Think Silicon

AN APPLIED MATERIALS COMPANY

NEMA®| dc MiP Panels Configuration

Application Note

Version v23.10

Part Number: D-APN-MIP

December 3, 2023

Disclaimer

This document is written in good faith with the intent to assist the readers in the use of the product. Circuit diagrams and other information relating to Think Silicon S.A products are included as a means of illustrating typical applications. Although the information has been checked and is believed to be accurate, no responsibility is assumed for inaccuracies. Information contained in this document is subject to continuous improvements and developments.

Think Silicon S.A products are not designed, intended, authorized or warranted for use in any life support or other application where product failure could cause or contribute to personal injury or severe property damage. Any and all such uses without prior written approval of Think Silicon S.A. will be fully at the risk of the customer.

Think Silicon S.A. disclaims and excludes any and all warranties, including without limitation any and all implied warranties of merchantability, fitness for a particular purpose, title, and infringement and the like, and any and all warranties arising from any course or dealing or usage of trade.

This document may not be copied, reproduced, or transmitted to others in any manner. Nor may any use of information in this document be made, except for the specific purposes for which it is transmitted to the recipient, without the prior written consent of Think Silicon S.A. This specification is subject to change at anytime without notice.

Think Silicon S.A. is not responsible for any errors contained herein. In no event shall Think Silicon S.A. be liable for any direct, indirect, incidental, special, punitive, or consequential damages; or for loss of data, profits, savings or revenues of any kind; regardless of the form of action, whether based on contract; tort; negligence of Think Silicon S.A or others; strict liability; breach of warranty; or otherwise; whether or not any remedy of buyers is held to have failed of its essential purpose, and whether or not Think Silicon S.A. has been advised of the possibility of such damages.

Questions or comments may be directed to:
Think Silicon S.A
Suite B8
Patras Science Park
Rion Achaias 26504, Greece
web: http://www.think-silicon.com
email:info@think-silicon.com
Tel: +30 2610 911543
Fax: +30 2610 911544

Contents

1 Overview

This application note aims to provide guidance in the configuration of NEMA®| dc, so that it can work with MiP (Memory in Pixel) panels that makes use of the parallel interface.

Memory in Pixel panels (mostly JDI and Sharp panels), require the display controller to drive display timing signals, where the relation of timings is quite strict.

Since some MiP Sharp panels bring the same interface, Table 1 captures the differences between JD panels and Sharp panels regarding the naming of signals:

Table 1: JDI to MIP signal name correspondence

Japan Display Inc	Sharp	Description
XRST	INTB	Reset signal for the horizontal and vertical driver
VST	GSP	Start signal for the vertical driver
VCK	GCK	Shift clock for the vertical driver
ENB	GEN	Write enable signal for the pixel memory
HST	BSP	Start signal for the horizontal driver
HCK	BCK	Shift clock for the horizontal driver
R1	R[0]	Red image data (odd pixels)
R2	R[1]	Red image data (even pixels)
G1	G[0]	Green image data (odd pixels)
G2	G[1]	Green image data (even pixels)
B1	B[0]	Blue image data (odd pixels)
B2	B[1]	Blue image data (even pixels)

To program NEMA®| dc to drive the MiP panel, you have to go through the timing diagrams of the panel documentation and export all the necessary timing information. Programming of NEMA®| dc must go from the configuration of input `pll_clk` to its software configuration.

This guide makes use of the Sharp LS014B7DD01 display manual as a use case. All values that were extracted are the typical values, where Min and Max were taken into consideration, in case there is no typical value.

2 Configure `format_clk` frequency

As a first step, define the `format_clk` frequency. To meet the typical times of the manual of the display, set the `format_clk` frequency to match 4-times of the BCK frequency (or half of the high level width or low level width of the BCK)

Attention:

The `pixel_clk`/`format_clk` ratio should be greater than 2:1. It can be any ratio with `pixel_clk` greater than format_clk and is irrelevant of the color format of the layer.[1]

Due to the architecture of the interface and NEMA®| dc pipeline, the faster the `pixel_clk` is, the faster GCK (or VCK) transition, in case of fast forward feature (partial update).

