LED dimming implemented on STM32™ microcontroller

1 Introduction

This application note deals with the general principles of LED dimming. This note also focuses on the use of pulse width modulation (PWM) for digital dimming. Its implementation on the STM32™ microcontroller in combination with the STP24DP05 LED driver is described in detail and the most common design approaches are compared.

The conclusion helps developers choose the best approach for their LED dimming application.
Contents

1 Introduction ............................................. 1

2 Description of LED dimming ............................. 5
  2.1 Driving multiple LED applications .................. 6
  2.2 Multi-channel dimming using serial-parallel conversion .... 6
  2.3 Specific dimming application using an STP24DP05 ........... 8

3 Dimming application using an STM32 microcontroller .......... 9
  3.1 Data transfers and timing ............................ 9
    3.1.1 Dynamic data transfer ............................. 9
    3.1.2 Static data transfer using lookup tables .............. 12
    3.1.3 Single interrupt .................................. 13
    3.1.4 Single interrupt with DMA .......................... 16
    3.1.5 Interrupt + DMA + interrupt ........................ 17
  3.2 Data preparation: redundant and non redundant use of memory .... 19
    3.2.1 Full lookup table (redundant) ....................... 19
    3.2.2 Reduced lookup table (non redundant) ............... 20
  3.3 Timing: set up of the main dimming timer .................. 22
    3.3.1 The main system timer, SysTick setting ................ 22

4 Achieved values .......................................... 25
  4.1 Dimming approach comparison and conclusion .............. 29
    4.1.1 Comparison between interrupt and interrupt + DMA approaches .... 29
    4.1.2 Comparison between dynamic preparation, static, and reduced static lookup table .... 29

5 References documents ................................... 30

6 Revision history .......................................... 30
List of tables

Table 1. LEDmap - requested LED brightness configuration ........................................ 19
Table 2. Full lookup table ......................................................................................... 20
Table 3. LEDmap - requested LED brightness configuration ........................................ 21
Table 4. LEDmap - reordered requested LED brightness configuration ..................... 22
Table 5. Reduced lookup table .................................................................................. 22
Table 6. TotalTime necessary for requested dimming parameters. ............................ 23
Table 7. Maximum refresh rate .................................................................................. 23
Table 8. SysTick-reload-time for 100 Hz ................................................................ 23
Table 9. Example of values used in Equations 3 and 4 ............................................. 24
Table 10. Lookup table size (full and reduced) .......................................................... 25
Table 11. Refresh frequencies .................................................................................... 26
Table 12. Refresh frequencies for better optimized code (8 and 12-bit PWM) ............ 28
Table 13. Refresh frequencies for better optimized code (10 and 11-bit PWM) .......... 28
Table 14. Document revision history ........................................................................ 30
List of figures

Figure 1. PWM signal with 16 steps of possible voltage values .......................... 5
Figure 2. PWM serial-parallel conversion for multiple LED applications .................. 7
Figure 3. Example of a specific lighting setting .................................................. 7
Figure 4. Simplified setup of an application using an STP24DP05 .......................... 8
Figure 5. STM32 evaluation board setup ......................................................... 9
Figure 6. STM32 evaluation board block diagram ............................................. 9
Figure 7. Computation of LEDdriverDATAbuffer variable ............................... 10
Figure 8. Conversion of LEDmap array to LEDdriverDATAbuffer ...................... 11
Figure 9. Dynamic data preparation - time consumption - block diagram ............... 12
Figure 10. Dynamic data preparation - time consumption - real signals .................. 12
Figure 11. Static lookup table time consumption with 166 kHz - 333 kHz LE signal .... 13
Figure 12. Timing diagram: Single interrupt method ........................................... 14
Figure 13. Real timing signals for single interrupt method .................................... 14
Figure 14. Timing diagram: Single interrupt with DMA ....................................... 16
Figure 15. Timer interrupt + DMA transfer + DMA interrupt ............................... 17
Figure 16. Real signals: timer interrupt + DMA transfer + DMA interrupt model ....... 18
2 Description of LED dimming

There are two categories of LED dimming:

- **Analog**: Uses a linear device to set the current flow through the LED.
  
  Advantages: Absence of digital noise; a simple implementation.
  
  Limitations: Color shift, low dynamics.

- **Digital**: Uses a switch to set nominal or zero current flow through the LED.
  
  Advantages: Accurate and fast color control.
  
  Limitations: The presence of digital noise and necessity to use a frequency high enough to prevent flickering; a complex implementation.

This document focuses on digital dimming and the use of PWM (pulse width modulation) in common applications.

When using PWM for lighting applications, the following rules must be kept in mind:

- For the LED light to be seen as non-flickering by the human eye, in most cases the PWM frequency must be higher than 100 Hz.

- The LED light brightness depends on the PWM duty cycle. The brightness depends on the resolution of the PWM duty cycle.

**Figure 1.** PWM signal with 16 steps of possible voltage values

![PWM signal with 16 steps of possible voltage values](AM00341)

Using a 4-bit PWM resolution, there can be 16 different light intensity (energy) values.

Concerning color applications, 16 million (256 × 256 × 256) colors can be reached using tricolor LEDs with an 8-bit dimming resolution for each single color LED.
2.1 Driving multiple LED applications

Displays, indicators and panels increasingly use more and more LEDs. There are three approaches for driving higher numbers of LEDs:

1. **Dedicated pin**
   - Each LED is driven by dedicated system (microcontroller, FPGA) pin.
   - Advantages: Simple implementation of dimming algorithm. No conversion.
   - Disadvantages: Unusable for high LED counts. Consumes pins of microcontroller.

