Application Note 2227
Anssi Raisanen
August 27, 2012

LP5523 Evaluation Kit

Getting Started with LP5523

This document describes how to get the LP5523 evaluation board up and running, how to use evaluation and programming software, and how to get started with programming. The application note is divided into four separate sections: The first section provides general information for getting started with the LP5523 evaluation hardware and software. Section 2 shows LP5523 programming flow. Section 3 gives a detailed description of the device's instruction set. Finally, the fourth section gives practical programming examples.

1. General Description

The LP5523 provides flexibility and programmability for dimming and sequencing control. Each LED can be controlled directly and independently through the serial bus (in other words, without programming the engines), or LED drivers can be controlled by programming the execution engines. The LP5523 has three independent program execution engines, so it is possible to form three independently programmable LED banks. LED drivers can be grouped based on their function so that, for example, the first bank of drivers can be assigned to the keypad illumination, the second bank to the "funlights" and the third group to the indicator LED(s). Each bank can contain 1 to 9 LED driver outputs. Instructions for program execution engines are stored in the internal program memory. The total amount of the program memory is 96 instructons and the user can allocate the memory as required by the engines. The LP5523 is programmed using I²C-compatible serial bus. Of course, it is possible to write programs for the LP5523 in the form of binary data, but the programming tools described in this document offer a more convenient way to write (and read) the registers and the SRAM memory and to program the device.

WHAT IS NEEDED

To get started you will need:

- A text editor
- LP5523 evaluation kit hardware
- LP5523 evaluation software (LP5523.exe)
- LP5523 compiler (LASM.EXE)
- FTDI virtual com port drivers (VCP)

A text editor is used to create source code for the assembler. Here, we use PSPAd as a text editor, but you should feel free to use whatever editor you're most comfortable with. PSPad is a freeware editor, © 1991 - 2007 Jan Fiala. Please see the PSPad copyright notice. PSPad text editor can be downloaded from http://www.pspad.com/

INSTALLING FTDI DRIVERS

The evaluation board has FTDI's FT232R USB to UART chip on it. FT232R allows the PC to see the evaluation board as a virtual COM port. For this FTDI VCP drivers must be installed to the host computer. Usually Windows can install the drivers automatically when the evaluation board is detected. If Windows fails to install the drivers, they can be downloaded from www.ftdichip.com. Make sure that you download and install VCP drivers and not the D2XX drivers.

Once the FTDI drivers are installed and evaluation board is plugged into the USB port, Windows generates an USB serial port. This port is given unique COM port number. Note that the evaluation program polls only the first 8 COM ports. If the evaluation program does not find the evaluation board you may need to change the COM port number manually. In Windows go to the Control Panel, select System, select Hardware tab, open Device Manager, select Ports and you should see the USB Serial Port number. If the port number is above 8, you will need to change it. by right clicking the USB Serial Port, selecting Properties, select Port Settings, click Advanced button and select COM port number between 1 to 8.

COPYING THE SOFTWARE

PSPad does not require installation it can be simply unpacked into any directory. The archive contains subdirectories and must be unpacked with subdirectory preservation enabled. Also copy the Lysti.ini –file into the Syntax-folder of PSPad (for example C:\Program Files\PSPad editor\Syntax). Lysti.ini –file contains customization information for the text editor and it must be saved into the Syntax-folder.

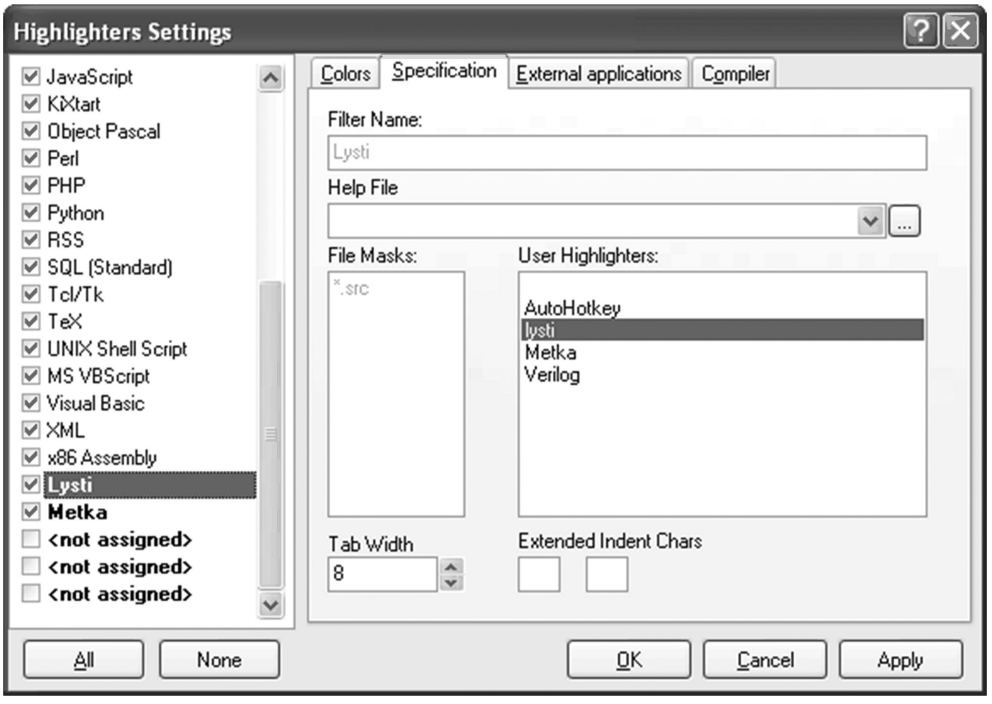
The LP5523 assembler (LASM.exe) and the LP5523 evaluation software can be copied to the PC's hard disk and run without installation. You will need to copy the following files: LP5523.exe, regmap.ini, usblptio.dll, rtl60.bpl and LASM.exe. All the files must lie in the same directory. Also the source code files which you will create (*.scr) should be saved/placed in the same directory as the LASM.exe before calling the assembler. Please avoid filenames or directory names containing a space character, since the assembler may fail when applied to filenames containing a space character. The evaluation software runs under Windows 2000/XP/Vista.

PSPAD CUSTOMIZATION

Once you run the PSPad editor, you should customize the editor for LP5523 as follows:

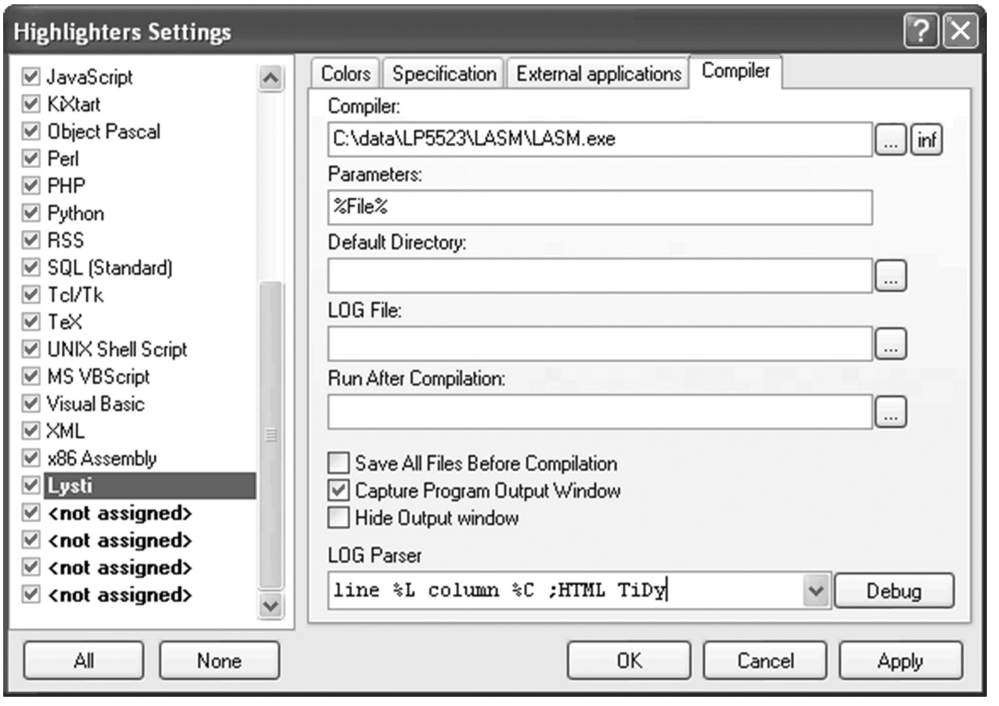
1. Select Settings menu > Highlighter settings.
2. Select Specification tab. See *Figure 1*.
3. On the left (highlighter) list, click on one of bolded highlighters (marked with **<not assigned>** -tab).
4. On the right side is list of user highlighters, select the Lysti highlighter and click on it (see *Figure 1*). Click on 'Apply' to confirm this action.

Also set the compiler search path and parameters for the LASM.exe, as shown in *Figure 2*. You may need to replace the shown filepath C:\data\LP5523\LASM\LASM.exe with your actual path. Tag the Capture Program Output Window as shown in *Figure 2*. Accept highlighter settings by clicking 'OK'. Finally, in the main window show the LOG window by pressing CTRL + L. All software needed should be now ready for writing and assembling the first program. Note: The maximum length of a source code line is 140 characters. Lines that are longer than 140 are not assembled correctly.



