Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

Contact us

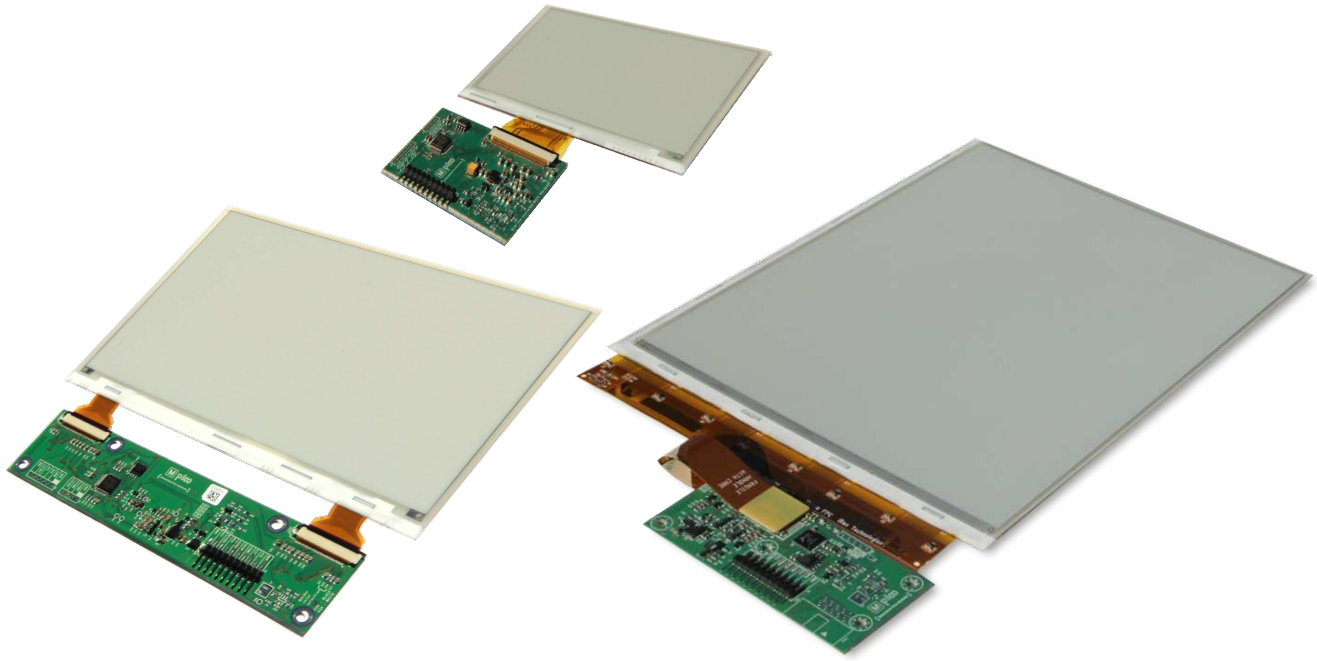
Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

Embedded Pico Systems

Developer's Guide

Timing Controller Solutions for Pervasive Displays 4.41″, 7.4″ and 10.2″ Panels

TCM-P441-230_v1.0, TC-P441-230_v1.0
TCM-P74-230_v1.0, TC-P74-230_v1.0
TCM-P102-220_v1.1



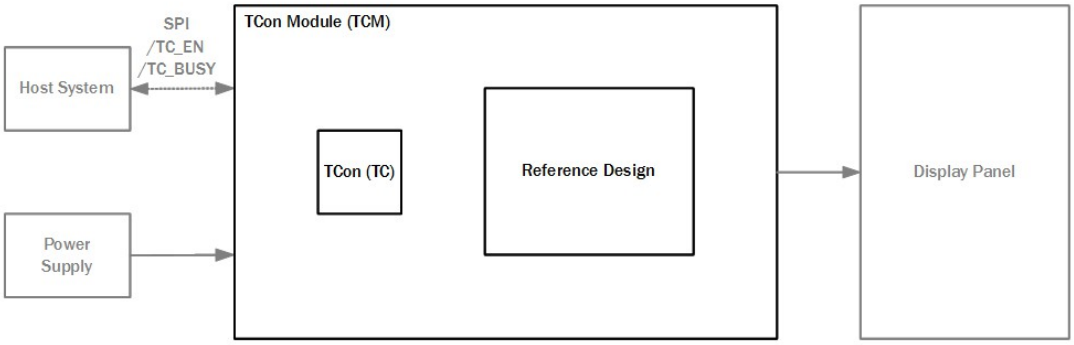
Classification: **Public**

Document Revision: **F**

Table of Contents

1 Introduction

E-paper Timing Controller Solutions provide timing controller (TCon) functionalities for **Pervasive Displays**' large size panels (**4.41″, 7.4″, and 10.2″**[1]). Solution for each of the panels provides identical functionality, command set and physical interface. Offered as a chip only (**Timing Controller – TC**) or as fully-assembled PCB module (**Timing Controller Module – TCM**), the solution allows a quick and easy integration with your host system, minimizing the cost and time-to-market.



TCon (as well as TCM) can be connected to a host microsystem via fast and reliable Serial Peripheral Interface (SPI). TCon is controlling both the source and gate drivers, composing waveforms required to generate high quality images on the display.

1.1 Supported Display Panels

TCon Module Part #	TCon Part #	Display Type	Display Part #	Display Resolution	Display Density
TCM-P441-230_v1.0	TC-P441-230_v1.0	4.41″ (v230 FPL)	RET044BS011[2]	400×300 px	113 dpi
TCM-P74-230_v1.0	TC-P74-230_v1.0	7.4″ (v230 FPL)	MEW074BT011[2]	480×800 px	126 dpi
TCM-P102-220_v1.1	–	10.2″ (v220 FPL)	MEZ102AT011	1024×1280 px	160 dpi

1.2 Features

- SPI interface to host
- SPI (slave device) with additional /TC_EN and /TC_BUSY lines
- 1-bit color (black and white)
- Temperature compensation
- Internal image buffer retains content during system power down

1.3 Characteristics

- From 2.7 to 3.3 V supply voltage
- From 0 to 40[3]/50[4] °C operating temperature range

1) TCon for 10.2″ display is not yet available
2) This display panel MPN may not become available for mass production
3) v230 FPL displays characteristics
4) V220 FPL displays characteristics

2 Outline

2.1 TCon

The information below applies to TC-P441-230_v1.0 and TC-P74-230_v1.0 products.