2. **Serial-parallel conversion**
   - Multiple LEDs are driven by a single system pin.
   - Advantages: Reduced number of system pins required. High LED count systems can be developed.
   - Disadvantages: Additional components and external buffers are needed. The data rate on a single pin must be \( n \) times greater than that for the single LED out of \( n \), where \( n \) is the number of LEDs.

3. **Matrix**
   - LEDs are arranged in a square matrix and driven by a complex and sophisticated algorithm that combines serial-parallel conversion with additional decomposition of the serial data stream into the rows and columns by another active component.

This document focuses on serial-parallel conversion applications.

2.2 Multi-channel dimming using serial-parallel conversion

The serial-parallel conversion approach is in fact a parallel-serial-parallel conversion since the data is stored in the system memory in parallel format and then is converted to serial format before its transfer over the serial interface. The serial stream is then reconverted to parallel format by the LED driver used to drive each LED. The serial data transfer speed must be \( n \) times faster than driving a single LED, where \( n \) is the number of parallel driver outputs.

Buffers (LED drivers) used for serial-parallel conversion come generally with the four most important signals:

- **SDI**: Serial data in
- **SDO**: Serial data out
- **CLK**: Clock signal used for loading data in the driver buffer via the SDI (input) and sending data from the buffer via the SDO (output). It also shifts all data actually present (previously loaded) in the driver.
- **LE**: Copies (latch enable) the actual values loaded in the driver into its output cells. The output cells drive parallel outputs.

The standard serial peripheral interface (SPI) contains SDO, SDI and CLK signals.

It is necessary to latch (LE signal pulse) internal driver data once per \( n \) CLK cycles, where \( n \) is the total LED count. This latching rule ensures the correct mapping of the serialized PWM data coming on SDI to the corresponding parallel output. Refer to **Figure 2**, **Figure 3** and **Figure 4**.
The following equation is used to determine how many LEDs can be used in an application with specific maximum SPI speed and PWM resolution requirements:

**Equation 1**

\[ N = \frac{f_{SPI}}{2^n f_{PWM}} \]

where:

- \( N \) is the maximum number of LEDs
- \( f_{SPI} \) is the SPI communication speed
- \( n \) is the PWM resolution
- \( f_{PWM} \) is the dimming frequency
The outcome of Equation 1 is valid only for the best-case scenario when the data are sent to the LED driver without any delay between blocks of data. This example, of course, is not a real one. The values for a real application can only get close to the value determined above.

2.3 **Specific dimming application using an STP24DP05**

*Figure 4* illustrates a simple dimming application using an STP24DP05 24-bit constant current LED sink driver with output error detection.

*Figure 4. Simplified setup of an application using an STP24DP05*
3 Dimming application using an STM32 microcontroller

This section describes a dimming application that uses an STM32 microcontroller (MCU). All the computations, results and tests were performed on an evaluation board (reference: STEVAL-ILL015V1) containing an STM32 MCU running at a clock frequency of 48 MHz.

For more information about the use of this evaluation board and dimming applications, please refer to user manuals UM0588 and UM0574 listed in Section 5: References documents.

Figure 5 and Figure 6 provide a brief overview of the STM32 evaluation board.

3.1 Data transfers and timing

This section describes the two approaches used to transfer data (dynamic and static) as well as timing considerations.

3.1.1 Dynamic data transfer

When using a dynamic data approach, all data is computed just before being transferred to the LED drivers.
The LED brightness value is defined by the user in the LEDmap array. The LEDmap is an array of 8-bit numbers (when using an 8-bit dimming resolution). Each value is dedicated to one single LED (Figure 7).

The dimming algorithm converts the LEDmap array into a serial stream. The serial stream in combination with the LE signal generates requested PWM outputs on the LED drivers as shown in Figure 8.

**Figure 7. Computation of LEDdriverDATAbuffer variable**
Figure 8. Conversion of LEDmap array to LEDdriverDATAbuffer

Example using a 4x8 LED matrix with an 8-bit PWM resolution

```c
while(1)
{
    for(depth = 1; depth <= 0xff; depth++)
    {
        for(DATACHAR = 0; DATACHAR < 4; DATACHAR++)
        {
            for(outputBIT = 0; outputBIT < 8; outputBIT++)
            {
                LEDdriverDATABuffer[DATACHAR] <<= 1;
                if(LEDmap[(DATACHAR*8) + outputBIT] > depth)
                    LEDdriverDATABuffer[DATACHAR] |= 0x0001;
                else
                    LEDdriverDATABuffer[DATACHAR] &= 0xFFFE;
            }
            SPI_SendData(SPI1, LEDdriverDATABuffer[DATACHAR]);
        }
        LE_LOW();
        LE_HIGH();
        LE_LOW();
    }
}
```

Even though the dynamic data preparation consumes a minimum of system memory, it takes a lot of computational time. This approach is only recommended for low LED count applications. A dynamic data preparation approach can reach 100 Hz for 32 LEDs (with maximum code optimization in C compiler on the STM32 evaluation board).
3.1.2 Static data transfer using lookup tables

When using static lookup tables, all data is computed before being sent to the LED drivers. Static lookup tables are used to convert the brightness value from an 8-, 9-, 10-, 11- or 12-bit representation from the LEDmap array into the serial stream representing the PWM of the LED driver outputs.
Once the brightness data is determined, the repetitive dimming procedure needs only to take the prepared data and send it to the LED driver. 