30187013

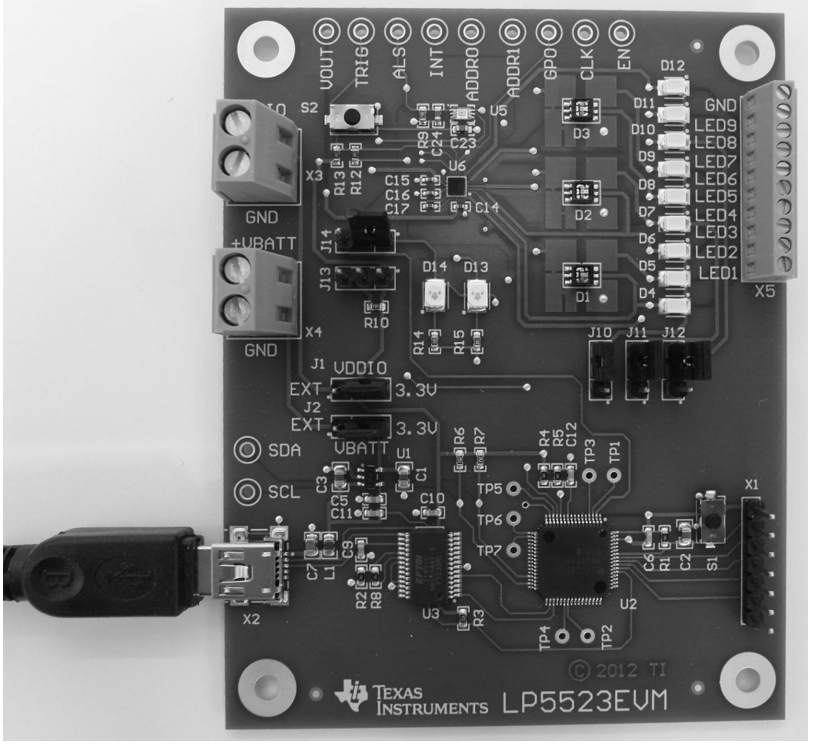
FIGURE 1. PSPad Highlighter Specification Settings for LP5523



30187014

FIGURE 2. PSPad compiler settings for LP5523. Note: These variable names are all case-sensitive so, for example %File % will work, but %file% and %FILE% will fail to produce the expected results.

HARDWARE SET-UP



30187025

FIGURE 3. LP5523 Evaluation Board

The LP5523 evaluation board has USB communication and evaluation-related components assembled onto one board. (See *Figure 3*.) The evaluation board was designed specially for evaluation; therefore, it is not optimized for the smallest layout size. The components are physically large to make changing of the value easy if needed. The LP5523 input voltage VDD is supplied from the USB port or from an external voltage applied to the X4 connector.

There are 7 pin connectors (jumpers) on the evaluation board for demonstrating some of the possible application options (see *Figure 3*). The voltage supplied to the V_{DD} input of the device can be selected using J2 connector. Connecting right and center pin with jumper selects that VDD is fed from USB, and connecting left and center pin with jumper selects that V_{DD} is fed from connector X4. Also whether V_{DDIO} is powered from USB or from external voltage supply (X3 connector) can be selected with J1. Connecting right and center pin with jumper selects that V_{DDIO} is fed from USB, and connecting left and center pin with jumper selects that V_{DDIO} is fed from connector X3. Voltage on the USB port is 5.0V, and the maximum current is 500 mA. There is a voltage regulator on the evaluation board which reduces the USB bus voltage to 3.3V. Connector X4 upper connection point is for VDD and lower for Ground. Connector X3 upper connection point is for V_{DDIO} and lower for Ground.

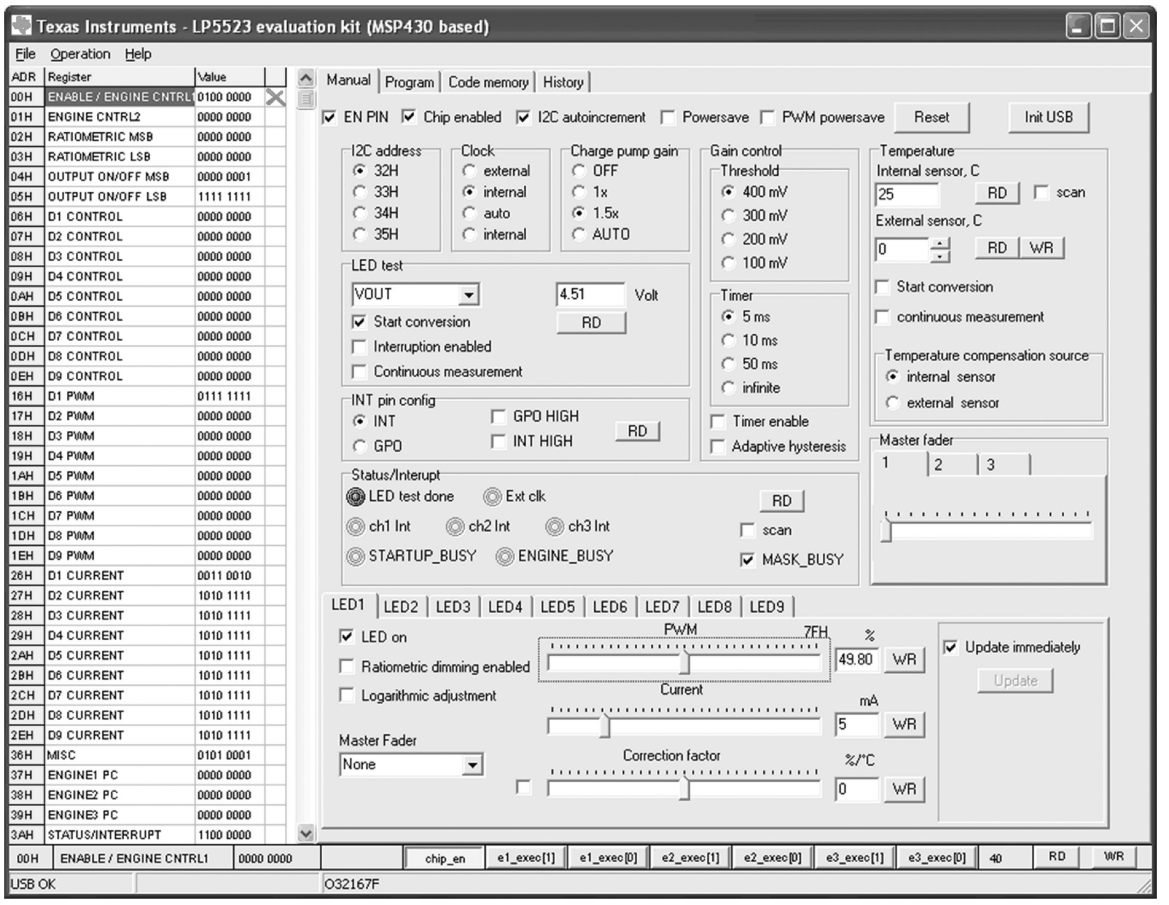
Pin connectors J10 J11, J12 are for selecting whether LP5523 LED outputs are connected to WLEDs or RGBs. If lower and center pins are connected with jumper, J10 connects output 1 to D1 green, output 2 to D1 blue and output 7 to D1 red. If upper and center pins are connected with jumper output 1 is connected to D4, output 2 to D5 and output 3 to D6. Pin connector J11 connects outputs 3 to D2 green, output 4 to D2 blue and output 8 to D2 red if lower and center pins connected with jumper. If upper and center pins are connected with jumper output 4 is connected to D7, output 5 to D8 and output 6 to D9. Pin connector J12 connects output 5 to D3 green, output 6 to D3 blue and output 9 D3 red if lower and center pins connected with jumper. If higher and center pins are connected with jumper output 5 is connected to D10, output 8 to D11 and output 9 to D12. It is recommended that either RGB or white LEDs are used and not mixed.

Pin connectors J13 and J14 are used to connect on board light sensor to LP5523. If J14 connectors left and center pins are connected with jumper light sensors output is connected to INT pin. If J14 connectors right and center pins are connected with jumper INT pin is connected to D14 LED. If J13 connectors left and center pins are connected with jumper light sensors GC1 pin is connected to the LP5523 GPO pin. In this case pulling GPO sets the light sensor into standby mode. If this function is not desired J13 can be left open. Light sensor's GC1 pin is pulled high with R10. *Table 1* summarizes the available options.

TABLE 1. Jumper Options

Jumper #	Left/Lower Pin	Center Pin	Right/Upper Pin
J1	VDDIO = external		
		VDDIO = 3.3V	
J2	VBATT = external		
		VBATT = 3.3V	
J10	RGB LED D1 in use (LED1 = G, LED2 = B, LED7 = R)		
		WLEDs in use (LED1 = D4, LED2 = D5, LED7 = D6)	
J11	RGB LED D2 in use (LED3 = G, LED4 = B, LED8 = R)		
		WLEDs in use (LED3 = D7, LED4 = D8, LED8 = D9)	
J12	RGB LED D3 in use (LED5 = G, LED6 = B, LED9 = R)		
		WLEDs in use (LED5 = D10, LED6 = D11, LED9 = D12)	
J13	Light sensor GC1 pin connected to LP5523 GPO pin. GPO can be used to disable light sensor.		
		Not needed. GC1 pulled up with R10 resistor.	
J14	INT pin connected to light sensor output.		
		INT pin connected to D14 LED.	

CONNECTING EVALUATION BOARD TO COMPUTER



30187001

FIGURE 4. LP5523 Evaluation Software Control Panel

1. Check that the jumpers on the evaluation board are on wanted positions.
2. Connect the evaluation board to your computer using a USB cable.
3. If this is first time you use the evaluation board install FTDI's VCP drivers.
4. Start the evaluation software *LP5523.exe*.
5. Click Init USB button.
6. Reset the LP5523 circuit by clicking the Reset button on upper right corner of the window. In the message box that appears, click OK to confirm the register reading.
7. The evaluation kit is now fully up and running and the device can be controlled through the PC software. *Figure 4* shows the evaluation software user interface (Control Panel).