LQFP48: plastic low profile quad flat package; 48 leads; body 7×7 × 1.4 mm

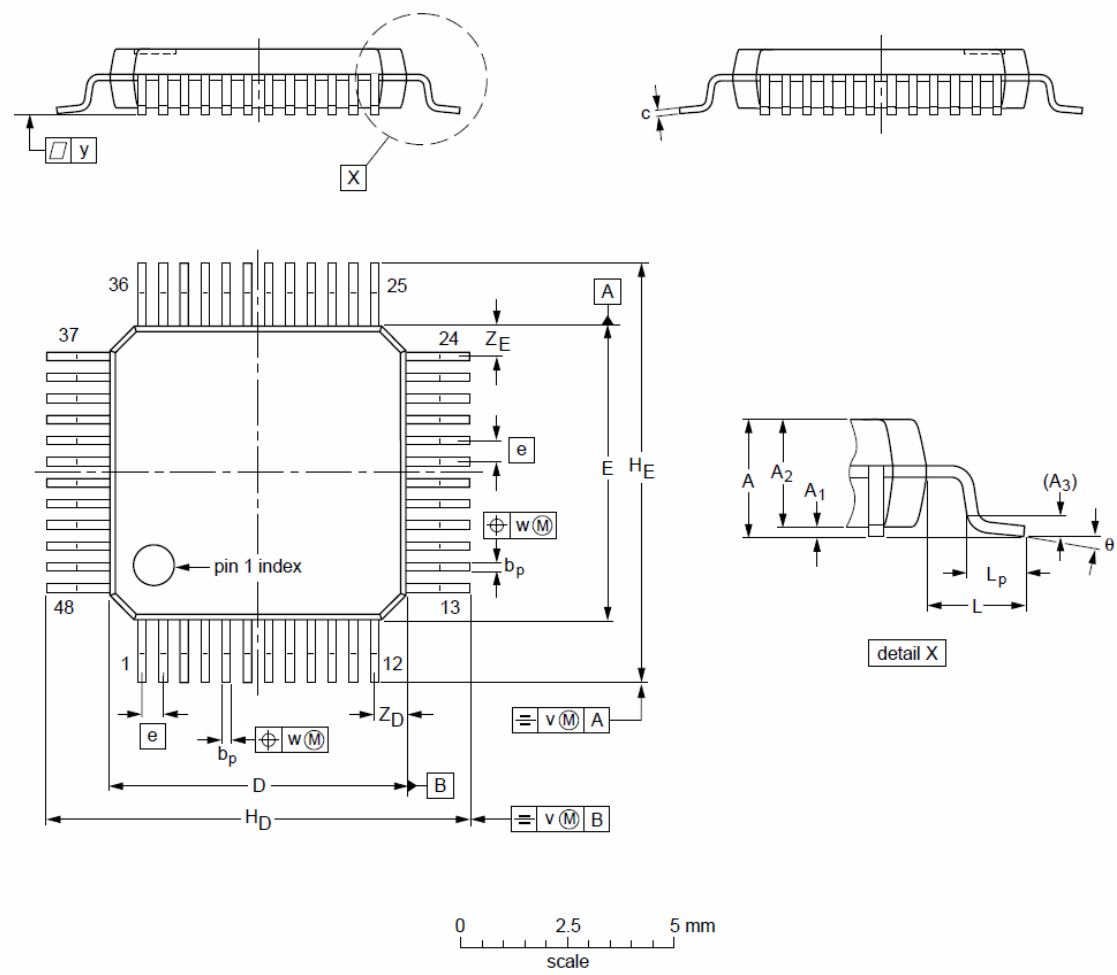


Figure 2.1: TCon dimensions

Unit	A max	A_1	A_2	A_3	b_p	c	$D^{(1)}$	$E^{(1)}$	e	H_D	H_E	L	L_p	v	w	y	$Z_D^{(1)}$	$Z_E^{(1)}$	θ
mm	1.6	.20	1.45	0.25	0.27	0.18	7.1	7.1	0.5	9.15	9.15	1	0.75	0.2	0.12	0.1	0.95	0.95	7°
		.05	1.35		0.17	0.12	6.9	6.9		8.85	8.85		0.45				0.55	0.55	0°

Table 2.1: Dimensions (mm are the original dimensions)[1]