*Figure 11* shows the calculation time of the algorithm.

**Figure 11.** Static lookup table time consumption with 166 kHz - 333 kHz LE signal

Accurate LE timing and the dimming part running in the application background are necessary requirements for a successful application. These targets can be achieved using the following methods:

- Single interrupt
- Single interrupt with DMA
- Interrupt + DMA + interrupt

### 3.1.3 Single interrupt

The simplest approach is to use a timer interrupt approach where only a timer interrupt is required for the data transfer timing. This approach which does not require direct memory access (DMA) is designed for microcontrollers without DMA capability.

Except for pattern loading, SPI data transfer and Latch Enable generation, the code must also contain the TX buffer Empty Flag test. This test must be taken into account if you compute the total time necessary for all dimming operations.
**Figure 12** shows a timing diagram for a dimming application using this method.

**Figure 12. Timing diagram: Single interrupt method**

![Timing diagram: Single interrupt method](image)

**Figure 13** shows the waveforms of the signals output by the MCUs controlling the LED drivers.

**Figure 13. Real timing signals for single interrupt method**

![Real timing signals for single interrupt method](image)
Example of C code implementation of the timer interrupt routine

```c
void SysTickHandler(void)
{
    LE_LOW();  // Step 1
    LE_HIGH();
    depth+=4;
    if(depth > 0x400) // Step 2
    {
        depth = 0;
        SPI_SendData(SPI1, LookUp[depth+0]);  // Step 3
        while(!SPI_GetFlagStatus(SPI1, SPI_FLAG_TXE));
        SPI_SendData(SPI1, LookUp[depth+1]);
        while(!SPI_GetFlagStatus(SPI1, SPI_FLAG_TXE));
        SPI_SendData(SPI1, LookUp[depth+2]);
        while(!SPI_GetFlagStatus(SPI1, SPI_FLAG_TXE));
        SPI_SendData(SPI1, LookUp[depth+3]);
        while(!SPI_GetFlagStatus(SPI1, SPI_FLAG_TXE));
        LE_LOW();  // Step 5
        LE_HIGH();
    }
}
```

In the above example, the waveform and program code are described in the following steps:

1. The generation of the LE after entering the interrupt.
2. Increment of the lookup table pointer.
3. The transfer of the first 8 bits.
4. Test of TX buffer Empty Flag.
5. An additional edge on LE. This is for measurement purposes only.
6. The time between the end of the current interrupt and the next one. This time can be used for calculating certain foreground tasks in the application.
### 3.1.4 Single interrupt with DMA

The data is transferred from the microcontroller RAM into the LED driver using a single interrupt with DMA approach. You cannot use a DMA approach without a timer, because you cannot change the speed of the DMA and thus you cannot change PWM parameters and the refresh rate in required steps. The benefit of using DMA is that the SPI buffer can be automatically filled. The time saved is nearly in the range of the time needed for the DMA to restart and re-configure itself for the next transfer (in comparison to previous approach). An example of a timing diagram for this programming model is shown in Figure 14. The timing can be compared in Figure 12 and Figure 14.

**Figure 14. Timing diagram: Single interrupt with DMA**

- DMA start
- DMA end
- Interrupt start
- Interrupt end
- Pattern reading from RAM
- SPI is sending data
- Latch Enable generation
- DMA set and restart
3.1.5 Interrupt + DMA + interrupt

*Figure 14* shows that data from the previous transfer is latched during the current timer interrupt which causes several problems. The algorithm is more complicated in this case. A slightly different approach that has no impact on the system can be used: it is obtained by generating the LE pulse when the DMA transfer is complete instead of generating it on a timer interrupt. The LE pulse is generated at the DMA transfer complete interrupt as shown in *Figure 15*.

*Figure 15. Timer interrupt + DMA transfer + DMA interrupt*

*Figure 16* shows the waveform of the outbound signals from the microcontrollers controlling the LED drivers. An example of the C code implementation of the timer routine is given below. A detailed description follows the example.
Step "1" in Figure 16 is not necessary, but it is used for a better visualization of the tracking of signals on the oscilloscope.

Example of C code of DMA and timer interrupt routine: interrupt + DMA + interrupt approach

```c
void SysTickHandler(void) {
    LE_LOW(); (Step 1)
    LE_HIGH();

    depth+=4;
    if(depth > 0x400)
        depth = 0;

    DMA_Channel3->CCR &= CCR_ENABLE_Reset;
    DMA_Channel3->CNDTR = 0x04;
    DMA_Channel3->CMAR = ((u32) (LookUp))+depth;
    DMA_Channel3->CCR |= CCR.Enable_Set; (Step 2)
    DMA_Channel3->CCR |= DMA.IT_TC;

    LE_LOW();
}

void DMAChannel3_IRQHandler(void) (Step 5) {
    while(!SPI_GetFlagStatus(SPI1, SPI_FLAG_TXE)); (Step 6)
    LE_HIGH(); (Step 7)
    LE_LOW();
    DMA_ClearITPendingBit(DMA.IT_GL3);
}
```
In the above example, the waveform and program code are described in the following steps:

1. The generation of the LE after entering the interrupt.
2. Setting the DMA: source address, data length, and Start flag.
3. Data is sent automatically by DMA.
4. Data is sent automatically by DMA.
5. DMA interrupt routine is called when the last byte is sent by DMA.
6. Wait until the data from the SPI TX buffer is delivered to the driver.
   (Step 6 can be exchanged with calculations that take the amount of time necessary for the SPI transfer to be finished).
7. LE generation.