You should see USB OK message on the status bar (shown in the lower part of the window). If the USB communication is not working correctly, shut down the evaluation software and unplug the USB cable. Plug in the USB cable again and reset the evaluation board by pressing the on board reset button (S1). Wait about 5 seconds and repeat steps 4 to 6 above. If the evaluation software is still unable to find evaluation board, you'll need to change com port settings as described in chapter *INSTALLING FTDI DRIVERS*.

CONTROLLING EVALUATION BOARD FROM PC

The evaluation software provides read-write control over the registers within the LP5523. Bits can be set from a logical '1' to a logical '0' or vice versa by a mouse click and for some settings there is also a slider control provided. The evaluation software window is divided into two panels: control panel and indicator panel (see *Figure 4*). The leftmost panel is the indicator panel, and it shows the status of the LP5523 registers (excluding the program memory).

The rightmost panel is the control panel — it is used to change the status of the registers, to program the device, or to debug the program depending on the selected view. The view is selected by the tabs on the top of the window. The Manual view appears when the evaluation software starts. The Manual view is used to run the device "manually," i.e,. the program execution engines are not used. It contains also some LP5523 basic settings, like serial bus address selection. In the next chapter the user is guided

through the Manual view and basic operations of the LP5523. The other views are Program, Code memory and History; these views are presented in *2. LP5523 Programming*.

Tip: The context-sensitive buttons in the lower part of the window can be used for direct Write/Read operations of the registers. The target register is indicated by red cross, and the target can be changed by clicking on the desired register on the indicator panel.

DIRECT CONTROL

Enabling the Device and Starting the Charge Pump

The first step to start with the LP5523 is to enable the chip; tag the Chip enabled check box. At first it is best to use the following settings: Clock internal and Charge pump gain 1.5x (see *Figure 4*). At this point it is good to ensure the charge pump output voltage with the LP5523 built-in LED error detection. To do this, select VOUT from the LED test drop down menu and tag Start conversion. Use the adjacent RD button to read the result: it should be around 4.5 volts. Also click on the RD button on the Status/Interrupt section to verify that the internal clock is used (Ext clk indicator light should be OFF) and that the LED test is done (LED test done indicator should be ON).

Setting the Led Current and PWM

The next step is to set the desired LED currents. This is done by Current slider. Set 5 mA current for LED1 and set PWM to 50% = 7Fhex. You should have a green LED switched on now. Note: When the Update immediately tag is set, the evaluation software will write the new settings immediately to the LP5523 registers. Otherwise you need to click on the Update button.

The basic idea behind LP5523 operation is to first set the LED currents to the required level and after that the current settings are not touched any more – all the fade-in and fade-out sections (ramps) and the temperature compensation is done by PWM . Each LED has its own constant current output and all the outputs can be controlled independently. Therefore LED pre-selection (matched LEDs) is not required and this kind of architecture supports also color control. The PWM, current and temperature compensation controls are organised under tabs labelled by LED1 to LED 9. Now, let's set 5 mA current and 50% PWM for LED2 and 3.5 mA current and 50% PWM for LED7. As a result, D1 on evaluation board should emit white light.

Master Fader

The PWM of LED driver outputs can be controlled individually as as described above, or alternatively the drivers can be grouped by function to provide a quick control. The LP5523 has three master fader registers, so it is possible to form three master fader groups. To assign an LED to a group, use the Master Fader drop down menu. Select Group 1 for LEDs 1, 2, and 7. Now you can control all the three outputs with a single master fader register. There is a Master fader slider provided on the right side of the control panel and dragging the slider under tab labelled by 1 controls now LEDs 1, 2, and 7. Note that the initial PWM and current for LEDs in a group needs to be set before using the master fader control, since master fader is simply a multiplier, which acts on the PWM registers.

Logarithmic vs Linear PWM Response

Logarithmic response is used to give the impression of a linear light intensity increase/decrease as the PWM duty cycle is raised/reduced. Logarithmic response is visually more pleasing especially when the overall brightness is low. To activate the logarithmic response, tag logarithmic adjustment. Activate logarithmic adjustment for LEDs 1, 2 and 7. Set also 5 mA current for LEDs 3 and 4, set 3.5mA for LED8. Set 50% PWM for LEDs 3, 4 and 8. Assign LEDs 3, 4 and 8 to the master fader group 1 so that you can control all the six LEDs with one slider. Now you should have logarithmic control over the LEDs 1, 2, 7 and linear control over the LEDs 3, 4, 8. To see the difference between the lin and log response, move the slider backwards and forwards to alter the intensity of all the six LEDs at once.

Temperature Compensation

The LP5523 has a temperature compensation function to correct for variations in light intensity and color caused by changes in ambient temperature.

Reset the LP5523 and start the charge pump as shown in *Figure 4*. Set 5 mA current, 50% PWM for LEDs 1 and 2; Set 3.5 mA current, 50% PWM for LED 7.

In order to activate the compensation function tag the check box next to Correction factor slider. The slope for the temperature compensation line can be set by the slider. A simple approximation for the RGB LED temperature compensation would be +1.3%/°C for red LED and +0,2%/°C for green and blue so that the intensity of all the colors remain approximately the same over the temperature from 25°C to 60°C .

To observe the effect of the compensation on color, try the following: Under Temperature label, tag Start conversion (this enables the LP5523 internal temperature sensor) and continuous measurement (continuous temperature measurement). Enter 25°C to the External sensor box and click on WR button to write the reading to the LP5523 memory. Now toggle between internal sensor and external sensor as you warm up the LEDs and LP5523 with a hair dryer. When the "external sensor" is active, the temperature information is read from register TEMPERATURE WRITE (addr. 40H). When the temperature is 25°C all the compensation settings have no effect, so when you have the external sensor activated, you will see the "uncompensated" situation. When you activate the internal sensor, you will see the effect of the compensation as the ambient temperature is raised. Without compensation emitted light will be drifted somewhat towards a more bluish white, because the red LED element of the RGB LED shows the strongest temperature dependence.

LED Error Detection

To measure VF of an LED set PWM = 100% for the LED and set the desired LED current with the current slider. Disable temperature compensation. On LED test portion tag Start conversion and Continuous measurement. Click on RD button to read the test result from LP5523's internal register. Try with different current settings and compare the result with the value specified by the manufacturer. If there is a short to ground in the LED circuit the result is ~0V and if there is an open the result is ~4,5V. This feature can also be addressed to measure the voltage on VDD, VOUT and INT pins. Typical example usage includes monitoring battery voltage or using INT pin as a light sensor interface (A/D converter value can be used as a variable in program control).

Charge Pump Gain Control

Charge pump gain can be forced to 1x, 1.5x, or you can let the LP5523 decide the best charge pump gain based on LED forward voltage requirements by selecting AUTO mode (this is the most useful setting in real applications). Note that outputs D7, D8, and D9 are powered directly from VDD and they have no effect on gain change decision-making. LP5523 charge pump has a gain change hysteresis which prevents the mode from toggling back and forth (1x -> 1.5x -> 1x...), which would cause ripple on VIN and LED flicker.

The hysteresis region width depends on the choice of threshold voltage. Threshold voltage is defined as follows VTHRESHOLD = VDD - MAX(voltage on D1 to D6). If VTHRESHOLD is larger than the set value (100mV to 400mV), the charge pump is in 1x mode. A good compromise solution between efficiency and performance is 200 mV. In addition to take the charge pump total load current into account enable the adaptive hysteresis.

Timer enable activates forced mode change. In forced mode charge pump mode change from 1.5x to 1x is attempted at the constant interval specified by the Timer control. With infinite setting the charge pump switches gain from 1x mode to 1.5x mode only. The gain reset back to 1x is enabled under certain conditions, for example in the power-save mode. Forced mode and treshold parameters are used to optimize efficiency in a real design - it is desired that 1.5x gain is used only when needed.

2. LP5523 Programming

PROGRAMMING FLOW CHART

Figure 5 shows the typical programming flow of the LP5523. The program is first typed in with PSPad (or equivalent) text editor. Then the program is compiled into a hex and binary file. Finally the hex file is loaded into the LP5523's memory and tested.

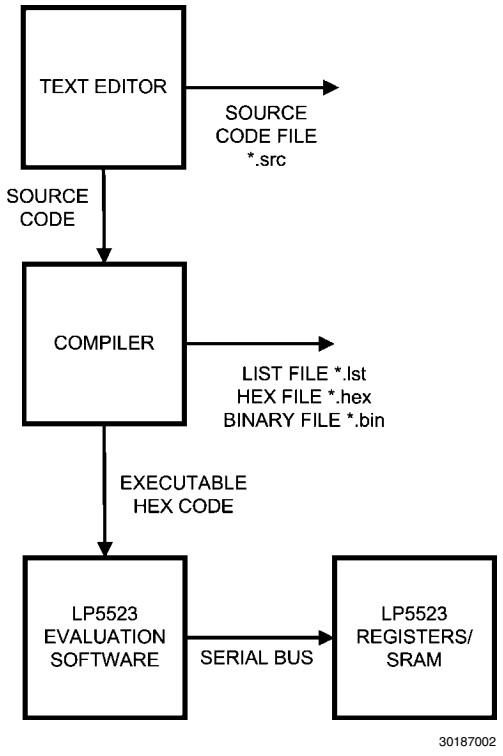


FIGURE 5. Programming Flow Chart

RESERVED KEYWORDS

The names of registers and instructions are assembler-reserved keywords. For the LP5523, the following words are reserved and may not be used as statement labels:

Register names

- ra
- rb
- rc
- rd

Instructions

- ramp
- set_pwm
- wait
- mux_ld_start
- mux_ld_end
- mux_map_start
- mux_sel
- mux_clr
- mux_map_next
- mux_map_prev
- mux_ld_next
- mux_ld_prev
- mux_ld_addr
- mux_map_addr
- rst

- branch
- int
- end
- trigger
- jne
- jl
- jge
- je
- ld
- add
- sub

Directives

- .segment
- ds
- dw

THE STRUCTURE OF A LP5523 PROGRAM

Figure 6 shows an example of a LP5523 program. When this program is run, the program will flash a red LED once per second. Although this program is short and simple it shows all the main parts of a typical LP5523 program.