1

2.2 TCon Module

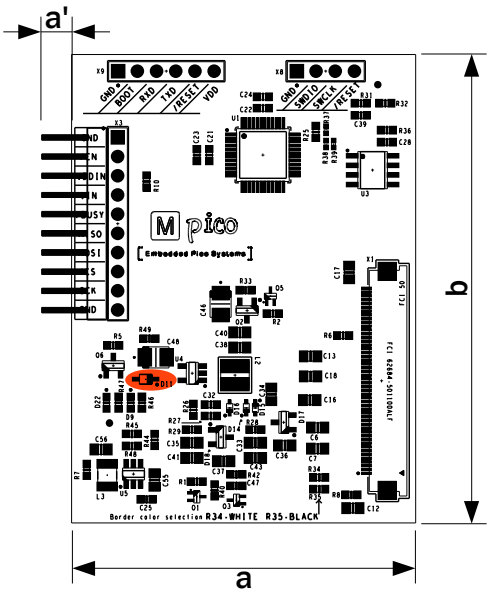


Figure 2.2: TCM-P441-230 Outline

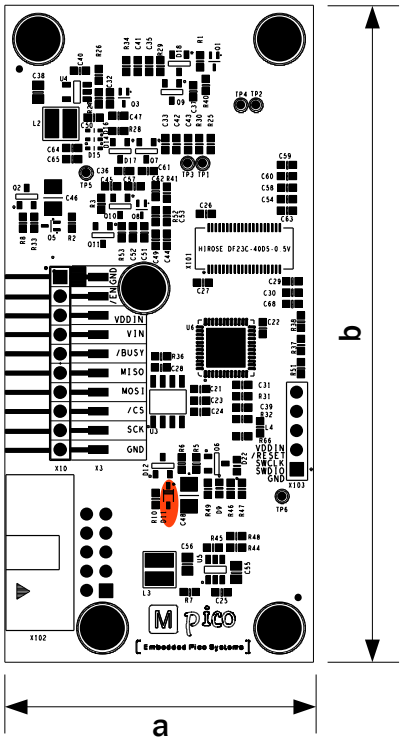


Figure 2.4: TCM-P102-220 Outline

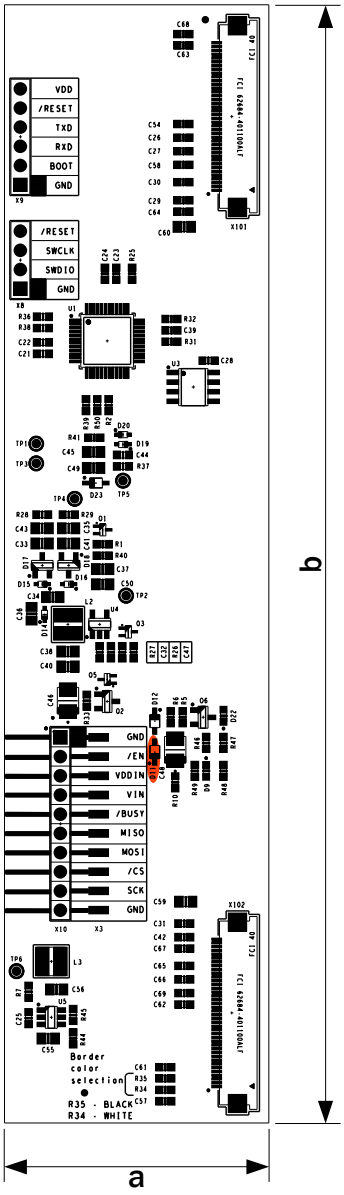


Figure 2.3: TCM-P74-230 Outline

Module	Dimensions (mm)		
	a	a'	b
TCM-P441-230	45.5	4.0	62.0
TCM-P74-230	35.0	–	148
TCM-P102-220	41.0	–	87.0

Table 2.2: TCM physical dimensions

NOTE TCM features solder pads for overvoltage protection 3.6 V Zener diode (D11). The diode is by default not mounted to limit the TCM current consumption. If required, the diode can be mounted in the designated spot at the Customer's own account. The diode placement is marked orange on the figures 2.2, 2.3 and 2.4. MpicoSys recommends using BZX384 3V6 diode. This will increase the average current consumption by 1 mA during all operations.

3 Electrical Characteristics

Unless specified otherwise, the values in this chapter are applicable to the whole product family, and both to TC and TCM.

3.1 Absolute Maximum Ratings

Symbol	Description	Min	Typ	Max	Unit
VDDIN	Digital supply voltage	0	–	3.6	V
VIN	Analog supply voltage	–0.3	–	6.0	V
T_{st}	Storage temperature	–20	–	+60	°C

Table 3.1: Absolute maximum ratings

3.2 Operating Conditions

Symbol	Description	Min	Typ	Max	Unit
VDDIN	Standard digital operating voltage	2.7	3.0	3.3	V
VIN	Standard analog operating voltage	2.0	3.0	5.5	V
T_{op}	Operating temperature	0	+23	+40[5]/+50[6]	°C

Table 3.2: Typical operating conditions

3.3 TCM Supply Current Characteristics

Measurement Setup

Current consumption measured with Agilent 34411A Multimeter;

VDDIN shorted with VIN; range from 2.7 V to 3.3 V.

NOTE Values vary with ambient temperature, supply voltage and the displayed pattern.

5) TCM-P441-230 and TCM-P74-230
6) TCM-P102-220

4.41" v230

Symbol	Description	Operation	Min	Max	Unit
IDD	Average current consumption	Display update	21.7	31.0	mA
		Data reception on SPI	21.9	23.0	mA
		Disabled (/TN_EN inactive)	0.02	1.0	µA
E	Average energy consumption in room temperature	Display update	145	216	mJ

Table 3.3: 4.41" v230 supply current characteristics

7.4" v230

Symbol	Description	Operation	Min	Max	Unit
IDD	Average current consumption	Display update	38.3	108	mA
		Data reception on SPI	23.1	24.0	mA
		Disabled (/TN_EN inactive)	0.02	1.0	µA
E	Average energy consumption in room temperature	Display update	233	649	mJ

Table 3.4: 7.4" v230 supply current characteristics

10.2" v220

Symbol	Description	Operation	Min	Max	Unit
IDD	Average current consumption	Display update	35.2	142	mA
		Data reception on SPI	16.1	16.4	mA
		Disabled (/TN_EN inactive)	0.02	1.0	µA
E	Average energy consumption in room temperature	Display update	286	1,030	mJ

Table 3.5: 10.2" v220 supply current characteristics

Measurement Results Conditions

The below table describes conditions at which the results from tables above were achieved. *ESL* images are presented below the table. *Checkerboard* image is a 1 pixel by 1 pixel black and white checkerboard fulfilling the whole display area.

Measurement		Value	Power Supply (VDD = VIN) [V]	Image Used for Measurement	Ambient Temp. [°C]
Average current consumption	Display update	Min	3.3	Transition *ESL* to *ESL*	22
		Max	2.7	Transition *Checkerboard* to *Checkerboard*	22
	Data reception on SPI	Min	2.7	*ESL*	22
		Max	3.3	*Checkerboard*	22
Average energy consumption in room temperature	Display update	Min	2.7	ESL	22
		Max	3.3	*Checkerboard*	22

Table 3.6: Measurement results conditions

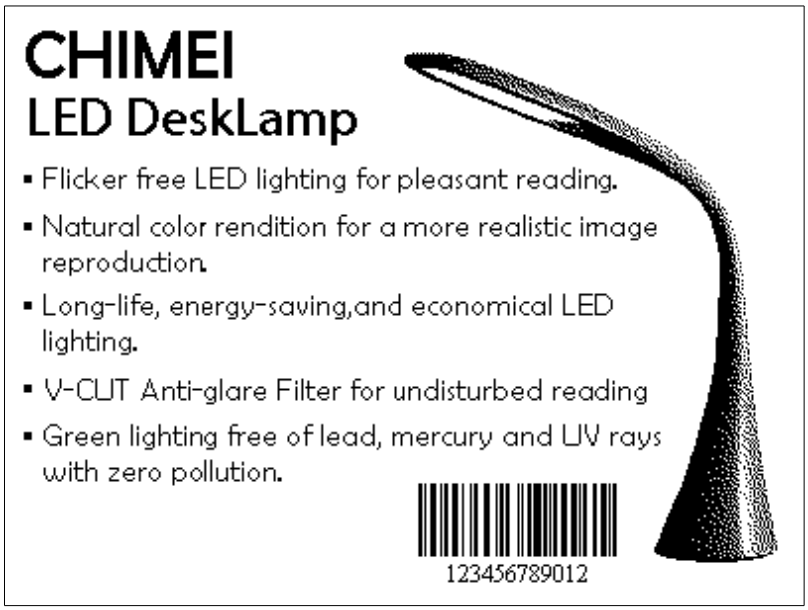


Figure 3.1: 4.41" ESL image

Figure 3.2: 7.4" ESL image

Figure 3.3: 10.2″ ESL image

3.4 DC Characteristics

Symbol	Description	Min	Max	Unit
VIH	Input high level voltage	0.7×VDD	–	V
VIL	Input low level voltage	–	0.3×VDD	V
VOH	Output high level voltage	VDD-0.4	–	V
VOL	Output low level voltage	–	0.4	V