### 3.2 Data preparation: redundant and non redundant use of memory

#### 3.2.1 Full lookup table (redundant)

The static lookup table generation is mentioned in Section 3.1.2. The lookup table generation procedure is described in more detail in this section.

The lookup table row is computed for every PWM step (8-bit PWM means 256 rows for 256 steps). The C code of the full lookup table generation is given in the example below.

**Example using a 4×8 LED matrix with an 8-bit PWM resolution**

```c
void generateLookUpTable(unsigned char *LEDmapIN){

unsigned int tmpdepth;

for(tmpdepth = 0; tmpdepth < 0xff; tmpdepth++){
    for(DATAchar = 0; DATAchar < 4; DATAchar++){
        for(outputBIT = 0; outputBIT < 8; outputBIT++){
            LookUp[tmpdepth*4 + DATAchar] <<= 1;
            if(LEDmapIN[(DATAchar*8) + outputBIT] > tmpdepth)
                LookUp[tmpdepth*4 + DATAchar] |= 0x01;
            else
                LookUp[tmpdepth*4 + DATAchar] &= 0xFE;
        }
    }
}
```

<table>
<thead>
<tr>
<th>Table 1.</th>
<th>LEDmap - requested LED brightness configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output</strong></td>
<td><strong>LED 1</strong></td>
</tr>
<tr>
<td>Duty cycle (brightness)</td>
<td>$2 / 2^n$</td>
</tr>
</tbody>
</table>
3.2.2 Reduced lookup table (non redundant)

Only the data that changes the LED driver output state is generated. A LED can change the state only once per complete PWM cycle: from one to zero. If there are \( i \) number of LEDs, there are \( i \) new data items for the LED drivers. For example, if \( i = 32 \) LEDs when using an 8-bit PWM, you need a cycle with \( i \times i = 32 \times 32 = 1024 \) loops.

The inside of the loop is the same for full lookup table generation as described in Section 3.2.1. The lookup table row is computed only in case you need to change the LED intensity pattern (specific value of a parallel output configuration). Table 3, Table 4 and Table 5 bring better understanding of the lookup table reduction. The C code of the reduced lookup table generation is shown in the example below.

<table>
<thead>
<tr>
<th>Step</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LED 1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>( 2^n )</td>
<td>0</td>
</tr>
</tbody>
</table>
Example of reduced lookup table generation

```c
void performGenerateReducedLookUpTable(unsigned char *LEDmapIN,
int LEDcount, int PWMdepth, unsigned char **tableIN, unsigned
char **tableKeyIN, int *table_size){

  int streamSize = LEDcount/8;
  *table_size = differentValuesCount(LEDmapIN, LEDcount);
  *tableKeyIN = SortAndReduce(LEDmapIN, LEDcount, *table_size);

  *tableIN = malloc(streamSize * (*table_size)
               * sizeof(unsigned char));
  for(tmpdepth = 0; tmpdepth < (*table_size); tmpdepth++){
    for(DATAchar = 0; DATAchar < streamSize; DATAchar++){
      for(outputBIT = 0; outputBIT < 8; outputBIT++){
        (*tableIN)[tmpdepth*streamSize + DATAchar] <<= 1;
        if(LEDmapIN[(DATAchar*8) + outputBIT] >
           (*tableKeyIN)[tmpdepth])
          (*tableIN)[tmpdepth*streamSize + DATAchar] |= 0x01;
        else
          (*tableIN)[tmpdepth*streamSize + DATAchar] &= 0xFE;
      }
    }
  }

  // Table 3.
  // LEDmap - requested LED brightness configuration

  // Outputs | LED 1 | LED 2 | LED 3 | ... | ... | LED i
  // Duty cycle (brightness) | 2 / 2^n | 6 / 2^n | 4 / 2^n | 100% |
```

Table 3. **LEDmap** - requested LED brightness configuration
Timing: set up of the main dimming timer

3.3.1 The main system timer, SysTick setting

The approach with interrupt + DMA + interrupt is considered the most effective.

The microcontroller can reach the maximum frame rate based on the minimum time required for all the necessary operations. A time reserve of 6% was used:

- \( t_{DMAen} \) re-enabling the DMA (~ 1.25 µs)
- \( t_{oneSPI} \) time necessary for transfer of the data over SPI (~ LEDs × 55 ns)
- \( t_{LEgen} \) LE generation and new data pointer generation (~ 1 µs)

The total time needed for correct processing is the sum of the times above:

**Equation 2**

\[
TotalTime = t_{DMAen} + t_{oneSPI} \times LEDs + t_{LEgen}
\]

\[
= 1.25 \mu s + 55 \text{ ns} \times \text{LEDs} + 1 \mu s
\]

\[
= 2.25 \mu s + 55 \text{ ns} \times \text{LEDs}
\]

The number of LEDs must be rounded up to the nearest number commensurable by the length of an SPI buffer. The SPI buffer length is 8 either or 16 bits in most cases.
More examples of TotalTime for different number of LEDs and PWM resolution can be seen in Table 6.