Commenting

Commenting starts with a semicolon (;). The compiler will ignore all characters after a semicolon. If you are using PSPad editor and have customized the editor according to the instructions on first pages of this document, the editor recognizes comments, directives, labels and instructions automatically and uses different highlighter colors for different datatypes.

Directives

The directives are not translated directly into the LP5523. Instead, directives are instructions for the LASM.exe compiler. Directives are used to adjust the location of the engine 1, 2 and 3 programs in memory and reserve memory resources in the LP5523 SRAM. For example **.segment** program1 is a directive which tells to the compiler that whatever follows is the program for the program execution engine 1. An overview of the directives is given in the following table.

Directive	Description	Example source code
.segment	Adjust the location of the programs in SRAM. Note the leading dot	**.segment** program1 **.segment** flashing_light
ds	**D**efine **S**torage; The directive reserves memory resources in the SRAM. The ds directive takes one parameter, which is the number of words to reserve. The number of bits in a word (word length) is 16. The allocated words are initialized with zeros	**ds** 3 **ds** 17
dw	**D**efine constant **W**ord. Inserts a binary word to the SRAM.	**dw** 0000000011111111b **dw** FFABh **dw** 3

Labels

A label is a symbolic address. Labels are used to mark program line(s), like in branch instruction and labeling mapping table rows. Labels must have the colon (:) suffix. In the *Figure 6* code *loop1:* is a label which indicates the starting address of the loop.

Instructions

Instructions are executable statements. LASM-compiler translates text-based language source instructions into hex- and binary-based executable codes. In the example code presented in *Figure 6* for example **set_pwm** is an instruction. Almost all the instructions take operands, which may be constants (like FFh), or variables stored in the LP5523 registers (like ra) - ra stands for variable a (register a), rb, for variable b, etc.

```
.segment program1               ;Beginning of a segment.
                 mux_sel 1      ;select LED1
loop1:           set_pwm 255    ;beginning of a loop, set PWM full scale.
                 wait 0.48      ;wait for 0.48 seconds.
                 set_pwm 0      ;set PWM to 0%
                 wait 0.48      ;wait for 0.48 seconds.
                 branch 0, loop1 ;endless loop
```

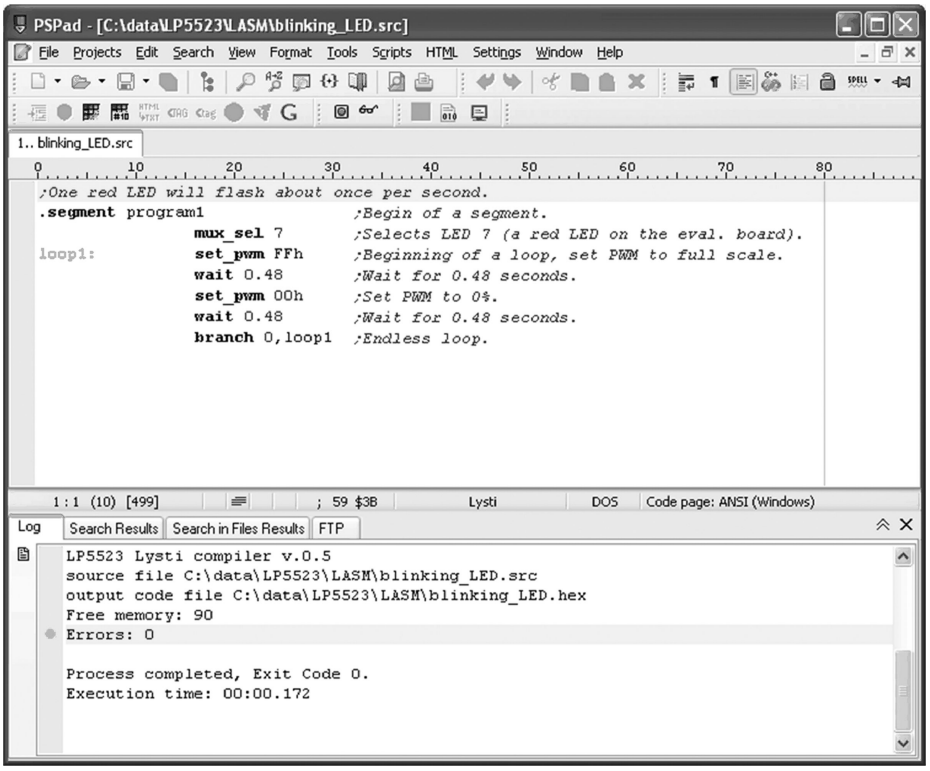
30187015

FIGURE 6. Example Code of LP5523 Program

PRODUCING AN EXECUTABLE FILE

Once the text-based source file is typed in and saved using the text editor (PSPad), the source code window should look like the one in *Figure 7*. To call the compiler routine, select *File >> Compile*. The PSPad LOG window shows the progress of compiling. If the compiler generates error messages, LOG window is necessary for locating these errors.

A listing file, a hex file and a binary file is produced by the LASM.exe compiler. The name of the files are the same name that you have given to the source code file, with the *.lst, *.hex or *.bin extension. *.hex and *.bin files contain the machine code.



30187010

FIGURE 7. Example of Compiling Source Code

LOADING A PROGRAM TO THE LP5523'S SRAM

To upload code into the LP5523 SRAM start the LP5523.exe evaluation software. Select the Program tab *Figure 10*. The Program tab is divided into two parts: the right contains the program's source code and the compiled version of the code; the left part contains program execution engine controls. Load the generated *.hex file into the evaluation software view: Click Open Source File button (*Figure 8*), browse the file and click Open.

30187012

FIGURE 8. Open Source File Button In Program Tab

To download the machine code into the LP5523, click on download button *Figure 9*. Pressing this button sets all the engine modes to *Load*.

30187016

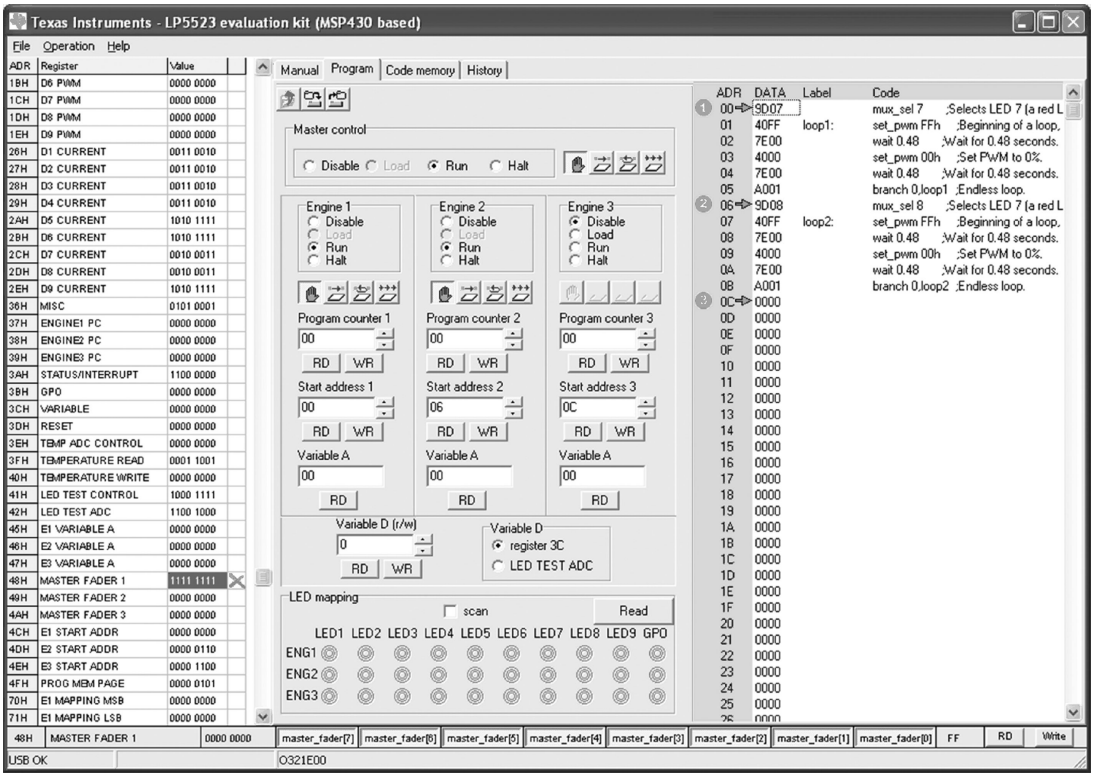
FIGURE 9. Download Opened Source File Into LP5523

Program can be loaded also from the *Code memory* tab *Figure 11*. In this case one must set the operation mode to *Load*. In *Code memory* tab one must first select the address where data is written. Once the address is active Data can be written in hexadecimal

to the Data entry field and push the Update button. Once all the data is updated to the wanted addresses, it can be written to the SRAM memory by pushing the Write Page, which writes two lines in code memory table (first two lines refer to page 0, next two lines to page 1, etc. Pages can be changed with the page selector) or by pushing Write All, which updates the whole memory. Once the data is written to SRAM, operation mode can be set to *Run* and Execution mode for example to *Free Run*. Note that Program Counter(s) must be set accordingly. This is not done automatically like loading program by clicking the Download-button. Note also that the program code does not show up in the program view (in *Program* tab).