Table 3.7: Typical operating conditions

4 Display Refresh Time

T_{amb} [°C]		0÷5	5÷10	10÷15	15÷20	20÷25	25÷30	30÷35	35÷40	40÷45	45÷50
Time [s]	4.41" v230	4.6	3.7	2.5	1.5	1.4	1.3	1.1	1.1	–	–
	7.4" v230	5.3	4.2	2.8	1.4	1.5	1.7	1.7	1.6	–	–
	10.2" v220	5.3	3.8	2.9	2.7	2.6	2.6	2.3	2.0	2.1	2.1

Table 4.1: Display refresh time versus ambient temperature

5 TCon Hands-on

Unless specified otherwise, all information contained in this chapter is applicable to the whole product family.

5.1 TCon Integration

TCon together with the reference schematic can be integrated with user's own host system. This enables the user to develop their own application utilizing e-paper technology.

Reference design is included in the Design Guide, distributed separately. Please contact sales@mpicosys.com for more information.[7]

5.2 TCM Interconnection

Use the below described host connector to connect TCM to your host system. It is a 10-pin single-row 2.54 mm-pitch male header.

NOTE Forward slash "/" in front of the pin name indicates the signal is active low

Pin #	Pin Name	Remarks
1	GND	Supply ground
2	/TC_EN	TC enable
3	VDDIN	Power supply for digital part
4	VIN	Power supply for analog part
5	/TC_BUSY	Host interface busy output
6	TC_MISO	Host interface data output
7	TC_MOSI	Host interface data input
8	/TC_CS	Host interface chip select input
9	TC_SCK	Host interface clock input
10	GND	Supply ground

Table 5.1: TCM host connector

7) TCon for 10.2" display is not yet available

5.3 TCM Power On

Connect your power supply to the VDDIN and VIN pins.

VDDIN supply for digital part has to be supplied from a stable power supply, e.g. stabilized by a DC/DC converter or a low-dropout regulator (LDO).

VIN can either be supplied directly from the battery (e.g. coin-cell) for improved efficiency, or can be shorted to VDDIN.

When connected to power supply, TCM is by default turned off to conserve energy. To switch it on, activate the /TC_EN signal.

5.4 Image Slot

TCon features one slot for storing image data. The image is stored in flash memory, thus it is retained when the system is not powered.

5.5 Interface

Connection To Host

User's host system can communicate with TCon via Serial Peripheral Interface (SPI) with additional /TC_EN and /TC_BUSY line. TCon works as a SPI slave device. TCon power has to be supplied by the host system. The SPI supports 8-bit frames of data flowing from the master to the slave and from the slave to the master.

Signals

Inputs:

- /TC_EN – active low
- /TC_CS – active low
- TC_SCK
- TC_MOSI

Outputs:

- TC_MISO
- /TC_BUSY – active low

SPI Settings

- Bit rate – up to 3 MHz
- Polarity – CPOL = 1; clock transition high-to-low on the leading edge and low-to-high on the trailing edge
- Phase – CPHA = 1; setup on the leading edge and sample on the trailing edge
- Bit order – MSB first
- Chip select polarity – active low

Reference SPI timing diagram below:

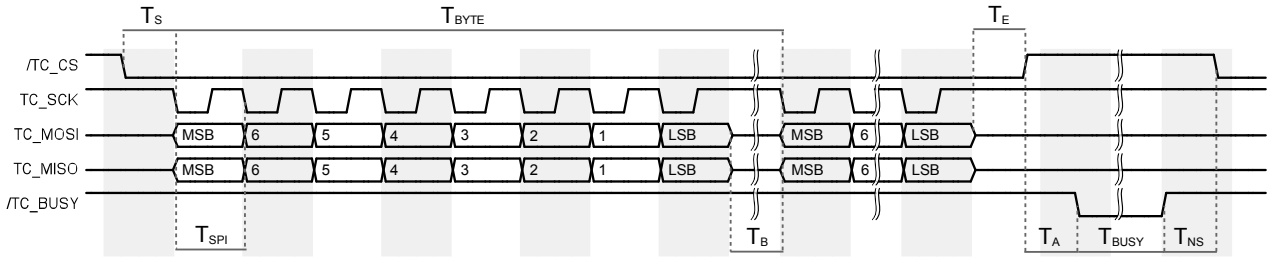


Figure 5.1: SPI timing diagram

Time	T_S	T_{BYTE}[1]	T_{SPI}[1]	T_B[1]	T_E	T_A	T_{BUSY}	T_{NS}
Min.	6.0 µs		82 ns	0	7.0 µs		30 µs	2.0 µs
Typ.		2.67 µs	333 ns	1.34 µs		25 µs		
Max.			8 us					

Table 5.2: TC-P441-230 SPI timing description

Time	T_S	T_{BYTE}	T_{SPI}	T_B	T_E	T_A	T_{BUSY}	T_{NS}
Min.	6.0 µs		82 ns	0	11.0 µs	5 µs	21 µs	2.0 µs
Typ.		2.67 µs	333 ns					
Max.			8 us					

Table 5.3: TC-P74-230 SPI timing description

Time	T_S	T_{BYTE}	T_{SPI}	T_B	T_E	T_A	T_{BUSY}	T_{NS}
Min.	4.0 µs		166 ns	0	1.0 µs	6.4 µs	12.4 µs	1.5 µs
Typ.		2.67 µs	333 ns			10 µs		
Max.			1 ms					

Table 5.4: TC-P102-220 SPI timing description

Communication Flow

TCon is able to communicate to the host system if /TC_BUSY signal is inactive. To start communication, the /TC_CS line has to be activated by the host. Then the command data can be passed. There is no timeout during the communication, so the command data can be passed with any delays. Only when /TC_CS line is deactivated, is the command interpreted by the TCon.