Table 2: BCK values

BCK		**Typical**
fBCK	BCK frequency	0.746 MHz
thwBCK	High Level Width	670 ns
tlwBCK	Low Level Width	670 ns

`format_clk` input frequency = `fBCK` * 4 = 2.984 MHz

`format_clk` input period = 335 ns

[1] Each layer fetches data from memory through the DMA, which is running with HCLK frequency. In this case, you have to ensure that the throughput of the layer is able to provide sufficient pixel data to the internal pipeline of NEMA®| dc, which consumes 1 pixel/pixel_clk.

3 Setting Panel Timing configuration

Since the MiP parallel interface panel requires specific timings, we must extract and pass the necessary information to NEMA®| dc.

For this interface, a struct is introduced that holds the configuration of the panel and consists of the following variables:

```
typedef struct __MiP_display_config_t {
    int resx;                    /** Panel Horizontal Resolution */
    int resy;                    /** Panel Vertical Resolution */
    int XRST_INTB_delay ;        /** Delay inserted prior of XRST or INTB in multiples
                                     of format_clk */
    int XRST_INTB_width ;        /** Width of High state of XRST or INTB in multiples
                                     of format_clk */
    int VST_GSP_delay ;          /** Delay inserted prior of VST or GSP in multiples
                                     of format_clk */
    int VST_GSP_width ;          /** Width of High state of VST or GSP in multiples
                                     of format_clk */
    int VCK_GCK_delay ;          /** Delay inserted prior of VCK or GCK in multiples
                                     of format_clk */
    int VCK_GCK_width ;          /** Width of High state of VCK or GCK in multiples
                                     of format_clk */
    int VCK_GCK_closing_pulses ; /** Number of VCK or GCK pulses without ENB or GEN
                                     signal at the end of frame */
    int HST_BSP_delay ;          /** Delay inserted prior of HST or BSP in multiples
                                     of format_clk */
    int HST_BSP_width ;          /** Width of High state of HST or BSP in multiples
                                     of format_clk */
    int HCK_BCK_data_start ;     /** The HCK or BCK cycle the pixel data should start
                                     at */
    int ENB_GEN_delay ;          /** Delay inserted prior of ENB or GEN in multiples
                                     of format_clk */
    int ENB_GEN_width ;          /** Width of High state of ENB or GEN in multiples
                                     of format_clk */
} MiP_display_config_t;
```

As a first step, you must define the values for Horizontal and Vertical Resolution of the panel. These values also define the NEMA®| dc frame resolution.

Table 3: Pixel Values from LS014B7DD01 manual

	Pixels	**Values defined on software**
Horizontal Pixel Number	280	resx = 280
Vertical Pixel Number	280	resy = 280

The next step is to define the INTB assertion point and high width. Since INTB is the reset signal, you can have it as the base of the vertical timing's setup. In that case, `XRST_INTB_delay` can be set to 1. A zero value is not acceptable.

Also INTB high width (`thwINTB`) can be configured to match the number of GCK pulse at the falling edge of INTB. From the vertical signal timing diagram (upcoming figure), we can see that GCK#566 is the pulse where INTB falling edge happens. `XRST_INTB_width` in this case must be configured with value 566.

```
XRST_INTB_delay = 1
XRST_INTB_width = 566
```

3.1 Vertical Signal Timing Configuration

The next task that needs to take place is the configuration of the Vertical Signal Timing.

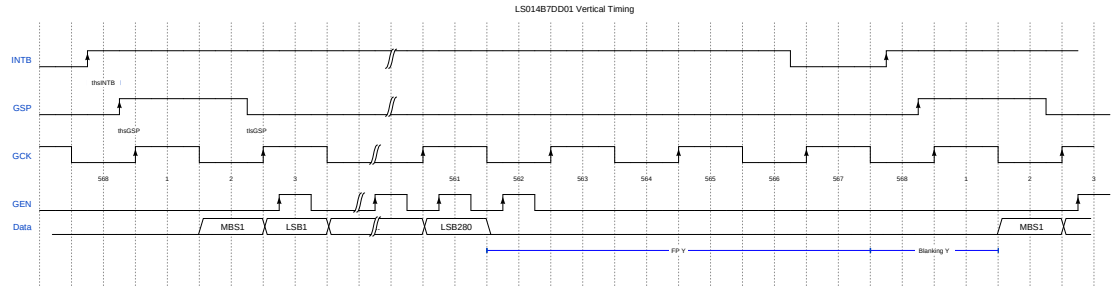

Figure 1: LS014B7DD01 Vertical Timing

As a first step you must configure GSP assertion point and high width.

