL_P_BRI") and attaches the value that must be sent as a string. Then a new HTTP request is generated and when a connection is established, the data is sent as a POST request.

The SimpleLink server calls the SimpleLinkHttpServerCallback function whenever a special token such as __SL_P_BRI is received and passes the data. The following code shows how to handle this issue:

```
void SimpleLinkHttpServerCallback(SlHttpServerEvent_t *pSlHttpServerEvent, SlHttpServerResponse_t
*pSlHttpServerResponse) {
    char spi_cmd[4];
    memset(spi_cmd, 0, sizeof(spi_cmd));
    switch (pSlHttpServerEvent->Event) {
        case SL_NETAPP_HTTPPOSTTOKENVALUE_EVENT:
            unsigned char *ptr = pSlHttpServerEvent->EventData.httpPostData.token_name.data;

            if (memcmp(ptr, BRIGHTNESS_token, strlen((const char *) BRIGHTNESS_token)) == 0) {
            ptr = pSlHttpServerEvent->EventData.httpPostData.token_value.data;
            *(pSlHttpServerEvent->EventData.httpPostData.token_value.len + ptr) = 0;
            spi_cmd[0] = 0x04;
            spi_cmd[1] = 0;
            spi_cmd[2] = 0;
            spi_cmd[3] = strtol((const char*) ptr, NULL, 10);
            send_spi(spi_cmd, 4);
        }
        break;
    }
}
```

Because this callback is also used for other events, such as GET requests, the first step is to decide wether it is a GET or a POST request and handle the request accordingly. Therefore, checking the content of pSlHttpServerEvent->Event is necessary and, in case of a POST request, the parameters must be checked further.

The name of the token is handed over as a pointer in the variable *pSlHttpServerEvent->EventData.httpPostData.token_name.data*. Depending of the string at this address, the code must decide how the data of the token must be handled. The data is stored as a pointer in *pSlHttpServerEvent->EventData.httpPostData.token_value.data* and the corresponding length in *pSlHttpServerEvent->EventData.httpPostData.token_value.len*. The data string is not a zero determined string; therefore, the user must handle this accordingly and with care. In this example, a zero is added after the end of the string, which can only be done when non-binary data is expected otherwise zero could be part of the data.

The data can then be parsed using the typical c functions. In this case, a string containing a number from 1 to 127 is expected, so it can be parsed to an integer using strtol: *spi_cmd[3] = strtol((const char*) ptr, NULL, 10);*. If a one-digit number is expected, parsing can be simplified by just subtracting the ASCII value of zero: *spi_cmd[3] = *ptr - '0';*.

Further processing is done in this example by sending the data through SPI to the previously mentioned MSP430F5528.

2.2.4 Socket Server

In addition to the integrated web server, a simple socket server is implemented to handle a command line interface. FreeRTOS runs as the operating system on the CC3200 and the SimpleLink software provides standard Berkeley sockets, so implementing a server is rather simple.

First, a task is created that opens a network port in the listening state:

```
osi_TaskCreate(SockServerTask, (signed char*) "SockServerTask", OSI_STACK_SIZE, NULL,
OOB_TASK_PRIORITY, NULL);
```

Calling *osi_TaskCreate* creates a task, which is implemented in the function *SockServerTask*.

In *SockServerTask* a socket is generated with *int sock=sl_Socket(SL_AF_INET,SL_SOCK_STREAM, 0)*, then the address (Port 1234, TCP) must be configured and assigned to this socket:

```
SlSockAddrIn_t addr;
addr.sin_family = SL_AF_INET;
addr.sin_port = sl_Htons((unsigned short)1234);
addr.sin_addr.s_addr = 0;
sl_Bind(sock, (SlSockAddr_t *) &addr, sizeof(addr))
```

The socket then must be set to the listening state so clients can connect to it: *sl_Listen(sock, 0)*. Now the software must check whether a client has connected to the port and accept the connection, after which data can be sent. The user does not have to implement the whole TCP/IP protocol, this is already done in the SimpleLink software.