After passing the command, it is being interpreted and executed by the TCon. The time of execution is indicated by /TC_BUSY signal active. During this time, the TCon does not accept any new commands.

1 Minimum T_{BYTE} value and typical T_{SPI} value reflect the maximum supported bit rate of 3 MHz. In this case T_B can equal 0 (typical value). However, the SPI clock can be set to higher frequency – up to 6 MHz – but in that case T_B value needs to be increased accordingly, so that T_{BYTE} minimum value is ensured.

Startup and Initialization Sequence

TC-P441-230 and TC-P74-230:

The below timing diagram (Figure 5.2) represents the TCon startup and initialization sequence after power-up. The TCon is ready for communication after $T_{STARTUP}$ + T_{INIT} which is indicated by /TC_BUSY rising edge.

T_{INIT} time is constant, whereas $T_{STARTUP}$ is related to flash memory access and increases with every use cycle. At the first use cycle the time has the minimum value as in the Table 5.5 below. After 256 use cycles the flash memory page storing initialization values is erased and the time increases; subsequently the time decreases back to the minimum value. The maximum value represents worst-case performance after 100,000 flash memory erase cycles.

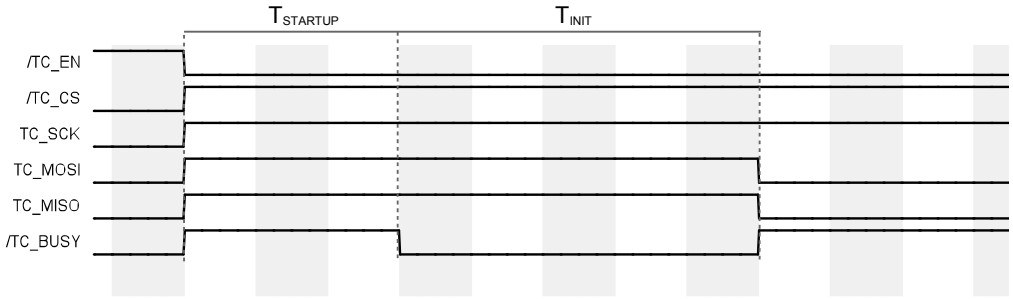


Figure 5.2: TC-P441-230 and TC-P74-230 initialization sequence

Time	Min	Max	Unit
$T_{STARTUP}$	3.0	3.0	ms
T_{INIT}	1	200	ms

Table 5.5: TC-P441-230 and TC-P74-230 startup and initialization times

TC-P102-220:

The below timing diagram (Figure 5.3) represents the TCon startup and initialization sequence after power-up. The TCon is ready for communication after $T_{STARTUP+INIT}$ which is indicated by /TC_BUSY rising edge.

$T_{STARTUP+INIT}$ is related to flash memory access and increases with every use cycle. At the first use cycle the time has the minimum value as in the Table 5.6 below. After 256 use cycles the flash memory page storing initialization values is erased and the time increases; subsequently the time decreases back to the minimum value. The maximum value represents worst-case performance after 100,000 flash memory erase cycles.

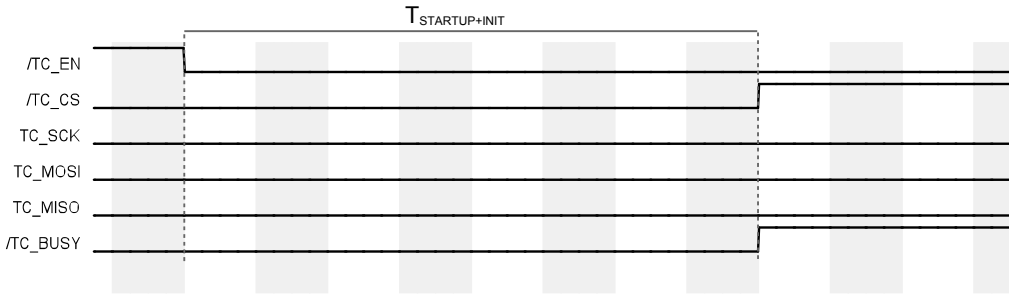


Figure 5.3: TC-P102-220 initialization sequence

Time	Min	Max	Unit
$T_{STARTUP+INIT}$	2.2	200	ms

Table 5.6: TC-P102-220 startup and initialization time

5.6 Command Description

Command Format

Each command is built up from 3 to 255 bytes. The command is divided into six fields.

The first three fields are used in each command:

- *INS* – command group specific
- *P1* – parameter
- *P2* – parameter

whereas the next three fields are only used by some particular commands:

- *Lc* – number of bytes in *Data* field
- *Data* – bytes forming command data; number of bytes determined by *Lc*
- *Le* – number of bytes of expected response

Returned Values

Upon each command, TCon returns a 2-byte command status code. The command status code is not included in the *Le* (expected response length).

Possible status codes are as follows:

- 0x9000 – EP_SW_NORMAL_PROCESSING – command successfully executed
- 0x6700 – EP_SW_WRONG_LENGTH – incorrect length (invalid *Lc* value or command too short or too long)
- 0x6C00 – EP_SW_INVALID_LE – invalid *Le* field
- 0x6A00 – EP_SW_WRONG_PARAMETERS_P1P2 – invalid *P1* or *P2* field
- 0x6D00 – EP_SW_INSTRUCTION_NOT_SUPPORTED – command not supported

If a command returns specific data, the status code is attached to the end of the data.

Data Readout

During each SPI clock cycle, a full-duplex data transmission takes place: the host sends a bit on the MOSI line, and the TCon sends a bit on the MISO line at the same time.

Thus, the command status should be read after the command is executed. To read the command status, the host should send the expected number of 0x00 bytes to TCon. The amount of bytes to be sent is dependent on the type of a command:

- If a command does not use the *Le* field, it will return only the two-byte status code; thus only two bytes should be sent by the host
- When *Le* field is used and set to 0x00, the response length is not determined; then the response should be read until 0x00 is encountered, indicating the response termination, and two additional bytes should be sent to acquire the command status
- When *Le* field is set to a value other than 0x00, the response length is determined by the value at *Le* field. The host should send the number of bytes indicated by the *Le* field, and two additional bytes to acquire the command status

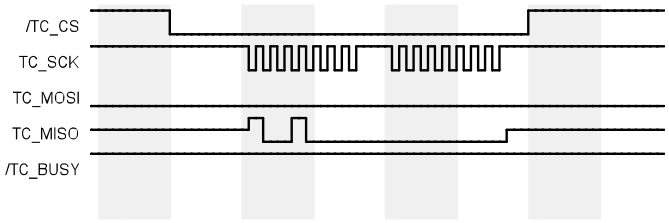


Figure 5.4: Example readout - 0x9000 response

5.6.1 Upload Image Data

This group of commands, starting with 0x20, handles the process of data upload to the TCon memory.