Table 4: Timings from LS014B7DD01 Manual

		Time (ns)		
		Min	**Typical**	**Max**
thsINTB	Timing from Rising INTB till Rising of GSP	23760	24120	24480
thsGSP	GSP set-up time high level	47520	48240	48960
tlsGSP	GSP set-up time low level	47520	48240	48960

Attention: Zero positions of INTB and GSP match.

In the case of GSP assertion point configuration, we must set the correct value to `VST_GSP_delay`. From the above mentioned table, the typical value of `thsINTB` is

24120ns. The value to be set to VST_GSP_delay is computed from the following expression:

```
VST_GSP_delay = thsINTB/format_clk_period + XRST_INTB_delay = (24120/335) + 1
VST_GSP_delay = 72 + XRST_INTB_delay
```

For the GSP high width, we have to calculate its value since the manual describes only the setup timings in respect to GCK. By combining thsGSP & tlsGSP & High Level Width of GCK, you have the GSP width that you must set to VST_GSP_width.

```
VST_GSP_width = (thsGSP + thwGCK + tlwGCK - tlsGSP) / 335 = 192960/335
VST_GSP_delay = 576
```

For the GCK signal, you must also configure the delay and high or low width, since these two values always match. You must also set the number of GCK "closing pulses", meaning the needed pulses from last GEN signal.

Table 5: Timings from LS014B7DD01 Manual

		Time (ns)		
		Min	**Typical**	**Max**
thwGCK	High Level Width of GCK	95040	96480	97920
tlwGCK	Low Level Width	95040	96480	97920

Attention: Zero positions of INTB, GSP and GCK match.

In the case of GCK delay, the value must be calculated from the combination of thsINTB and thsGSP (see Figure 1). The value must be placed on VCK_GCK_delay.

```
VCK_GCK_delay = (thsINTB + thsGSP)/format_clk_period + XRST_INTB_delay
VCK_GCK_delay = ( (24120+48240)/335) + XRST_INTB_delay
VCK_GCK_delay = 216 + XRST_INTB_delay
```

For the GCK width to take effect, VCK_GCK_width must be configured.

```
VCK_GCK_width = thwGCK/format_clk_period = 96480 / 335
VCK_GCK_width = 288
```

GCK "closing pulses" is the last value to configure for the GCK signal. From the Vertical Timing diagram, the following GCK pulses after the last GEN signal pulse (GCK #563, #564, #565, #566, #567, #568) are needed for configuring the closing pulses. The number of extra pulses (in this case 6) must be placed on VCK_GCK_closing_pulses.

```
VCK_GCK_closing_pulses = 6
```

3.2 Horizontal Signal Timing Configuration

The next step is the configuration on Horizontal Signal Timing.

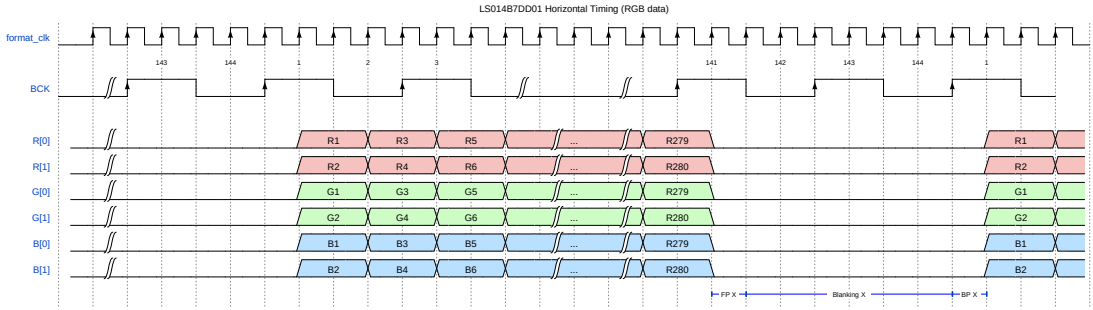

Figure 2: LS014B7DD01 Horizontal Timing (RGB data)

As it was mentioned in the first section, high and low width of BCK is controlled by the format_clk period, and the number of BCKs is dependent on the width of GCK.