### Table 6. TotalTime necessary for requested dimming parameters

<table>
<thead>
<tr>
<th>LEDs</th>
<th>PWM</th>
<th>TotalTime</th>
<th>Max. LE frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8-bit</td>
<td>2.25 + 0.055 × 8 = 2.69 µs</td>
<td>371.7 kHz</td>
</tr>
<tr>
<td>8</td>
<td>12-bit</td>
<td>2.25 + 0.055 × 8 = 2.69 µs</td>
<td>371.7 kHz</td>
</tr>
<tr>
<td>672</td>
<td>8-bit</td>
<td>2.25 + 0.055 × 672 = 39.21 µs</td>
<td>25.5 kHz</td>
</tr>
</tbody>
</table>

*Table 6* shows the time required for processing one PWM step, one buffer load and one buffer latch. With an 8-bit PWM resolution, 256 buffer latches are needed; the maximum LE (Latch Enable) frequency must be divided by 256. These results are shown in *Table 7*.

### Table 7. Maximum refresh rate

<table>
<thead>
<tr>
<th>LEDs</th>
<th>PWM</th>
<th>Max. LE frequency</th>
<th>Max. refresh rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8-bit</td>
<td>371.7 kHz</td>
<td>1452 Hz</td>
</tr>
<tr>
<td>8</td>
<td>12-bit</td>
<td>371.7 kHz</td>
<td>90.7 Hz</td>
</tr>
<tr>
<td>672</td>
<td>8-bit</td>
<td>25.5 kHz</td>
<td>99.6 Hz</td>
</tr>
</tbody>
</table>

*Table 7* shows the maximum refresh frequencies for controlled LEDs. But the inverse computation must be used to solve our task. *Table 8* shows the inversion of the *Table 7* values and gives the required refresh frequency. The LE frequency must be derived and the SysTick timer preload value (SysTick-preload-value) set to generate the LE pulse by generating an interrupt at every counter reload event.

### Table 8. SysTick-reload-time for 100 Hz

<table>
<thead>
<tr>
<th>LEDs</th>
<th>PWM</th>
<th>Requested refresh rate</th>
<th>SysTick-reload-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8-bit</td>
<td>100 Hz</td>
<td>39 µs (25.6 kHz)</td>
</tr>
<tr>
<td>8</td>
<td>12-bit</td>
<td>90.7 Hz</td>
<td>2.7 µs (371.7 kHz)</td>
</tr>
<tr>
<td>672</td>
<td>8-bit</td>
<td>100 Hz</td>
<td>39 µs (25.6 kHz)</td>
</tr>
<tr>
<td>32</td>
<td>8-bit</td>
<td>100 Hz</td>
<td>39 µs (25.6 kHz)</td>
</tr>
</tbody>
</table>

*Table 7* and *Table 8* show that SysTick-reload-frequency will be set between 25.6 kHz and 371.7 kHz according to requested PWM resolution and LED count.

The STM32 SysTick timer uses a reference clock of 72 MHz when set to the maximum value. The interrupt period for this timer is the SysTick-preload-value multiplied by 1 / (72 000 000).

The reference clock of STM32 evaluation board is set to 48 MHz for the USB to operate correctly. This value should be taken into the account while making measurements on this evaluation board as their computation will influence *Equation 3*.

The reference clock can be also divided by 8 by using the hardware divider. The divided reference clock is then 9 MHz. In this case, the interrupt period for this timer is equal to a preload value multiplied by 1 / (9 000 000).

\[
\text{Preload value} = \frac{\text{requested SysTick interrupt time}}{\text{reference period}}
\]

\[
\text{Preload value} = \frac{\text{reference frequency}}{\text{requested SysTick interrupt frequency}}
\]
Equation 3

\[
\text{SysTick-preload-value} = \frac{72\,000\,000}{25\,600} = 2812
\]

The frequency ranges 25.6 kHz and 371.7 kHz required for all possible applications ranges can be achieved using preload values: 2812 and 194, or with divider by 8: 351 or 24.

SysTick timer setting with no divider on the reference clock:

\[
\begin{align*}
&\text{SysTick\_CLKSourceConfig(SysTick\_CLKSource\_HCLK);} \\
&\text{SysTick\_SetReload(900);}
\end{align*}
\]

generates pulses with the period of 12 µs.

SysTick timer setting with the divider by 8 on the reference clock:

\[
\begin{align*}
&\text{SysTick\_CLKSourceConfig(SysTick\_CLKSource\_HCLK\_Div8);} \\
&\text{SysTick\_SetReload(900);}
\end{align*}
\]

generates pulses with the period of 100 µs.

The setup algorithm for the SysTick timer configuration has to check if it is possible to set preload value to keep enough time for processing (TotalTime) together with reaching the requested refresh frequency:

\[
\begin{align*}
\text{TotalTime} &= 2.25 + 0.055 \times \text{LEDs} \\
\text{SysTick-reload-time} &= \frac{1}{\text{SysTick-reload-frequency}}
\end{align*}
\]

TotalTime is the time slot, during which the SPI transfer and all the arbitrations needs to be achieved. The SysTick time is the time slot which must be respected to reach the required refresh rate.