5.6.1.1 UploadImageData

Command

INS	P1	P2	Lc	Data
0x20	0x01	0x00	Data packet size (max 0xFA)	[Lc Data bytes]

Description

The command uploads image data (in EPD file format) to TCon image memory. The data needs to be divided into packets and transferred with multiple UploadImageData commands. In order to send the full image data, the user has to make sure to send it packet by packet.

While writing to the TCon internal memory, the TCon data pointer will be internally increased by the size of the current packet, until reaching maximum of slot memory:

- 16,384 bytes in case of TC-P441
- 49,152 bytes in case of TC-P74

Data pointer will then start from the beginning.

Data

Image file in EPD format, see 6 EPD File Format). Maximum packet size is 251 bytes (as maximum command size is 255 bytes.)

Possible return values

- 0x9000
- 0x6700
- 0x6C00
- 0x6A00

5.6.1.2 ResetDataPointer

Command

INS	P1	P2
0x20	0x0D	0x00

Description

The command resets data pointer for Upload Image Data command.

NOTE Data pointer is automatically reset when TCon is enabled by /TC_EN activation

Possible return values

- 0x9000
- 0x6700
- 0x6C00
- 0x6A00

5.6.1.3 DisplayUpdate

Command

INS	P1	P2
0x24	0x01	0x00

Description

The command starts the display refresh sequence, displaying the current content of the image memory.

- If data was uploaded with UploadImageData command, the new data is going to be displayed
- If no data was sent, currently visible image will be refreshed (cleared and displayed again)

Possible return values

- 0x9000
- 0x6700
- 0x6C00
- 0x6A00

5.6.2 Device Info

This group of commands, starting with INS = 0x30 byte, manages the acquirement of hardware information from TCon.

5.6.2.1 GetDeviceInfo

Command

INS	P1	P2	Le
0x30	0x01	0x01	0x00

Description

The command returns information on system hardware. String data is specific for the particular device type and is constant for the same type of devices if no hardware differences occur.

Possible return values

- [String: "MpicoSys TC-P441-230_v1.0" terminated by 0x00 byte] + 0x9000
- [String: "MpicoSys TC-P74-230_v1.0" terminated by 0x00 byte] + 0x9000
- [String: "MpicoSys TC-P102-220_v1.1" terminated by 0x00 byte] + 0x9000
- 6700
- 6C00
- 6A00

5.6.2.2 GetDeviceId

Command

INS	P1	P2	Le
0x30	0x02	0x01	0x14

Description

The command returns unique device ID number.

Possible return values

- [20 bytes of data] + 0x9000
- 6700
- 6C00
- 6A00

5.6.3 System Info

This group of commands, starting with INS = 0x31 byte, deals with acquirement of firmware information from TCon.

5.6.3.1 GetSystemInfo

Command

INS	P1	P2	Le
0x31	0x01	0x01	0x00

Description

The command returns information on system firmware.

Possible return values

- [String: "MpicoSys TC-P441-230_fD_BIN" terminated by 0x00 byte] + 0x9000
- [String: "MpicoSys TC-P74-230_fC_BIN" terminated by 0x00 byte] + 0x9000
- [String: "MpicoSys TC-P102-220_fG_BIN" terminated by 0x00 byte] + 0x9000
- 6700
- 6C00
- 6A00

5.6.3.2 GetSystemVersionCode

Command

INS	P1	P2	Le
0x31	0x02	0x01	0x10

Description

The command returns information on system version.

Possible return values

- 0x D0 A5 00 03 00 00 00 00 33 01 03 00 00 00 00 00 + 0x9000 in case of TC-P441-230
- 0x D0 AA 00 01 00 00 00 00 3A 01 03 00 00 00 00 00 + 0x9000 in case of TC-P74-230
- 0x D0 AC 01 06 00 00 00 00 3D 01 00 00 00 00 00 00 + 0x9000 in case of TC-P102-220
- 6700
- 6C00
- 6A00

5.6.4 Sensor Data

5.6.4.1 ReadSensorData

Command

INS	P1	P2	Le
0xE5	0x01	0x00	0x02

Description

The command returns the temperature value measured by the TCon temperature sensor. The sensor is built in the TCM board and is included in the TCon reference design. The measurement is based on a NCP18WB473E03RB Thermistor and 8-bit ADC. The read value (x) is mapped to temperature value according to the following chart:

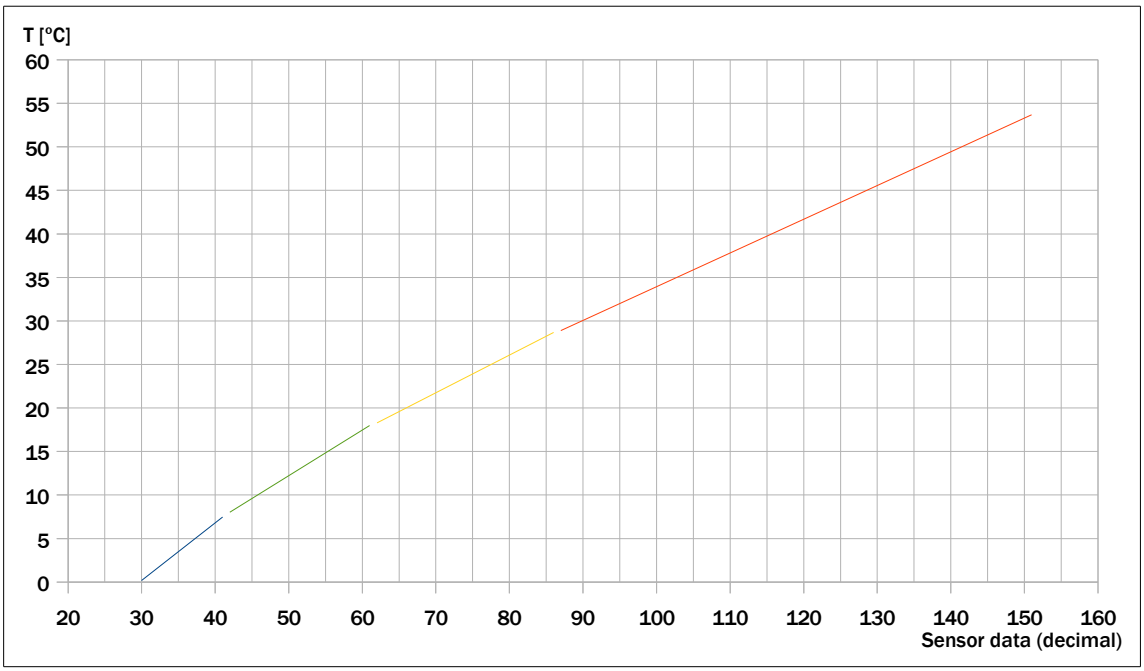


Figure 5.5: Temperature versus ADC 8-bit value chart

For more accurate approximation, the ADC values have been divided into four ranges. The temperature can be calculated based on the following linear approximation formula:

$$T = ax + b$$

where:
T – temperature [°C]
x – sensor data

x	30÷41	42÷61	62÷86	87÷160
a	0.66	0.52	0.43	0.39
b	–19.69	–13.95	–8.55	–4.75

Table 5.7: Linear approximation of the sensor data readout

Possible return values

- [2 bytes of sensor data] + 0x9000
- 6700
- 6C00
- 6A00

6 EPD File Format

EPD is a specific raster graphics image file format, accepted by TCon. EPD file format was developed to maximize the decoding efficiency on the target platform. The EPD file comprises of two parts:

- Header
- Image data

Table below describes the various panels resolution and corresponding image data array sizes, as well as EPD files sizes.

Panel size	Image resolution [px]	Image color depth [bit]	Header size [bytes]	Image data array size [bytes]	EPD file size [bytes]
4.41″	400×300	1	16	15,000	15,016
7.4″	480×800			48,000	48,016
10.2″	1024×1280			163,840	163,856

6.1 Header

EPD file begins with a header. The header size is 16 bytes. The consecutive bytes are described in the table below:

Field name	Size	Possible values	Description	
panel type	1 byte	0x33	Panel code	4.41″
		0x3A		7.4″
		0x3D		10.2″
X res	2 bytes	0x0190	400 px	
		0x01E0	480 px	
		0x0400	1024 px	
Y res	2 bytes	0x012C	300 px	
		0x0320	800 px	
		0x0500	1280 px	
color depth	1 byte	0x01	Image color depth – 1-bit (black and white)	
pixel data format	1 byte	0x00	Image pixel data format type 0	
		0x02	Image pixel data format type 2	
		0x04	Image pixel data format type 4	
RFU	9 bytes	0x00	Reserved for future use	

Based on the information from the table above, here are complete header values depending on the panel size:

- TC-P441-230: 0x 33 01 90 01 2C 01 00 00 00 00 00 00 00 00 00 00
  0x 33 01 90 01 2C 01 02 00 00 00 00 00 00 00 00 00
- TC-P74-230: 0x 3A 01 E0 03 20 01 04 00 00 00 00 00 00 00 00 00
- TC-P102-220 0x 3D 04 00 05 00 01 00 00 00 00 00 00 00 00 00 00

6.2 Image Data

Each byte of the image data encodes information on eight pixels (a single pixel is described by one bit of a single byte).

1-bit gray scale provides 2 colors. Bit value 0 corresponds to black color while value 1 represents white color.

6.2.1 Pixel Data Format Type 0

This format is used in TC-P441-230 and TC-P102-220. Each byte of image data shall convey information on 8 consecutive pixels of the RAW image.

Conversion Algorithm

The algorithm for conversion from standard RAW 4-bit data to EPD format is described below.

- Start with a byte array of image data which is already downsampled to 1-bit monochrome; each byte conveys information on 1 pixel

1) Get a single row of 8 bytes (8 pixels):

Input byte No.:	0	1	2	3	4	5	6	7
Pixel value:	0	1	1	1	0	1	1	0

Table 6.1: Input data – 8 bytes

2) Merge the input byte values (numbering from 0 to 7) into one output byte, conveying information on 8 pixels

Input byte No.:	0	1	2	3	4	5	6	7
Pixel value:	0	1	1	1	0	1	1	0
Output byte value:	0x76 | 0b01110110							

Table 6.2: Output data – single byte

3) Go back to Step 1), getting the following row; repeat until all the bytes are processed

Sample Code

Below is sample Java code for image conversion:

```
static byte[] convertTo1bit_PixelFormatType0(byte[] picData, int w, int h)
{
   byte[] newRow = new byte[picData.length * 1 / 8];
   // join nibbles (so 1 byte is 8 pixels)
   int j = 0;
   for (int i = 0; i < picData.length; i += 8)
   {
          newRow[j] = (byte)( ((picData[i + 0] << 7) & 0x80) |
                              ((picData[i + 1] << 6) & 0x40) |
                              ((picData[i + 2] << 5) & 0x20) |
                              ((picData[i + 3] << 4) & 0x10) |
                              ((picData[i + 4] << 3) & 0x08) |
                              ((picData[i + 5] << 2) & 0x04) |
                              ((picData[i + 6] << 1) & 0x02) |
                              ((picData[i + 7])      & 0x01));
          j++;
   }
   return newRow;
}
```

6.2.2 Pixel Data Format Type 2

This format is used in TC-P441-230.

Conversion Algorithm

The algorithm for conversion from standard RAW 4-bit data to EPD format is described below.

- Start with a byte array of image data which is already downsampled to 1-bit monochrome; each byte conveys information on 1 pixel

4) Get a single row of 8 bytes (8 pixels):

Input byte No.:	0	1	2	3	4	5	6	7
Pixel value:	0	1	1	1	0	1	1	0

Table 6.3: Input data – 8 bytes

5) Assign the input byte values (numbering from 0 to 7) to the output byte, conveying information on 8 pixels, as follows:
   - Input byte 0: assign to output byte bit 0
   - Input byte 1: assign to output byte bit 2
   - Input byte 2: assign to output byte bit 4
   - Input byte 3: assign to output byte bit 6
   - Input byte 4: assign to output byte bit 1
   - Input byte 5: assign to output byte bit 3
   - Input byte 6: assign to output byte bit 5
   - Input byte 7: assign to output byte bit 7

Output byte bit No.:	0	1	2	3	4	5	6	7
Input byte No.:	0	4	1	5	2	6	3	7
Pixel value:	0	0	1	1	1	1	1	0
Output byte value:	0x3E | 0b00111110							

Table 6.4: Output data – single byte

6) Go back to Step 1), getting the following row; repeat until all the bytes are processed