Accepting connections must be done in a loop to enable more than one connection. A new task has to be spawned and the socket must be handed over for each connection. The following code example shows this action in a simplified way:

```
int cli_sock;
SlSockAddrIn_t cli_addr;
while(1){
        cli_sock = sl_Accept(sock, (struct SlSockAddr_t *)&cli_addr, 0);
        osi_TaskCreate(SockServerTaskHandler, (signed char*) "SockServerTaskHandler",
OSI_STACK_SIZE, (void*)cli_sock, OOB_TASK_PRIORITY, NULL);
    }
```

In the function *SockServerTaskHandler* all the communication with a client is handled.

```
static void SockServerTaskHandler(void *pvParameters) {
int cli_sock = (int) pvParameters;
int status;
char buf[32];
while(1){
    status = sl_Recv(cli_sock, buf, sizeof(buf), 0);
    if (status > 0){
        // fill buf with data
        sl_Send(cli_sock, buf, strlen(buf), 0);
        }
    }
}
```

By calling the function, *sl_Recv* data is received from the client and copied in buf. The return value of this function is the number of received bytes (in case of a positive value) or an error code. The data in buf can now be handled and overwritten before it is sent back to the client using the *sl_Send* function.

Using these functions, a simple interface is implemented for accessing the SPI registers explained in Table 5. By sending "r" following the register in hexadecimal, a register can be read. A "w" followed by the register and the value must be sent for writing the registers. Section 3.3 shows examples.

3 Getting Started Hardware

Use the following instructions to begin using the hardware.

1. Verify the jumper settings (J9 must be shorted, J5 pin 1 and pin 3 also must be shorted)
2. Connect five TIDA-00979LD boards together (J2 to J11)
3. Connect J2 of the TIDA-00979LD board to J7 of the TIDA-00979PW board
4. Stack the TIDA-00979PW board to the CC3200 LaunchPad
5. Provide 24 V to J3 of TIDA-00979PW
6. Connect the CC3200 LaunchPad to USB

The TIDA-00979 runs in demo mode after completing the previous five steps.

3.1 Programming Firmware

Connect the programming tool to J10 to program the MSP430F5528 on the TIDA-00979PW. The software can now be flashed from the Code Composer Studio™ (CCS) software.

Programming the CC3200 LaunchPad can be done by shorting J15 and pushing the reset button. The board can now be flashed using CCS UniFlash.

3.2 Using Web Interface

After flashing the application and powering up both boards, the design starts in demo mode and displays different patterns. The CC3200 is configured to act as the access point with an SSID starting with "mysimplelink". After connecting a smart phone or a computer to this access point, the website can be accessed by typing the IP address of the CC3200 in the address bar of any web browser. Figure 10 shows a screenshot of the webpage after start-up.

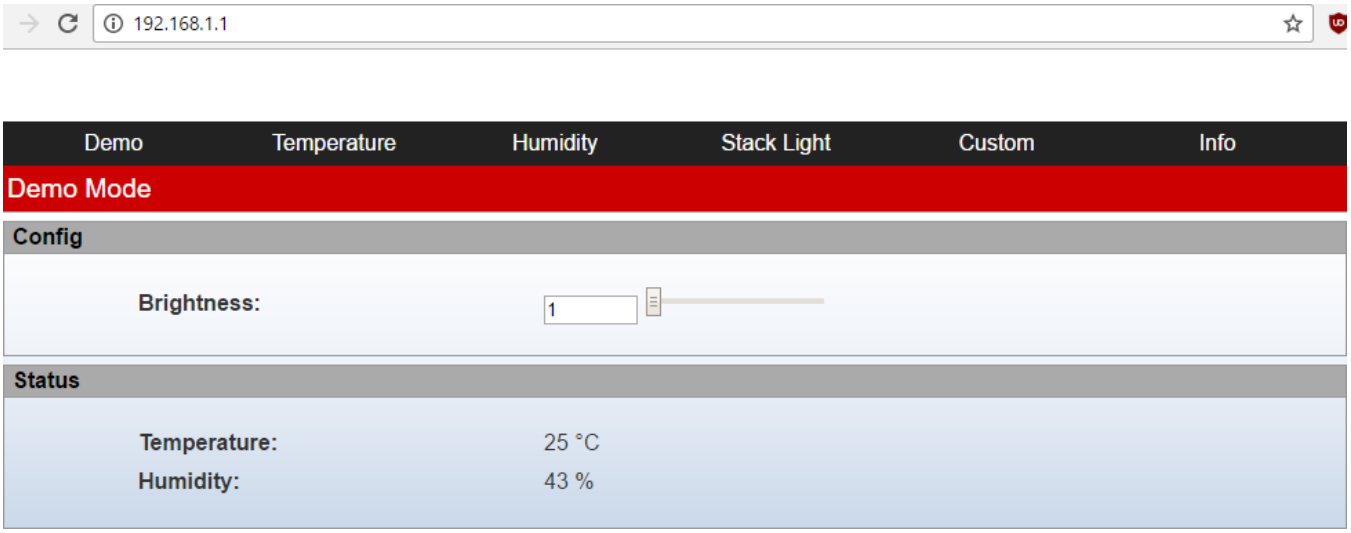


Figure 10. Demo Mode

In this mode, only the brightness of the light-emitting diodes (LEDs) can be changed using either the slider or the text box in the *Config* section. In the *Status* section, the current temperature and humidity measured on the TIDA-00979PW board is displayed and refreshed every 500 ms.