The next signal for configuration is the GEN signal line:

Table 6: Timings from LS014B7DD01 Manual

		Time (ns)		
		Min	**Typical**	**Max**
thwGEN	GEN High width	29000	-	-
tsGCK1	GCK set-up time	19100	-	-
thGCK1	GCK hold time	19100	-	-

In the case of GEN assertion point and width, since these two only have minimum values to reach, the configuration values can vary. As an example, the chosen width is a little over the minimum width of GEN signal high width.

```
GEN High width value must be placed in ENB_GEN_width
ENB_GEN_width = (thwGEN + 1000ns)/335=(29000+1000)/335=89.6
ENB_GEN_width = 90
```

`tsGCK1` and `thGCK1` values must be calculated to configure GEN assertion point. GEN assertion point is calculated based on the width of GEN, GCK and for GEN high state to be in the middle of GCK pulse. Offset of the rising edge of GEN signal counts format clock cycles from the rising (or the falling) edge of the GCK signal and must be applied in the `ENB_GEN_delay`.

Since VCK_GCK_width and ENB_GEN_width express the width of the equivalent signals in terms of format_clk periods, the delay of GEN can be calculated based on them.

```
ENB_GEN_delay = (VCK_GCK_width-ENB_GEN_width)/2=(288-90)/2=99
ENB_GEN_delay=(VCK_GCK_width-ENB_GEN_width)/2
```

In the above described example 99*format_clk=33165ns covers the minimum values of tsGCK1 and thGCK1.

In the case of BSP assertion point and width, the following parameters must be configured to match the timings of the display specification. For the assertion point of BSP, `HST_BSP_delay` must be configured, while setting `HST_BSP_width` controls the high level width of BSP.

Table 7: Timings from LS014B7DD01 Manual

		Time (ns)		
		Min	**Typical**	**Max**
tsGCK2	BSP delay time from GCK clock pulse change	0	335	-
thsBSP	BSP data set-up time high level	330	335	340
tlsBSP	BSP data set-up time low level	330	335	340

In order to meet the specification of thsBSP and tlsBSP in this example, given the position of GCK and BCK transition, the delay of BSP (tsGCK2) has to be equal to 670ns.

```
HST_BSP_delay = tsGCK2/335=670/335
HST_BSP_delay=2
```

In order to calculate the width of BSP, the following equation must be used:

```
HST_BSP_width=(thsBSP + BCK clock period - tlsBSP)/335
HST_BSP_width=4
```

Last thing needed to configure is to set the BCK pulse number at the time where the pixel data are latched on the RGB lines. From the LS014B7DD01 Manual this seems to be on the pulse of BCK #1.

```
HCK_BCK_data_start=1
```

Wrapping up, here is the complete struct that holds the timing configuration of LS014B7DD01 panel and the `nemadc_set_mip_panel_parameters` function that takes

as arguments the struct that holds the configuration of the panel, in order to set the needed parameters in the MiP interface.

```
MiP_display_config_t LS014B7DD01;

LS014B7DD01.resx = 280;
LS014B7DD01.resy = 280;
LS014B7DD01.XRST_INTB_delay = 1;
LS014B7DD01.XRST_INTB_width = 566;
LS014B7DD01.VST_GSP_delay = 72 + LS014B7DD01.XRST_INTB_delay;
LS014B7DD01.VST_GSP_width = 576;
LS014B7DD01.VCK_GCK_delay = 216 + LS014B7DD01.XRST_INTB_delay;
LS014B7DD01.VCK_GCK_width = 288;
LS014B7DD01.VCK_GCK_closing_pulses = 6;
LS014B7DD01.HST_BSP_delay = 2;
LS014B7DD01.HST_BSP_width = 4;
LS014B7DD01.HCK_BCK_data_start = 1;
LS014B7DD01.ENB_GEN_width = 90;
LS014B7DD01.ENB_GEN_delay = (LS014B7DD01.VCK_GCK_width-LS014B7DD01.ENB_GEN_width)/2;

nemadc_set_mip_panel_parameters(&LS014B7DD01);
```

Attention: The `nemadc_set_mip_panel_parameters` function and timing configuration setup of the panel need to run only once in the beginning of the program.

4 Configuration of NemaDC (NemaDC output, layers and MiP interface)

Setting NemaDC output frame resolution and layers configuration in the case of MiP interface is handled within a new incorporated function that is required when working with MiP interface. In the following code example, you can observe that `nemadc_timing` and `nemadc_set_layer` functions are eliminated.