Sample Code

Below is sample Java code for image conversion:

```
static byte[] convertTo1bit_PixelFormatType2(byte[] picData, int w, int h)
{
   byte[] newRow = new byte[picData.length * 1 / 8];
   // join nibbles (so 1 byte is 8 pixels) and interlace at the same time
   int j = 0;
   for (int i = 0; i < picData.length; i += 8)
   {
          newRow[j] = (byte)( ((picData[i + 0] << 7) & 0x80) |
                              ((picData[i + 4] << 6) & 0x40) |
                              ((picData[i + 1] << 5) & 0x20) |
                              ((picData[i + 5] << 4) & 0x10) |
                              ((picData[i + 2] << 3) & 0x08) |
                              ((picData[i + 6] << 2) & 0x04) |
                              ((picData[i + 3] << 1) & 0x02) |
                              ((picData[i + 7])      & 0x01));
          j++;
   }
   return newRow;
}
```

6.2.3 Pixel Data Format Type 4

This format is used in TC-P74-230.

Conversion Algorithm

The algorithm for conversion from standard RAW 4-bit data to EPD format is described below.

- Start with a byte array of image data which is already downsampled to 1-bit monochrome; each byte conveys information on 1 pixel

1) Get a single row of 480 bytes (480 pixels) – the *Bytes value* represent the value of 8 consecutive bytes if merged into one byte:

Input Byte No.:	0÷7	8÷15	16÷23	24÷31	...	...	...	...	472÷479
Bytes value:	0x76	0x4C	0xA3	0x1F	...	...	...	...	...

Table 6.5: Input data – 60 bytes (480 pixels)

2) Get first 16 bytes from that row:

Input Byte No.:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Pixel value:	0	1	1	1	0	1	1	0	0	1	0	0	1	1	0	0
Bytes value:	0x76 | 0b01110110								0x4C | 0b01001100							

Table 6.6: Input data – first 16 bytes

3) Create a 2-byte Intermediate array:

Int. byte No.:	0								1							
Position:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Input byte No.:																
Pixel value:																
Int. byte value:																

Table 6.7: Intermediate array – empty

4) Assign the Input Byte values (numbering from 0 to 15) to Intermediate array, conveying information on 16 pixels, as follows:
   - Input byte 0: assign to Intermediate array position 6
   - Input byte 1: assign to Intermediate array position 8
   - Input byte 2: assign to Intermediate array position 4
   - Input byte 3: assign to Intermediate array position 10
   - Input byte 4: assign to Intermediate array position 2
   - Input byte 5: assign to Intermediate array position 12
   - Input byte 6: assign to Intermediate array position 0
   - Input byte 7: assign to Intermediate array position 14
   - Input byte 8: assign to Intermediate array position 7
   - Input byte 9: assign to Intermediate array position 9
   - Input byte 10: assign to Intermediate array position 5
   - Input byte 11: assign to Intermediate array position 11
   - Input byte 12: assign to Intermediate array position 3
   - Input byte 13: assign to Intermediate array position 13
   - Input byte 14: assign to Intermediate array position 1
   - Input byte 15: assign to Intermediate array position 15

Int. byte No.:	0								1							
Position:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Input Byte No.:	6	14	4	12	2	10	0	8	1	9	3	11	5	13	7	15
Pixel value:	1	0	0	1	1	0	0	0	1	1	1	0	1	1	0	0
Int. byte value:	0x98 | 0b10011000								0xEC | 0b11101100							

Table 6.8: Intermediate array – first 16 bytes processed

5) Create an output array of 60 bytes
6) Assign Intermediate byte 0 (the one with even pixels) to position 29 of the Output Array;
   Assign Intermediate byte 1 (the one with odd pixels) to position 59 of the Output Array

Output Byte No.:	0	1	…	…	…	…	28	29	30	31	…	…	…	…	58	59
Int. byte No.:								0								1
Output Byte value:								0x98								0xEC

Table 6.9: Output data filled in with first two bytes

7) Go back to Step 2), getting the following 16 bytes from the row (bytes 16÷31)

Input Byte No.:	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Bytes value:	0xA3 | 0b10100011								0x1F | 0b00011111							

Table 6.10: Input data – following 16 bytes

- Assign the bytes as in Step 4)

Int. byte No.:	2								3							
Int. byte value:	0xDA | 0b11011010								0x17 | 0b00010111							

Table 6.11: Intermediate data – following 16 bytes processed

- Assign Intermediate byte 2 to position 28 of the Output Array
  Assign Intermediate byte 3 to position 58 of the Output Array

Output Byte No.:	0	1	…	…	…	…	28	29	30	31	…	…	…	…	58	59
Int. byte No.:							2	0							3	1
Output Byte value:							0xDA	0x98							0x17	0xEC

Table 6.12: Output data filled in with following two bytes

8) Repeat until the full 480-byte row is processed
9) Get the following 480-byte row and repeat the process until the whole image is processed

Sample Code

Below is sample Java code for image conversion:

```
static byte[] convertTo1bit_PixelFormatType4(byte[] picData, int w, int h)
{
   byte[] newPicData = new byte[picData.length / 8];
   int row = 30, s = 1;
   for (i = 0; i < picData.length; i += 16)
   {
          newPicData[row-s] = (byte)(       ((picData[i + 6 ] << 7) & 0x80) |
                                            ((picData[i + 14] << 6) & 0x40) |
                                            ((picData[i + 4 ] << 5) & 0x20) |
```

```
                                        ((picData[i + 12] << 4) & 0x10) |
                                        ((picData[i + 2 ] << 3) & 0x08) |
                                        ((picData[i + 10] << 2) & 0x04) |
                                        ((picData[i + 0 ] << 1) & 0x02) |
                                        ((picData[i + 8 ] << 0) & 0x01));

        newPicData[row+30-s] = (byte) (   ((picData[i + 1 ] << 7) & 0x80) |
                                        ((picData[i + 9 ] << 6) & 0x40) |
                                        ((picData[i + 3 ] << 5) & 0x20) |
                                        ((picData[i + 11] << 4) & 0x10) |
                                        ((picData[i + 5 ] << 3) & 0x08) |
                                        ((picData[i + 13] << 2) & 0x04) |
                                        ((picData[i + 7 ] << 1) & 0x02) |
                                        ((picData[i + 15] << 0) & 0x01));
        s++;
        if(s==31)
        {
               s=1;
               row+=60;
        }
   }
   return newPicData;
}
```