After clicking *Temperature* in the menu bar, the software switches to temperature mode and the current ambient temperature is displayed on the LED bar. The website then offers the possibility to change the range of the display, as the screenshot in Figure 11 shows.

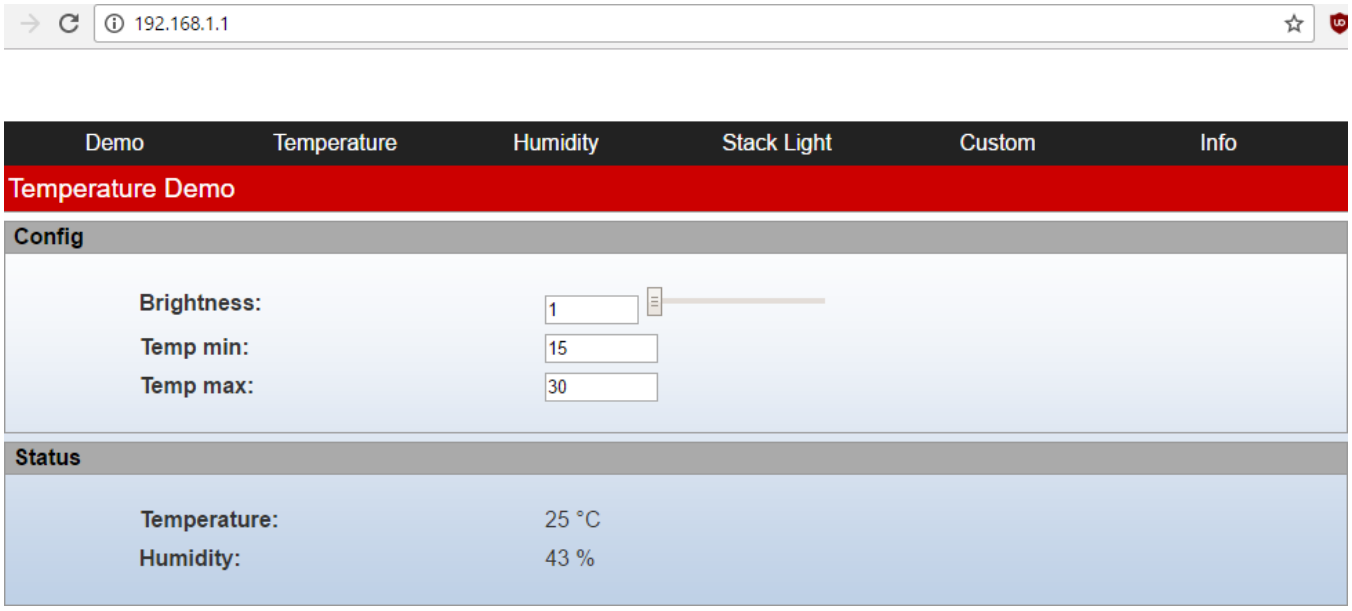


Figure 11. Temperature Mode

The pattern automatically changes when the different tabs are selected.

The *Custom* tab offers a more dynamic range of features where every value that can be configured on the TIDA-00979 can be changed by the user. The screenshot in Figure 12 shows the various settings.