Equation 4

\[
\text{SysTick-reload-time} > \text{TotalTime}
\]

To reach the requested refresh rate and set up the SysTick timer preload value:

\[
\text{SysTick-reload-frequency} = \text{requested-refresh-rate} \times \text{PWMdepth} (100 \, \text{Hz} \times 256 \, \text{steps}).
\]
4 Achieved values

Dynamic computing takes a lot of time and 120 Hz can be reached only for 32 LEDs. In addition, no other foreground or background tasks can be performed during computation. As a consequence, this solution is recommended only for simple applications that use only a few diodes. For other more complicated applications, it is recommended to use the lookup table approach using a single interrupt only or interrupt + DMA (if DMA is available).

The LED brightness resolution is limited by the size of the lookup table (Table 10). For every level of brightness, you need a single string of zeros and ones for all the LEDs.

Table 10. Lookup table size (full and reduced)

<table>
<thead>
<tr>
<th>Bit depth</th>
<th>LEDs</th>
<th>Lookup table size: $i \times 2^n$</th>
<th>Reduced table size$^{(1)}$: $i \times n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>32</td>
<td>$256 \times 4 = 1\ \text{kB}$</td>
<td>$32 \times 4 = 128\ \text{B}$</td>
</tr>
<tr>
<td>12</td>
<td>32</td>
<td>$4096 \times 4 = 16\ \text{kB}$</td>
<td>$32 \times 4 = 128\ \text{B}$</td>
</tr>
<tr>
<td>8</td>
<td>128</td>
<td>$256 \times 16 = 4\ \text{kB}$</td>
<td>$128 \times 16 = 2\ \text{kB}$</td>
</tr>
<tr>
<td>12</td>
<td>128</td>
<td>$4096 \times 16 = 64\ \text{kB}$</td>
<td>$128 \times 16 = 2\ \text{kB}$</td>
</tr>
<tr>
<td>8</td>
<td>512</td>
<td>$256 \times 64 = 16\ \text{kB}$</td>
<td>$256 \times 64 = 16\ \text{kB}$</td>
</tr>
<tr>
<td>12</td>
<td>512</td>
<td>$4096 \times 64 = 256\ \text{kB}$</td>
<td>$512 \times 64 = 32\ \text{kB}$</td>
</tr>
</tbody>
</table>

1. Values indicates the maximum size.

From Figure 13, the maximum refresh frequency can be computed. The time between two interrupts is 6 µs. This is a refresh rate for one brightness level, for 32 LEDs, 166 kHz. Table 11. shows the possible refresh rates for whole brightness depth and different LED count.
Table 11. Refresh frequencies

<table>
<thead>
<tr>
<th>LEDs/PWM</th>
<th>8-bit [Hz]</th>
<th>9-bit [Hz]</th>
<th>10-bit [Hz]</th>
<th>11-bit [Hz]</th>
<th>12-bit [Hz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1514</td>
<td>757</td>
<td>379</td>
<td>189</td>
<td>95² (1)</td>
</tr>
<tr>
<td>16</td>
<td>1293</td>
<td>647</td>
<td>323</td>
<td>162</td>
<td>81</td>
</tr>
<tr>
<td>24</td>
<td>1129</td>
<td>564</td>
<td>282</td>
<td>141</td>
<td>71</td>
</tr>
<tr>
<td>32</td>
<td>1002</td>
<td>501</td>
<td>250</td>
<td>125</td>
<td>63</td>
</tr>
<tr>
<td>40</td>
<td>900</td>
<td>450</td>
<td>225</td>
<td>113</td>
<td>56</td>
</tr>
<tr>
<td>48</td>
<td>817</td>
<td>409</td>
<td>204</td>
<td>102</td>
<td>51</td>
</tr>
<tr>
<td>56</td>
<td>748</td>
<td>374</td>
<td>187</td>
<td>94</td>
<td>47</td>
</tr>
<tr>
<td>64</td>
<td>690</td>
<td>345</td>
<td>173</td>
<td>86</td>
<td>43</td>
</tr>
<tr>
<td>72</td>
<td>640</td>
<td>320</td>
<td>160</td>
<td>80</td>
<td>40</td>
</tr>
<tr>
<td>80</td>
<td>597</td>
<td>299</td>
<td>149</td>
<td>75</td>
<td>37</td>
</tr>
<tr>
<td>88</td>
<td>560</td>
<td>280</td>
<td>140</td>
<td>70</td>
<td>35</td>
</tr>
<tr>
<td>96</td>
<td>526</td>
<td>263</td>
<td>132</td>
<td>66</td>
<td>33</td>
</tr>
<tr>
<td>104</td>
<td>497</td>
<td>248</td>
<td>124</td>
<td>62</td>
<td>31</td>
</tr>
<tr>
<td>112</td>
<td>471</td>
<td>235</td>
<td>118</td>
<td>59</td>
<td>29</td>
</tr>
<tr>
<td>120</td>
<td>447</td>
<td>223</td>
<td>112</td>
<td>56</td>
<td>28</td>
</tr>
<tr>
<td>128</td>
<td>426</td>
<td>213</td>
<td>106</td>
<td>53</td>
<td>27</td>
</tr>
<tr>
<td>136</td>
<td>406</td>
<td>203</td>
<td>102</td>
<td>51</td>
<td>25</td>
</tr>
<tr>
<td>144</td>
<td>388</td>
<td>194</td>
<td>97</td>
<td>49</td>
<td>24</td>
</tr>
<tr>
<td>152</td>
<td>372</td>
<td>186</td>
<td>93</td>
<td>47</td>
<td>23</td>
</tr>
<tr>
<td>160</td>
<td>357</td>
<td>179</td>
<td>89</td>
<td>45</td>
<td>22</td>
</tr>
<tr>
<td>168</td>
<td>343</td>
<td>172</td>
<td>86</td>
<td>43</td>
<td>21</td>
</tr>
<tr>
<td>176</td>
<td>330</td>
<td>165</td>
<td>83</td>
<td>41</td>
<td>21</td>
</tr>
<tr>
<td>184</td>
<td>319</td>
<td>159</td>
<td>80</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>192</td>
<td>308</td>
<td>154</td>
<td>77</td>
<td>38</td>
<td>19</td>
</tr>
<tr>
<td>7200</td>
<td>297</td>
<td>149</td>
<td>74</td>
<td>37</td>
<td>19</td>
</tr>
<tr>
<td>208</td>
<td>288</td>
<td>144</td>
<td>72</td>
<td>36</td>
<td>18</td>
</tr>
<tr>
<td>216</td>
<td>279</td>
<td>139</td>
<td>70</td>
<td>35</td>
<td>17</td>
</tr>
<tr>
<td>224</td>
<td>270</td>
<td>135</td>
<td>68</td>
<td>34</td>
<td>17</td>
</tr>
<tr>
<td>232</td>
<td>262</td>
<td>131</td>
<td>66</td>
<td>33</td>
<td>16</td>
</tr>
<tr>
<td>240</td>
<td>255</td>
<td>127</td>
<td>64</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>248</td>
<td>248</td>
<td>124</td>
<td>62</td>
<td>31</td>
<td>15</td>
</tr>
<tr>
<td>256</td>
<td>241</td>
<td>120</td>
<td>60</td>
<td>30</td>
<td>15</td>
</tr>
<tr>
<td>264</td>
<td>234</td>
<td>117</td>
<td>59</td>
<td>29</td>
<td>15</td>
</tr>
</tbody>
</table>
Table 11. Refresh frequencies (continued)