As indicated earlier, the `nemadc_mip_setup` function is needed for configuring the MiP interface, the setup of NemaDC output and layers. Also, an extra functionality that this function provides is the configuration of MiP interface to transmit the frame pixel data in up to 16 partial regions.

```
void nemadc_mip_setup( int layer0_active, nemadc_layer_t *layer0,
                       int layer1_active, nemadc_layer_t *layer1,
                       int layer2_active, nemadc_layer_t *layer2,
                       int layer3_active, nemadc_layer_t *layer3,
                       int partial_regions, ...);
```

Attention: The `nemadc_mip_setup` function is required to be called prior a single frame update.

The `nemadc_mip_setup` function takes as arguments the structs of the layers and an indication for each layer (0 or 1) if the given layer must be active on that frame.

Also, there is an argument to indicate the number of partial regions needed to send on the current frame update. This argument can vary from 0 (full frame update), up to 16.

The next arguments the function can accept are the partial start rows and partial end rows in pairs of the needed partial regions. If the partial regions you are configuring are more than the partial regions you indicate in the `partial_regions` argument, the extra regions are ignored.

Partial Region Row Start of a partial region must be bigger than the Partial Region Row End of the previous region.

Example of full frame update with 2 active layers

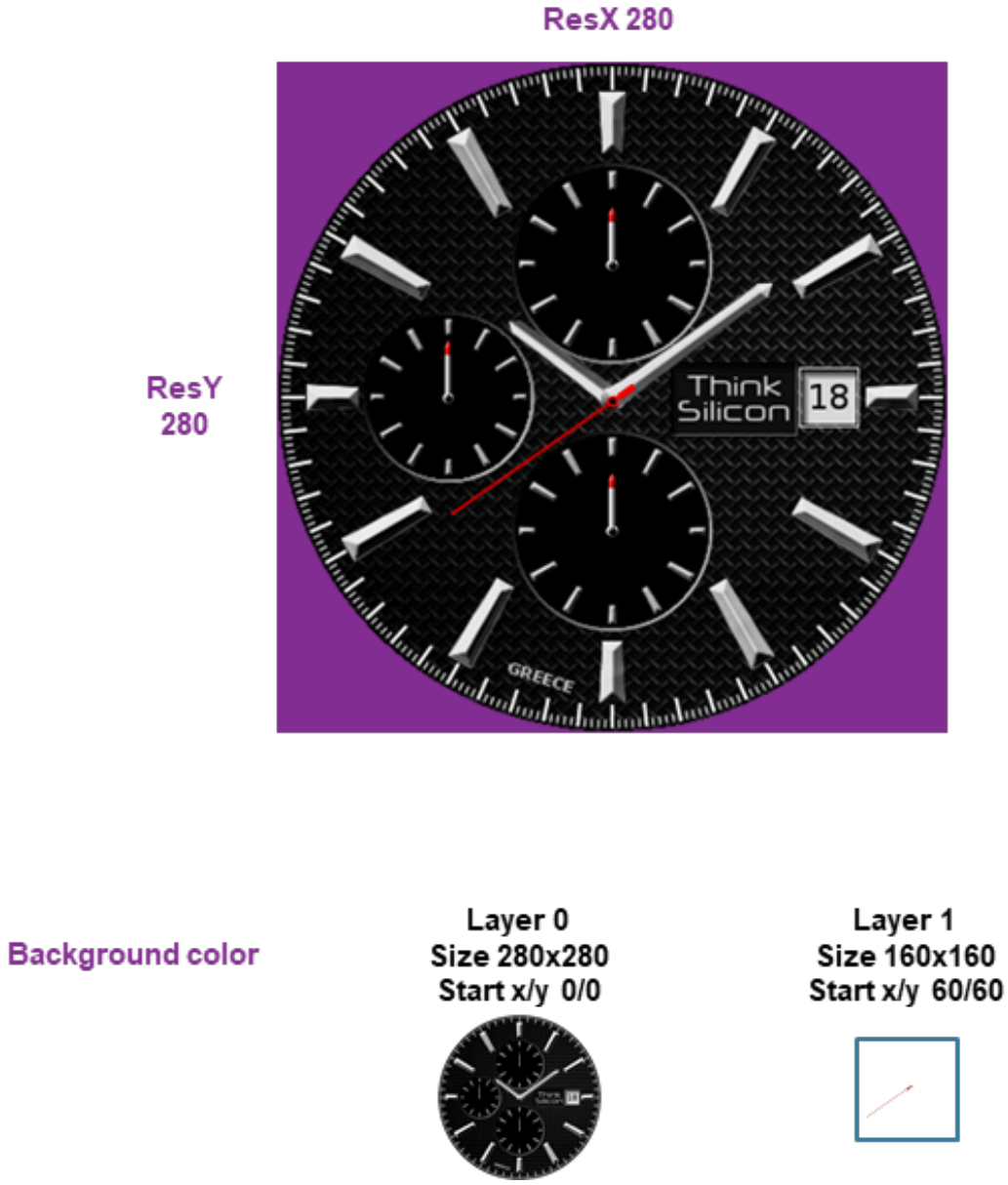

Figure 3: Full frame update with 2 active layers

```
#define LAYER_ACTIVE 1
#define LAYER_INACTIVE 0

// Full frame update -> set partial regions to 0.
nemadc_mip_setup( LAYER_ACTIVE , &layer[0],
                  LAYER_ACTIVE , &layer[1],
```

```
                LAYER_INACTIVE, NULL,
                LAYER_INACTIVE, NULL,
                0 );
```

Example with 3 partial regions update per frame with 2 active layers

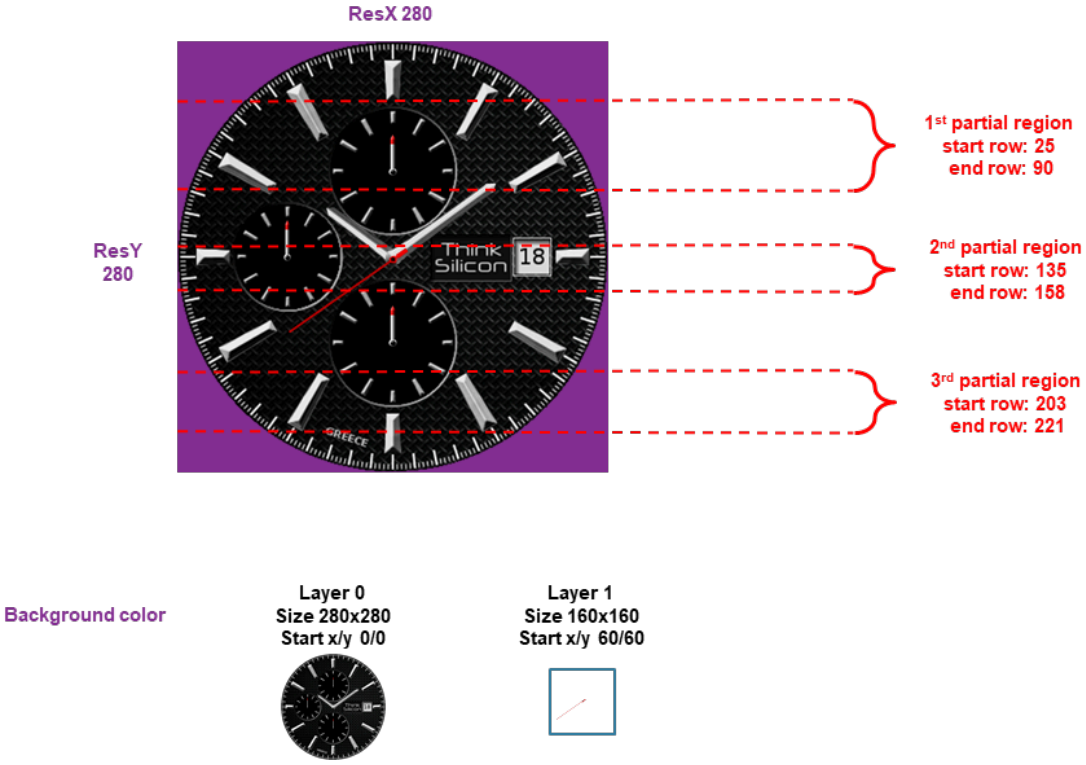

Figure 4: Three partial regions update with 2 active layers

```
#define LAYER_ACTIVE 1
#define LAYER_INACTIVE 0
nemadc_mip_setup( LAYER_ACTIVE , &layer[0],
                  LAYER_ACTIVE , &layer[1],
                  LAYER_INACTIVE, &layer[2],
                  LAYER_INACTIVE, &layer[3],
                  3, // Partial regions
                  25, 90,
                  135, 158,
                  203, 221
                  );
```

Full code example with the use of the MiP Parallel interface

The following code example demonstrates the 3 partial regions update mentioned earlier and gives the full configuration of NemaDC.