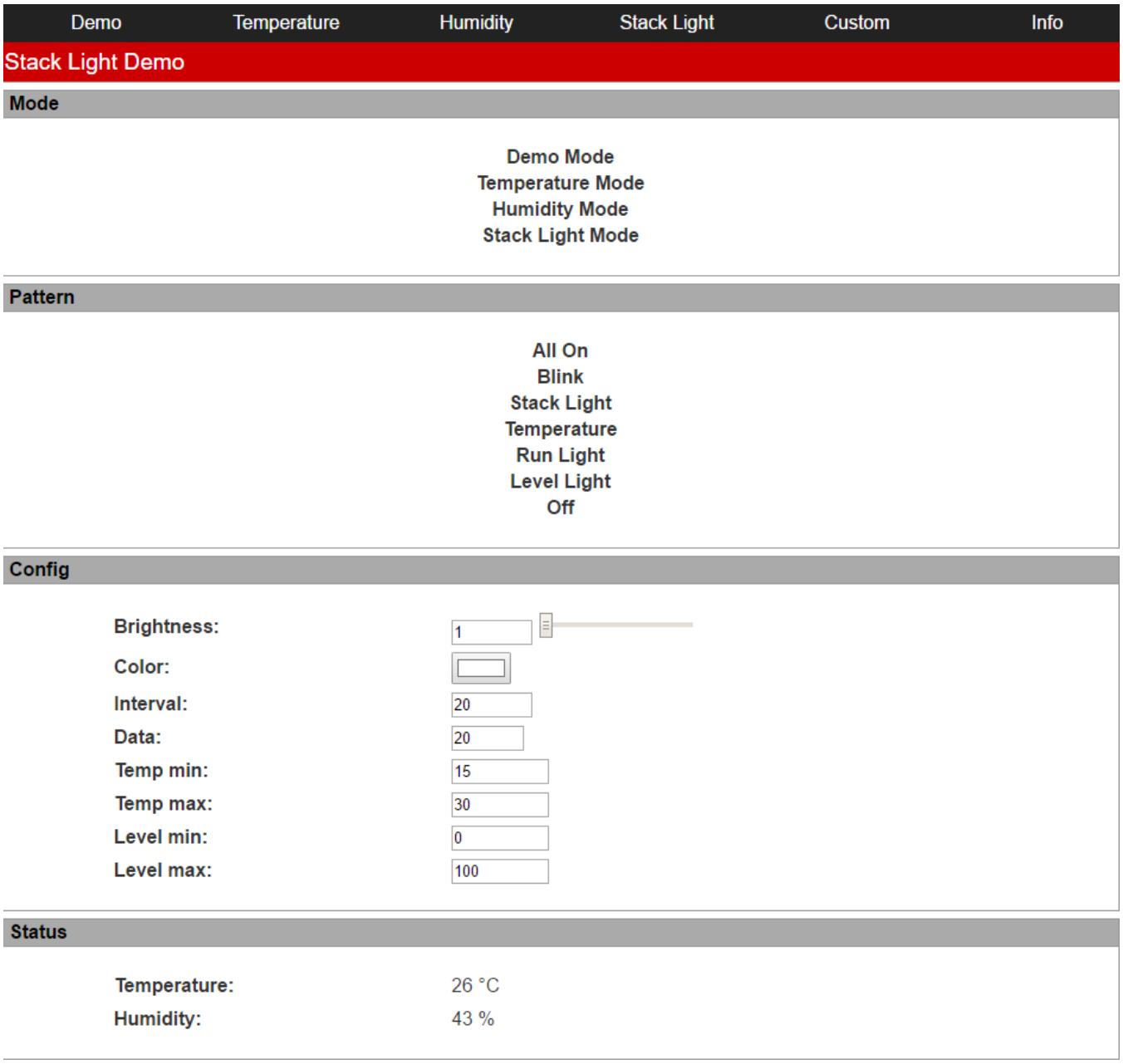


Figure 12. Custom Configuration

When accessing the *Custom* tab in this mode, the values do not automatically set. All modes and patterns are available for selection in this view. Depending on the selection, the different values have different meanings, as Table 5 explains.

3.3 Command Line Interface

In addition to the web interface, a simple command line interface is also implemented on port 1234 TCP. One example for accessing this interface is by using PuTTY in RAW mode, connecting to 192.168.1.1 on port 1234. Each SPI register can be set manually in this mode. The screenshot in Figure 13 shows a few examples, which are sent using the simple interface on port 1234.

The first row issues a write command to register 0x01 with the value 0x02, which leads to a level pattern showing the current ambient humidity.

In the second row a read of address 0x0B, which contains the temperature, is requested. The value is printed in the next row. The value of 0x1B equals 27°C.

The next line reads the humidity value. In this example a 0x27 equaling 39% is read.