<table>
<thead>
<tr>
<th>LEDs/PWM</th>
<th>8-bit [Hz]</th>
<th>9-bit [Hz]</th>
<th>10-bit [Hz]</th>
<th>11-bit [Hz]</th>
<th>12-bit [Hz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>272</td>
<td>228</td>
<td>114</td>
<td>57</td>
<td>29</td>
<td>14</td>
</tr>
<tr>
<td>280</td>
<td>223</td>
<td>111</td>
<td>56</td>
<td>28</td>
<td>14</td>
</tr>
<tr>
<td>288</td>
<td>217</td>
<td>109</td>
<td>54</td>
<td>27</td>
<td>14</td>
</tr>
<tr>
<td>296</td>
<td>212</td>
<td>106</td>
<td>53</td>
<td>27</td>
<td>13</td>
</tr>
<tr>
<td>304</td>
<td>207</td>
<td>104</td>
<td>52</td>
<td>26</td>
<td>13</td>
</tr>
<tr>
<td>312</td>
<td>202</td>
<td>101</td>
<td>51</td>
<td>25</td>
<td>13</td>
</tr>
<tr>
<td>320</td>
<td>198</td>
<td>99</td>
<td>49</td>
<td>25</td>
<td>12</td>
</tr>
<tr>
<td>328</td>
<td>194</td>
<td>97</td>
<td>48</td>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>336</td>
<td>189</td>
<td>95</td>
<td>47</td>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>344</td>
<td>185</td>
<td>93</td>
<td>46</td>
<td>23</td>
<td>12</td>
</tr>
<tr>
<td>352</td>
<td>182</td>
<td>91</td>
<td>45</td>
<td>23</td>
<td>11</td>
</tr>
<tr>
<td>360</td>
<td>178</td>
<td>89</td>
<td>45</td>
<td>22</td>
<td>11</td>
</tr>
<tr>
<td>368</td>
<td>175</td>
<td>87</td>
<td>44</td>
<td>22</td>
<td>11</td>
</tr>
<tr>
<td>376</td>
<td>171</td>
<td>86</td>
<td>43</td>
<td>21</td>
<td>11</td>
</tr>
<tr>
<td>384</td>
<td>168</td>
<td>84</td>
<td>42</td>
<td>21</td>
<td>10</td>
</tr>
<tr>
<td>392</td>
<td>165</td>
<td>82</td>
<td>41</td>
<td>21</td>
<td>10</td>
</tr>
<tr>
<td>400</td>
<td>162</td>
<td>81</td>
<td>40</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>408</td>
<td>159</td>
<td>79</td>
<td>40</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>416</td>
<td>156</td>
<td>78</td>
<td>39</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>424</td>
<td>153</td>
<td>77</td>
<td>38</td>
<td>19</td>
<td>10</td>
</tr>
<tr>
<td>432</td>
<td>151</td>
<td>75</td>
<td>38</td>
<td>19</td>
<td>9</td>
</tr>
<tr>
<td>440</td>
<td>148</td>
<td>74</td>
<td>37</td>
<td>19</td>
<td>9</td>
</tr>
<tr>
<td>448</td>
<td>146</td>
<td>73</td>
<td>36</td>
<td>18</td>
<td>9</td>
</tr>
<tr>
<td>456</td>
<td>144</td>
<td>72</td>
<td>36</td>
<td>18</td>
<td>9</td>
</tr>
<tr>
<td>464</td>
<td>141</td>
<td>71</td>
<td>35</td>
<td>18</td>
<td>9</td>
</tr>
<tr>
<td>472</td>
<td>139</td>
<td>70</td>
<td>35</td>
<td>17</td>
<td>9</td>
</tr>
<tr>
<td>480</td>
<td>137</td>
<td>68</td>
<td>34</td>
<td>17</td>
<td>9</td>
</tr>
<tr>
<td>488</td>
<td>135</td>
<td>67</td>
<td>34</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td>496</td>
<td>133</td>
<td>66</td>
<td>33</td>
<td>17</td>
<td>8</td>
</tr>
<tr>
<td>504</td>
<td>131</td>
<td>65</td>
<td>33</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>512</td>
<td>129</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>520</td>
<td>127</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>528</td>
<td>125</td>
<td>63</td>
<td>31</td>
<td>16</td>
<td>8</td>
</tr>
</tbody>
</table>
### Table 11. Refresh frequencies (continued)