RUNNING THE PROGRAM

The program is run by checking the Run from the Master control and clicking on Free run-button. This way all the engines will start at the same time. If you have less than three engines in use, extra engines must be disabled by checking off the box in each engine section. As seen in *Figure 10*, the program has only two engines in use, and the engine three is checked off. If this is not done the programs may not work correctly. Engines can also be run from individual engine control. One convenient way of debugging the programs is by running the individual engines step by step. Individual engine control can be used also with multiple engines, but then they wont start at the same time.



30187017

FIGURE 10. Master Control of the Program Tab

For operating, the program's following four modes exist (see *Figure 10*). Operation mode is selected by clicking the desired check box.

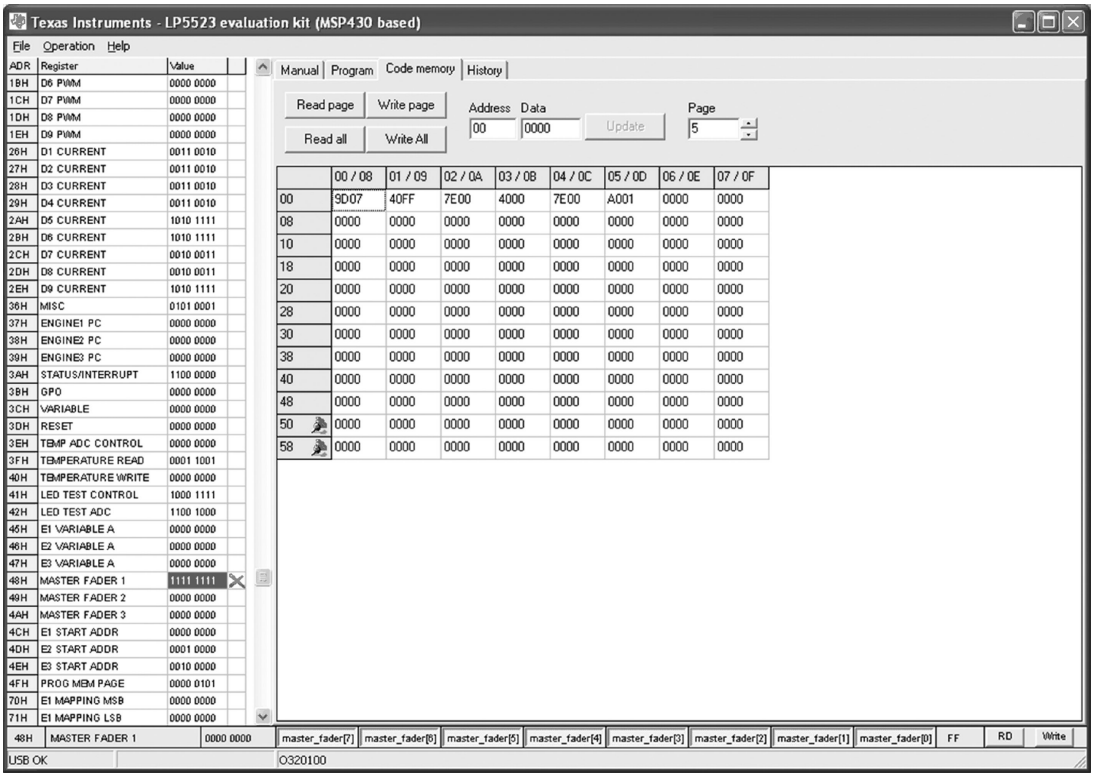
Operation modes

- **Disable** — Engine operation is disabled and they can not be run
- **Load** — In this mode writing to program memory is allowed. All the three engines are in hold while one or more engines are in load program mode. PWM values are also frozen. Program execution continues when all the engines are out of load program mode. Load program mode resets the program counter of the respective engine. Load program mode can be entered from the disabled mode only. Entering load program mode from the run program mode is not allowed. Note that load mode does not automatically load the program opened with Open Source File button *Figure 8*. When using this operation mode, one must write the program through the Code memory tab .
- **Run** — This mode executes the instructions stored in the program memory. Execution register (ENG1_EXEC etc.) bits define how the program is executed (hold, step, free run or execute once). Program start address can be programmed to Program Counter (PC) register. The Program Counter is reset to zero when the PC's upper limit value is reached.
- **Halt** — In this mode instruction execution aborts immediately and engine operation halts. Execution can be continued if operation mode is set to Run again.

For executing the programs following four modes exist. Execution mode is selected with the four push buttons. Functions of the buttons from left to right are:

Executions modes

- **Stop** — Engine execution is stopped. The current instruction is executed and then execution stopped.
- **Step** — Execute the instruction at the location pointed by the Program Counter, increment the Program Counter by one and then reset ENG1_EXEC bits to 00 (enter Stop-mode).
- **Execute Command** — Execute the instruction pointed by the current Program Counter value and reset ENG1_EXEC to 00 (i.e. enter Stop-mode). The difference between Step and Execute Command is that Execute Command does not increment the Program Counter.
- **Free Run** — Start program execution from the instruction pointed by the Program Counter.



30187021

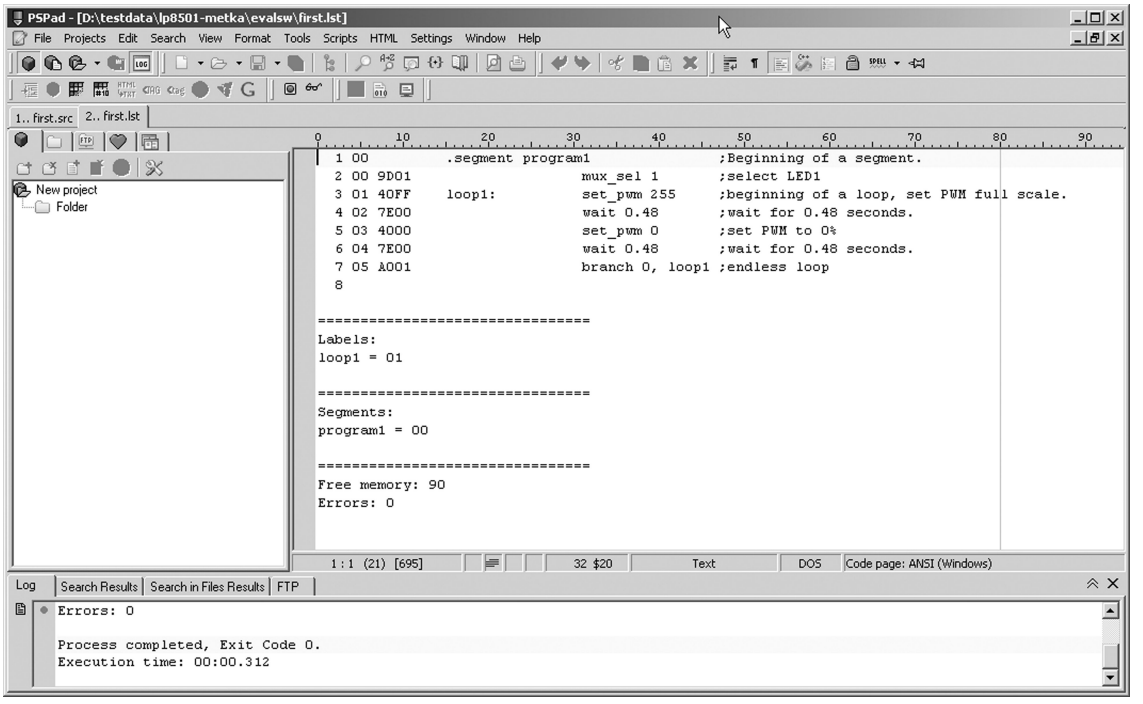
FIGURE 11. Code Memory Tab

DEBUGGING CONSIDERATIONS

There are a few ways to see how the compiler translates the instructions to machine code. Listing file (*.lst) may be used for locating assembling errors. The listing file contains the source code along with the compiled machine code. You can examine the files in any text editor. This is helpful for debugging and seeing how source code is translated into machine code. *Figure 12* shows an example of listing file. The first column is the row number, second column indicates the SRAM memory address, third shows the machine code data and fourth column includes the source code. Note that the .segment directives show the start address of the program, i.e., where to the Program Counters should point.