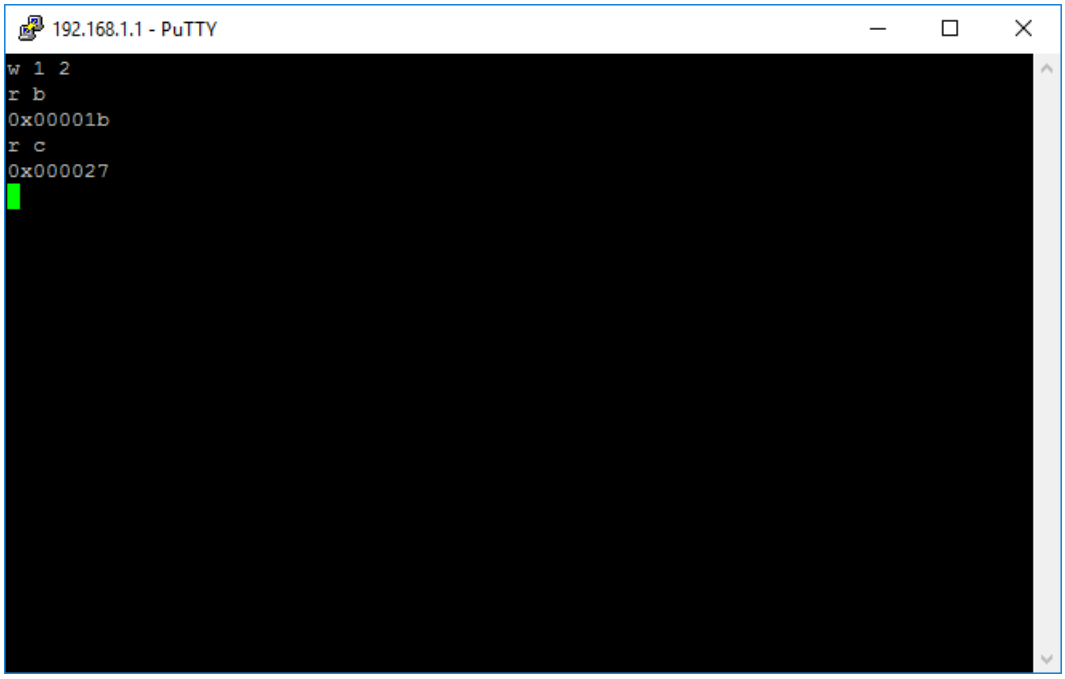


Figure 13. Command Line Interface

A plausibility check is not implemented and every command that is entered is sent through SPI to the TIDA-00979 board.

The preceding commands yield to operation in level mode and a display similar to Figure 14.

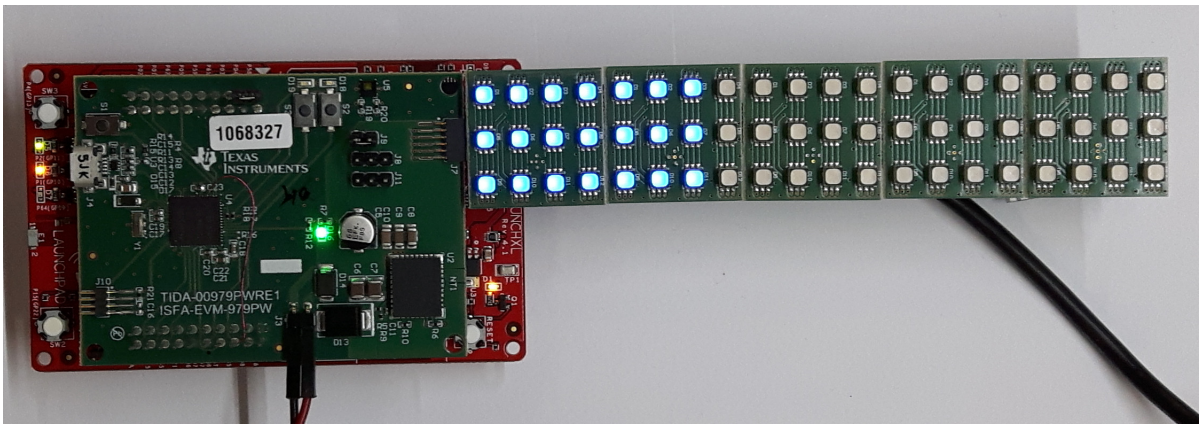


Figure 14. Level Light Displaying Humidity

4 Design Files

4.1 Schematics

To download the schematics, see the design files at TIDA-00994.

4.2 Bill of Materials

To download the bill of materials (BOM), see the design files at TIDA-00994.

4.3 Altium Project

To download the Altium project files, see the design files at TIDA-00994.

4.4 Layout Guidelines

The layout of the TIDA-00979LD and TIDA-00979LW boards are based on the layout guidelines from their corresponding datasheets. Depending on the LEDs and the forward current, a dedicated PCB layout must be made for proper heat dissipation.

4.5 Trademarks

LaunchPad, Internet-on-a-Chip, SimpleLink, Code Composer Studio are trademarks of Texas Instruments.
SIMPLE SWITCHER is a registered trademark of Texas Instruments.
ARM, Cortex are registered trademarks of ARM Ltd..
Wi-Fi is a registered trademark of Wi-Fi Alliance.

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265