<table>
<thead>
<tr>
<th>LEDs/PWM</th>
<th>8-bit [Hz]</th>
<th>9-bit [Hz]</th>
<th>10-bit [Hz]</th>
<th>11-bit [Hz]</th>
<th>12-bit [Hz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>536</td>
<td>124</td>
<td>62</td>
<td>31</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>544</td>
<td>122</td>
<td>61</td>
<td>30</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>552</td>
<td>120</td>
<td>60</td>
<td>30</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>560</td>
<td>119</td>
<td>59</td>
<td>30</td>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>568</td>
<td>117</td>
<td>59</td>
<td>29</td>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>576</td>
<td>116</td>
<td>58</td>
<td>29</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>584</td>
<td>114</td>
<td>57</td>
<td>29</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>592</td>
<td>113</td>
<td>56</td>
<td>28</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>600</td>
<td>111</td>
<td>56</td>
<td>28</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>608</td>
<td>110</td>
<td>55</td>
<td>27</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>616</td>
<td>110</td>
<td>55</td>
<td>27</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>624</td>
<td>110</td>
<td>55</td>
<td>27</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>632</td>
<td>106</td>
<td>53</td>
<td>26</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>640</td>
<td>105</td>
<td>52</td>
<td>26</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>648</td>
<td>103</td>
<td>52</td>
<td>26</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>656</td>
<td>102</td>
<td>51</td>
<td>26</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>664</td>
<td>101</td>
<td>51</td>
<td>25</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>672</td>
<td>100</td>
<td>50</td>
<td>25</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>680</td>
<td>99</td>
<td>49</td>
<td>25</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>688</td>
<td>98</td>
<td>49</td>
<td>24</td>
<td>12</td>
<td>6</td>
</tr>
</tbody>
</table>

1. 100 Hz can be reached with very high level of optimization.

### Table 12. Refresh frequencies for better optimized code (8- and 12-bit PWM)

<table>
<thead>
<tr>
<th>LEDs</th>
<th>8-bit PWM freq. [Hz]</th>
<th>12-bit PWM freq. [Hz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>250 000 / 256 = 1000</td>
<td>250 000 / 4096 = 61</td>
</tr>
<tr>
<td>128</td>
<td>(250 000 / 256) / 4 = 244</td>
<td>(250 000 / 4096) / 4 = 15</td>
</tr>
<tr>
<td>512</td>
<td>(250 000 / 256) / 16 = 61</td>
<td>(250 000 / 4096) / 16 = 4</td>
</tr>
</tbody>
</table>

### Table 13. Refresh frequencies for better optimized code (10- and 11-bit PWM)

<table>
<thead>
<tr>
<th>LEDs</th>
<th>10-bit PWM freq. [Hz]</th>
<th>11-bit PWM freq. [Hz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>250 000 / 1024 = 244</td>
<td>250 000 / 2048 = 122</td>
</tr>
<tr>
<td>128</td>
<td>(250 000 / 1024) / 4 = 64</td>
<td>(250 000 / 2048) / 4 = 30</td>
</tr>
<tr>
<td>512</td>
<td>(250 000 / 1024) / 16 = 15</td>
<td>(250 000 / 2048) / 16 = 7</td>
</tr>
</tbody>
</table>
Table 12 and Table 13 show the results in case you remove the additional LE edge used for the measurement of the interrupt end and optimize time between two interrupts (used for SPI in 16-bit mode).

4.1 Dimming approach comparison and conclusion

4.1.1 Comparison between interrupt and interrupt + DMA approaches

When using the single interrupt approach described in Figure 12 and Figure 13, the time spent by the MCU to drive the LEDs is long and little time is left to execute other tasks.

The combination of the single interrupt + DMA approach frees more computing time for other tasks. This approach does not improve the refresh rate.

4.1.2 Comparison between dynamic preparation, static, and reduced static lookup table

The dynamic data preparation shown in Figure 9 allows to save memory but requires additional computing time.

The static lookup table method saves computing time. However, the computing must be performed at the beginning of the dimming.

The reduced static lookup table offers the same functionalities as a full lookup table, while saving memory space.
5 References documents

1. User manual UM0588 - Multilayer C library for LED dimming used on systems with SPI and DMA capabilities

6 Revision history

Table 14. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-Nov-2008</td>
<td>1</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>