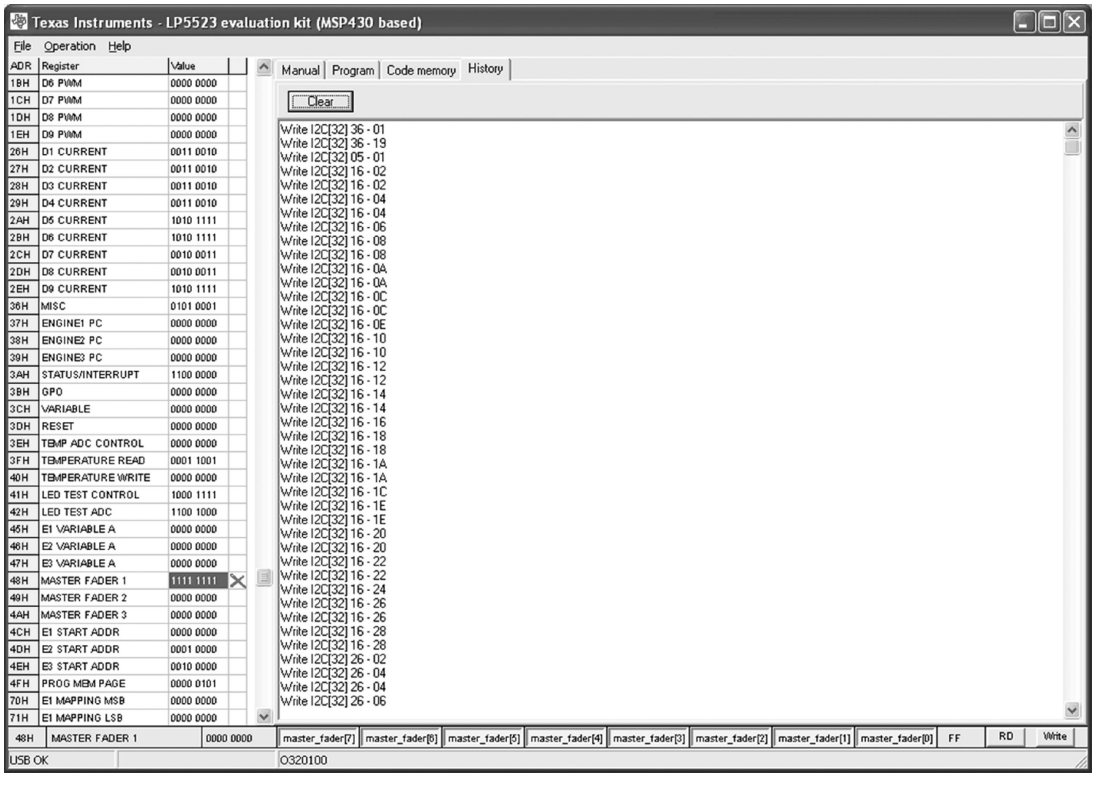
From the produced two hex-files user can see the pure machine code represented in hexadecimal in two different ways. In *.he2 —file representation of data is 0xYY (YY being the changing data information). In *.hex file the data is represented YY, where YY is the changing data information. In *.he2 file the 8-bit long data elements are separated with comma whereas in *.hex file they are separated with tab. In both files data is represented like in *Code memory* tab memory table. First two columns correspond to first column in *Code memory* tab memory table, third and fourth columns correspond to second column etc. For example, if the user would have mux_inc instruction (9D80), in he2 it would be 0x9D, 0x80, and in hex file it would be 9D [tab] 80. In hex-file the start addresses of the programs are at the bottom whereas in *.he2 file they are on the first row engine 1 start address being first, engine 2 second, and engine 3 start address third. Also in the bin-file user can see the pure machine code represented in binary. First three rows represent the start addresses of the programs. After the start addresses the program code follows.

Programs can be debugged in the evaluation software *Program* tab by running the program in steps using *Step* or *Execute command* execution modes. Also one way to see what is written to the LP5523 is to look at the evaluation program *History* tab *Figure 13*.



30187011

FIGURE 12. Listing File of the Example Program First



30187022

FIGURE 13. History Tab

3. Instruction Set Details

This section provides the syntax with detailed examples for all the LP5523 instructions supported by the LASM assembler.

LED DRIVER INSTRUCTIONS

INSTRUCTION SYNTAX	FUNCTION	EXAMPLE	16-BIT ASSEMBLED BIT SEQUENCE	ASSEMBLED CODE HEX
ramp time, PWM Time is a positive constant (0.000484*PWM to 0.484*PWM); PWM is a positive or negative constant (-255 to 255). Note: time is rounded by assembler if needed.	Output PWM with increasing / decreasing duty cycle.	ramp 0.6, 255 ;Ramp up to full scale over 0.6s	0000 1010 1111 1111	0A FF
		ramp 1.2,-255 ;Ramp down to zero over 1.2s	0001 0101 1111 1111	15 FF

ramp var1, prescale, var2
Var1 is a variable (ra, rb, rc, rd);
Prescale is a boolean constant (pre=0 or pre=1);
Var2 is a variable (ra, rb, rc, rd).
	Output PWM with increasing / decreasing duty cycle.	ld ra, 31
		ld rb, 255
		ramp ra, pre=0,+rb
		;Ramp up to full scale over 3.9s
			1000 0100 0000 0001	84 01

		ld ra, 1
		ld rb, 255
		ramp ra, pre=0,-rb
		;Ramp down to zero over 0.12s
			1000 0100 0001 0001	84 11

set_pwm PWM PWM is a constant (0-255 or 0 - FFh).	Generate a continuous PWM output.	set_pwm 128 ;Set PWM Duty-Cycle to 50%	0100 0000 1000 0000	4080
set_pwm var1 Var1 is a variable (ra, rb, rc, rd).	Generate a continuous PWM output.	ld rc, 128 set_pwm rc ;Set PWM Duty-Cycle to 50%	1000 0100 0110 0010	8462
wait time Time is a positive constant (0 to 0.484). Note: time is rounded by assembler if needed.	Pause for some time.	wait 0.25 ;Wait 0.25 seconds	0110 0000 0000 0000	6000

LED MAPPING INSTRUCTIONS

INSTRUCTION SYNTAX	FUNCTION	EXAMPLE	16-BIT ASSEMBLED BIT SEQUENCE	ASSEMBLED CODE HEX
mux_ld_start address Address is a label which specifies where to find the first row.	Defines the start address of the mapping data table.	mux_ld_start row1 ; The first row can be found at the address marked with row1	1001 1110 0000 0000 Assumed that row1 points to addr 00h.	9E00
mux_map_start address Address is a label which specifies where to find the first row.	Defines the start address of the mapping data table and sets the row active.	mux_map_start row1 ; The first row can be found at the address marked with row1	1001 1100 0000 0000 Assumed that row1 points to addr 00h.	9C00
mux_ld_end address Address is a label which specifies where to find the last row.	Defines the end address of the mapping data table.	mux_ld_end row9 ; The last row can be found at the address marked with row9	1001 1100 1000 1000 Assumed that row9 points to addr 08h.	9C88
mux_sel output Output is a constant (0 to 9 or 16).	Connects one and only one LED output to an engine.	mux_sel 1 ; D1 output will be connected to the engine.	1001 1101 0000 0001	9D01
mux_clr	Clears engine-to-driver mapping.	mux_clr	1001 1101 0000 0000	9D00
mux_map_next	Sets the next row active in the mapping table.	mux_map_next	1001 1101 1000 0000	9D80
mux_map_prev	Sets the previous row active in the mapping table.	mux_map_prev	1001 1101 1100 0000	9DC0
mux_ld_next	The index pointer will be set to point to the next row in the mapping table.	mux_ld_next	1001 1101 1000 0000	9D81
mux_ld_prev	The index pointer will be set to point to the previous row in the mapping table.	mux_ld_prev	1001 1101 1100 0000	9DC1
mux_ld_addr address Address is a label which specifies the row to which the pointer is to be moved.	Sets the index pointer to point the mapping table row defined by address.	mux_ld_addr row2 ; The index pointer will be set to point to the row labelled with row2.	1001 1111 0000 0001 Assumed that row2 points to addr 01h.	9F01
mux_map_addr address Address is a label which specifies the row of the table that will be set active.	Sets the index pointer to point the mapping table row defined by address and sets the row active.	mux_map_addr row2 ; The index pointer will be set to point to the row labelled with row2 and the row will be set active.	1001 1111 1000 0001 Assumed that row2 points to addr 01h.	9F81

BRANCH INSTRUCTIONS

INSTRUCTION SYNTAX	FUNCTION	EXAMPLE	16-BIT ASSEMBLED BIT SEQUENCE	ASSEMBLED CODE HEX
rst	Resets program counter and start the program again.	rst	0000 0000 0000 0000	0000
branch loopcount, address Loopcount is a constant (0 to 63); Address is a label which specifies the offset.	Repeat a section of code.	branch 20, loop1 ; define loop for 20 times	1010 1010 0000 0000 Assumed that loop1 points to addr 00h.	AA00
branch var1, address Var1 is a variable (ra, rb, rc, rd); Address is a label which specifies the offset.	Repeat a section of code.	ld ra, 20 branch ra, loop1 ; define loop for 20 times	1000 0110 0000 0000 Assumed that loop1 points to addr 00h.	8600
int	Causes an interrupt.	int	1100 0100 0000 0000	C400
end interrupt, reset Interrupt (i) is an optional flag. Reset (r) is an optional flag.	End program execution.	end i ; End program execution and send an interrupt.	1101 0000 0000 0000	D000
trigger w{source1|source2...} Source is the source of the trigger (1, 2, 3, e).	Wait a trigger.	trigger w{1} ;Wait a trigger from the engine 1.	1110 0000 1000 0000	E080
trigger s{target1|target2...} Target is the target of the trigger (1, 2, 3, e).	Send a trigger.	trigger s{1} ;Send a trigger to the engine 1.	1110 0000 0000 0010	E002
jne var1, var2, address Var1 is a variable (ra, rb, rc, rd); Var2 is a variable (ra, rb, rc, rd); Address is a label which specifies the offset.	Jump if not equal.	jne ra, rb, flash ;Jump to flash if A != B.	1000 1000 0010 0001 Assumed that offset = 2.	8821
jl var1, var2, address Var1 is a variable (ra, rb, rc, rd); Var2 is a variable (ra, rb, rc, rd); Address is a label which specifies the offset.	Jump if less.	jl ra, rb, flash ;Jump to flash if A < B.	1000 1010 0001 0001 Assumed that offset = 1	8A11
jge var1, var2, address Var1 is a variable (ra, rb, rc, rd); Var2 is a variable (ra, rb, rc, rd); Address is a label which specifies the offset.	Jump if greater or equal.	jge ra, rb, flash ;Jump to flash if A >= B.	1000 1100 0001 0001 Assumed that offset = 1.	8C11
je var1, var2, address Var1 is a variable (ra, rb, rc, rd); Var2 is a variable (ra, rb, rc, rd); Address is a label which specifies the offset.	Jump if equal.	je ra, rb, flash ;Jump to flash if A = B.	1000 1110 0001 0001 Assumed that offset = 1.	8E11