Assuming that PLL clock (input pll_clk to NemaDC Clock Divider) is configured to run at 38.86MHz and through software configuration of NemaDC's clock divider, the example code is targeting to format_clk to be equal to 2.98MHz, while pixel_clk to be equal to PLL clock.

```
#include "nema_dc.h"
#include "nema_dc_jdi.h"

MiP_display_config_t LS014B7DD01;

nemadc_layer_t layer[4] = {{0}};

main()
{
    int ret;

    //Initialize Nema|dc
    ret = nemadc_init();
    if (ret) return ret;

    LS014B7DD01.resx = 280;
    LS014B7DD01.resy = 280;
    LS014B7DD01.XRST_INTB_delay = 1;
    LS014B7DD01.XRST_INTB_width = 566;
    LS014B7DD01.VST_GSP_delay = LS014B7DD01.XRST_INTB_delay + 72;
    LS014B7DD01.VST_GSP_width = 576;
    LS014B7DD01.VCK_GCK_delay = LS014B7DD01.XRST_INTB_delay + 216;
    LS014B7DD01.VCK_GCK_width = 288;
    LS014B7DD01.VCK_GCK_closing_pulses = 6;
    LS014B7DD01.ENB_GEN_width = 90;
    LS014B7DD01.ENB_GEN_delay = (LS014B7DD01.VCK_GCK_width - LS014B7DD01.EN►
B_GEN_width) / 2;
    LS014B7DD01.HST_BSP_delay = 2;
    LS014B7DD01.HST_BSP_width = 4;
    LS014B7DD01.HCK_BCK_data_start = 1;

    nemadc_clkdiv(13, 1, 4, 0); // pll_in clk = 38.86 MHz

    // Swap pixel_clk / format_clk on Clock Divider
    nemadc_reg_write(NEMADC_REG_CLKCTRL_CG, (NemaDC_clkctrl_cg_clk_swap |
NemaDC_clkctrl_cg_clk_en));

    layer[0].resx = 280;
    layer[0].resy = 280;
    layer[0].format = NEMADC_RGBA2222;
    layer[0].blendmode = NEMADC_BL_SRC;
    layer[0].stride = layer[0].resx;
    layer[0].alpha = 0xff;
```

```
    layer[0].startx = 0;
    layer[0].starty = 0;
    layer[0].flipx_en = 0;
    layer[0].flipy_en = 0;
    layer[0].baseaddr_virt = BASE_OF_BG_WATCHFACE;
    layer[0].baseaddr_phys = (unsigned)tsi_virt2phys(BASE_OF_BG_WATCHFACE);

    layer[1].resx = 160;
    layer[1].resy = 160;
    layer[1].format = NEMADC_RGBA2222;
    layer[1].blendmode = NEMADC_BL_SRC;
    layer[1].stride = layer[1].resx;
    layer[1].alpha = 0xff;
    layer[1].startx = 60;
    layer[1].starty = 60;
    layer[1].flipx_en = 0;
    layer[1].flipy_en = 0;
    layer[1].baseaddr_virt = BASE_OF_SEC_LAYER;
    layer[1].baseaddr_phys = (unsigned)tsi_virt2phys(BASE_OF_SEC_LAYER);

    // Set NemaDC output Background color.
    nemadc_set_bgcolor(0x863094ff);

    // Set Panel Timing Configuration.
    nemadc_set_mip_panel_parameters(&LS014B7DD01);

    #define LAYER_ACTIVE 1
    #define LAYER_INACTIVE 0
    #define FRAME_END_INTERRUPT (1<<4)

    // Run for 10 frames.
    for (int i = 0; i < 10; ++i) {
        nemadc_reg_write(NEMADC_REG_INTERRUPT, FRAME_END_INTERRUPT);
        // Configure NemaDC to send for this frame.
        nemadc_mip_setup( LAYER_ACTIVE , &layer[0],
                          LAYER_ACTIVE , &layer[1],
                          LAYER_INACTIVE, &layer[2],
                          LAYER_INACTIVE, &layer[3],
                          3, // Partial regions
                          25, 90,
                          135, 158,
                          203, 221
                          );

        // Single Frame Update.
        nemadc_set_mode(NEMADC_ONE_FRAME | NEMADC_MIP_IF | NEMADC_SCANDOUBLE);

        nemadc_wait_for_irq();

        nemadc_set_mode(0);
    }
}
```