DATA TRANSFER AND ARITHMETIC INSTRUCTIONS

INSTRUCTION SYNTAX	FUNCTION	EXAMPLE	16-BIT ASSEMBLED BIT SEQUENCE	ASSEMBLED CODE HEX
ld var, value Var is a variable (ra, rb, rc); Value is a constant (0 to 255 or 0 to FFh).	Assigns a value to a variable.	ld ra, 10 ;Variable A = 10.	1001 0000 0000 1010	900A
add var, value Var is a variable (ra, rb, rc); Value is a constant (0 to 255 or 0 to FFh).	Add the 8-bit value to the variable value.	add ra, 30 ;A = A + 30.	1001 0001 0001 1110	911E
add var1, var2, var3 Var1 is a variable (ra, rb, rc); Var2 is a variable (ra, rb, rc, rd); Var3 is a variable (ra, rb, rc, rd);	Add the value of var3 to the value of var2 and store the result in var1.	add ra, rc, rd ;A = C + D.	1001 0011 0000 1010	930B
sub var, value Var is a variable (ra, rb, rc); Value is a constant (0 to 255 or 0 to FFh).	Subtract the 8-bit value from the variable value.	sub ra, 30 ;A = A - 30.	1001 0010 0001 1110	921E
sub var1, var2, var3 Var1 is a variable (ra, rb, rc); Var2 is a variable (ra, rb, rc, rd); Var3 is a variable (ra, rb, rc, rd);	Subtract the value of var3 from the value of var2 and store the result in var1.	sub ra, rc, rd ;A = C - D	1001 0011 0001 1011	931B

4. Programming Examples

This section gives practical programming examples. A series of programs introduce the main features of the LP5523 chip and the example programs are easy to tailor to the specific needs of end application.

EXAMPLE 1: CONTROLLING MULTIPLE LEDS WITH ONE ENGINE

The example below is basically the program shown in the *Figure 6* above (simple LP5523 program);, the LED mapping table below is used to establish engine1-to-LEDs connection. In the mapping table '0' means that the LED isn't connected to the engine; '1' means that the LED is connected to the engine. In the example program below bits 6 to 8 are set to '1' so that outputs 7, 8 and 9 are mapped to the engine 1.

LED Mapping Chart

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Output	GPO	N/A	N/A	N/A	N/A	N/A	N/A	D9	D8	D7	D6	D5	D4	D3	D2	D1

```
mapping_table:          dw 0000000111000000b        ;Red LEDs on the evaluation board.
                                                    ;'1' = LED is mapped; '0'= LED isn't mapped.
.segment program1                                   ;Begin of a segment.
                        mux_map_start mapping_table ;Mapping table start address in the memory.
                                                    ;The first row of the table will be activated
loop1:                  set_pwm FFh                 ;Beginning of a loop, set PWM to full scale.
                        wait 0.48                   ;Wait for 0.48 seconds.
                        set_pwm 00h                 ;Set PWM to 0%.
                        wait 0.48                   ;Wait for 0.48 seconds.
                        branch 0,loop1              ;Endless loop.
```

EXAMPLE 2: FADE-IN/FADEOUT EFFECTS AND CHANGING MAPPING OF LEDS

In this example ramp instruction is used to produce fade-in/fade-out effects. Note: The use of a logarithmic PWM profile with ramp instruction ensures the light and color changes appear smooth to the human eye. The same effect is repeated for all the LEDs by changing the engine1-to-LED mapping. After that, the intensity of the LEDs is increased gradually and finally all the LEDs are set OFF.

```
row1:           dw 0000000000000001b    ;Map green LED = D1 on the eval. board.
                dw 0000000000000010b    ;Map blue LED = D2 on the eval. board.
                dw 0000000001000000b    ;Map red LED = D7 on the eval. board.
                dw 0000000000000100b    ;Map green LED = D3 on the eval. board.
                dw 0000000000001000b    ;Map blue LED = D4 on the eval. board.
                dw 0000000010000000b    ;Map red LED = D8 on the eval. board.
                dw 0000000000010000b    ;Map green LED = D5 on the eval. board.
                dw 0000000000100000b    ;Map blue LED = D6 on the eval. board.
row9:           dw 0000000100000000b    ;Map red LED = D9 on the eval. board.
row10:          dw 0000000111111111b    ;Map all LEDs on the eval. board.

.segment program1                       ;Program for engine 1.
                mux_map_start row1      ;Map the first LED.
                mux_ld_end row9         ;End address of the mapping data table.
loop1:          ramp 1.0, 255           ;Increase PWM 0->100% in 1 second.
                ramp 1.5, -255          ;Decrease PWM 100->0% in 1.5 seconds.
                wait 0.4;               ;Wait for 0.4 seconds.
                mux_map_next            ;Set the next row active in the mapping table.
                branch 8,loop1          ;Loop 8 times
loop2:          ramp 0.5, 128           ;Increase PWM by 128 steps in 0.5 second.
                mux_map_next            ;Set the next row active in the mapping table.
                                        ;Note roll over of the mapping.
                branch 17, loop2        ;Loop 17 times.
                mux_map_addr row10      ;Set row10 active (all LEDs mapped).
                set_pwm 0;              ;Set all the LEDs OFF.
rst                                     ;Reset program counter and start the program again.

.segment program2
                                        ;Program for engine 2 (empty).
rst

.segment program3
                                        ;Program for engine 3(empty).
rst
```

EXAMPLE 3: USING MORE THAN ONE ENGINE AND NESTED LOOPS (LOOP INSIDE A LOOP)

This program below shows how to use all the three engines simultaneously. The engines are used to create a police beacon-type light effect. Hint: To see the effect of separate engines, start/stop engines separately using Engine 1, Engine 2 and Engine 3 controls instead of the Master control.

```
blue1:          dw 0000000000000010b    ;Map blue LED on D2.
blue2:          dw 0000000000100000b    ;Map blue LED on D6.
blue3:          dw 0000000000001000b    ;Map blue LED on D4.
red_led:        dw 0000000010000000b    ;Map red LED on D8.

.segment program1                       ;Program for blue LEDs on D2 and D6.
                mux_map_start blue1     ;Mapping table start address.
                mux_ld_end blue2        ;Mapping table end address.
loop2:
loop1:          set_pwm 255
                wait 0.1
                set_pwm 0
                wait 0.05
                branch 1, loop1
                wait 0.2
                mux_map_next
                branch 3, loop2         ;Note nested loop (loop1 is within loop2).
loop4:
loop3:          set_pwm 255
                wait 0.05
                set_pwm 0
                wait 0.05
                branch 5, loop3
                mux_map_next
                branch 3, loop4
                rst

.segment program2                       ;Program for red LED on D8.
                mux_map_addr red_led    ;Red LED mapping.
                wait 0.45
                ramp 0.5, 255           ;PWM 0%->100% in 0.5 second.
                ramp 1, -255            ;PWM 100%->0% in 1 second.
                rst

.segment program3                       ;Program for blue LED on D4.
                mux_map_start blue3     ;Mapping table start address.
                set_pwm 255
                wait 0.05
                set_pwm 0
                wait 0.2
                set_pwm 255
                wait 0.05
                set_pwm 0
                wait 0.45
                rst
```

EXAMPLE 4: TRIGGERS AND INTERRUPT

This program shows how to use triggers: the interrupt. Engine 1 sends a trigger to Engine 2 when it has set an LED to 100% PWM – Engine 2 fades the LED out to 0% PWM. Engine 1 sends an interrupt when it has finished loop1 and waits trigger from Engine 2 AND an external trigger. Pushing the button on the evaluation board causes an external trigger and the sequence starts over. Use RD button on the Status/Interrupt section (see *Figure 5*) to clear the interrupt.

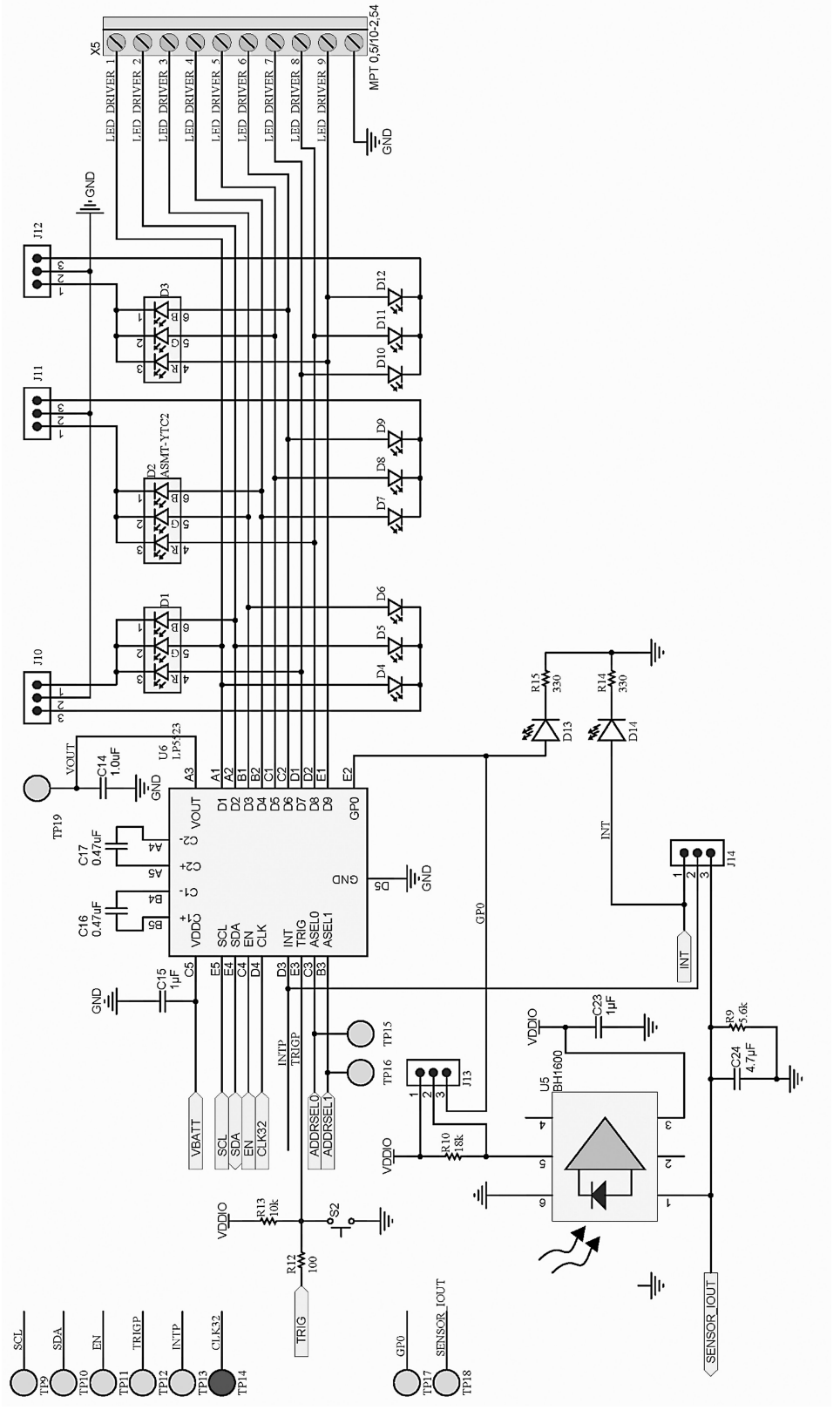
```
row1:       dw 0000000000000001b    ;Map green LED = D1 on the eval. board.
            dw 0000000000000010b    ;Map blue LED = D2 on the eval. board.
            dw 0000000001000000b    ;Map red LED = D7 on the eval. board.
            dw 0000000000000100b    ;Map green LED = D3 on the eval. board.
            dw 0000000000001000b    ;Map blue LED = D4 on the eval. board.
            dw 0000000010000000b    ;Map red LED = D8 on the eval. board.
            dw 0000000000010000b    ;Map green LED = D5 on the eval. board.
            dw 0000000000100000b    ;Map blue LED = D6 on the eval. board.
row9:       dw 0000000100000000b    ;Map red LED = D9 on the eval. board.

.segment program1                   ;Program for engine 1.
            mux_map_start row1      ;Map the first LED.
            mux_ld_end row9         ;End address of the mapping data table.
loop1:      ramp 1.0, 255           ;Increase PWM 0->100% in 1 second.
            trigger s{2}            ;Send trigger to engine2
            mux_map_next            ;Set the next row active in the mapping table.
            branch 8,loop1          ;Loop 8 times.
            int                     ;Send an interrupt.
            trigger w{2|e}          ;Wait trigger from engine 2 and external trigger.
            rst                     ;Reset program counter and start the program again.

.segment program2                   ;Program for engine 2.
            mux_map_start row1      ;Map the first LED.
            mux_ld_end row9         ;End address of the mapping data table.
loop2:      trigger w{1}            ;Wait for trigger from engine1.
            ramp 1.0,-255           ;Decrease PWM 100->0% in 1 second.
            mux_map_next            ;Set the next row active in the mapping table.
            branch 8,loop2          ;Loop 8 times.
            trigger s{1}            ;Send trigger to engine1.
            rst

.segment program3
                                    ;Program for engine 3(empty).
rst
```

Schematic and Layout



30187023

FIGURE 14. Schematics of the Evaluation Board Application Side

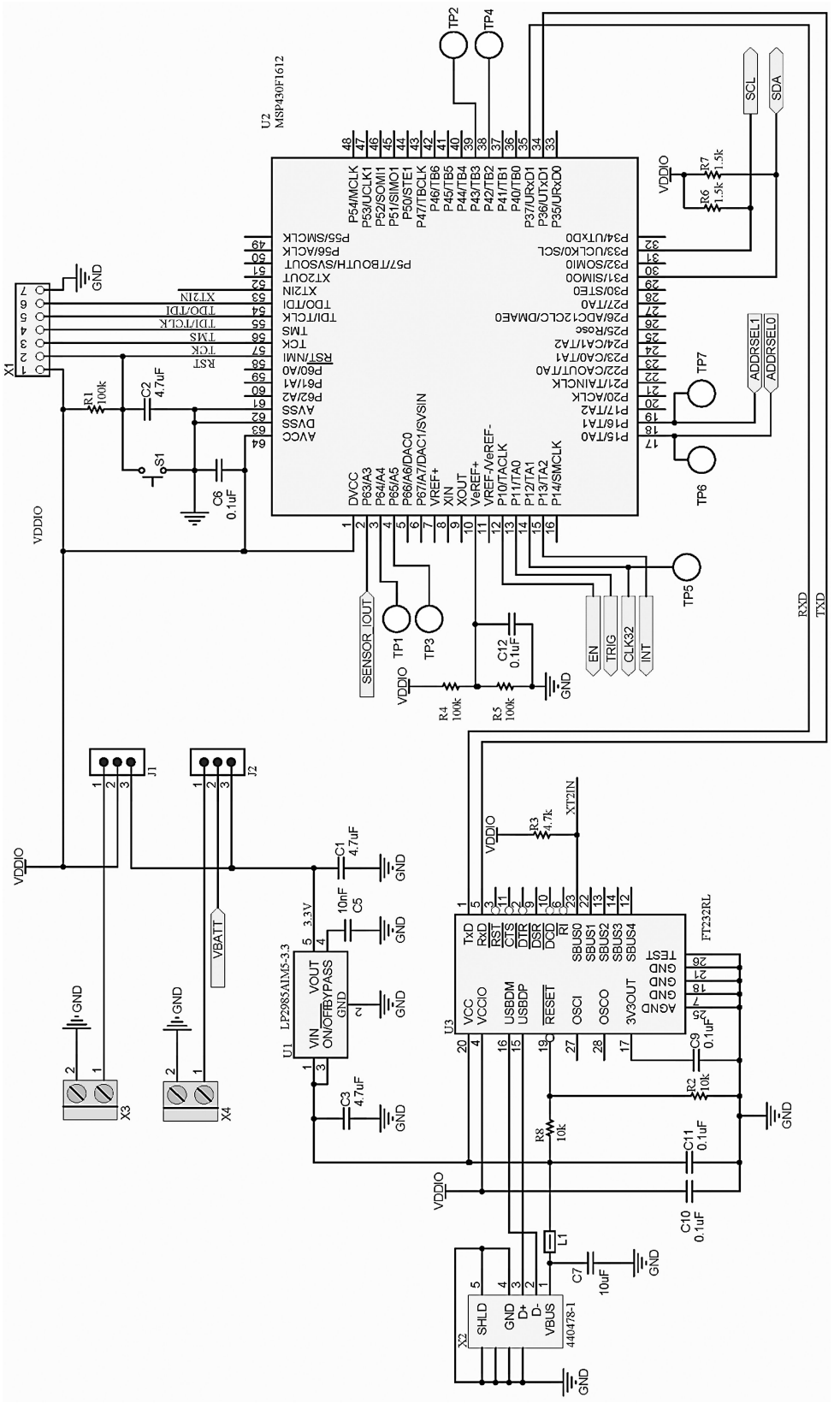


FIGURE 15. Schematics of the Evaluation Board USB Side

Bill of Materials

Designator	Quantity	Part Number	Description	Value	Footprint
U6	1	LP5523TM	Lighting management unit		25-bump micro SMD
U3	1	FT232RL	USB to UART		28-SSOP
U2	1	MSP430F1612IPM	microcontroller		64-LQFP
U1	1	LP2985AIM5-3.3	LDO regulator		0603L
U5	1	BH1600FVC-TR	Light Sensor		
D1, D2, D3	3	ASMT-YTC2-0AA02	RGB LED		6-PLCC
D13, D14	2	HSMS-A100-J00J1	Red LED		2-PLLC
D4, D5, D6, D7, D8, D9, D10, D11, D12	9	CLM3C-WKW-CWBYA453	White LED		2-PLCC
C1, C2, C3	3	LMK212B7475KG-T	Ceramic capacitor	4.7uF, 10V	0805
C5	1	C0603C103J5RACTU	Ceramic capacitor	10nF , 50V	0603
C6, C9, C10, C11, C12	5	C0603C104K8RACTU	Ceramic capacitor	0.1uF , 10V	0603
C7	1	C2012Y5V1A106Z	Ceramic capacitor	10uF, 10V	0805
C15, C14	2	LMK105BJ105KV-F	Ceramic capacitor	1uF, 10V	0402
C16, C17	2	JMK105BJ474KV-F	Ceramic capacitor	0.47uF, 6.3V	0402
C23	1	C0603C105Z8VACTU	Ceramic capacitor	1uF , 10V	0603
C24	1	C0603C475K8PACTU	Ceramic capacitor	4.7uF , 10V	0603
R1, R4, R5	3	ERJ-3GEYJ104V	Resistor	100k	0603
R2, R8, R13	3	ERJ-3GEYJ103V	Resistor	10k	0603
R3	1	RC0603JR-074K7L	Resistor	4.7k	0603
R9	1	ERJ-3GEYJ562V	Resistor	5.6k	0603
R10	1	ERJ-3GEYJ183V	Resistor	18k	0603
R12	1	ERJ-3GEYJ101V	Resistor	100	0603
R14, R15	2	ERJ-3GEYJ331V	Resistor	330	0603
R6, R7	2	ERJ-3GEYJ152V	Resistor	1.5k	0603
L1	1	MMZ2012S400A	Ferrite		0805

Notes

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46C and to discontinue any product or service per JESD48B. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information 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.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components which meet ISO/TS16949 requirements, mainly for automotive use. Components which have not been so designated are neither designed nor intended for automotive use; and TI will not be responsible for any failure of such components to meet such requirements.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Mobile Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

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