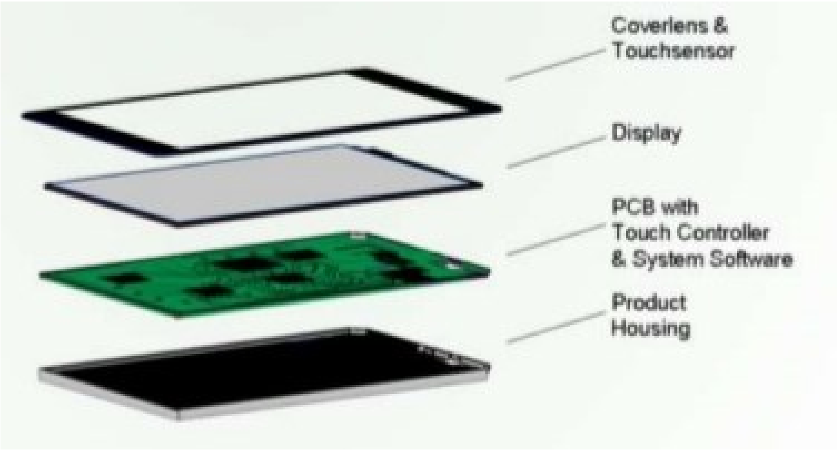
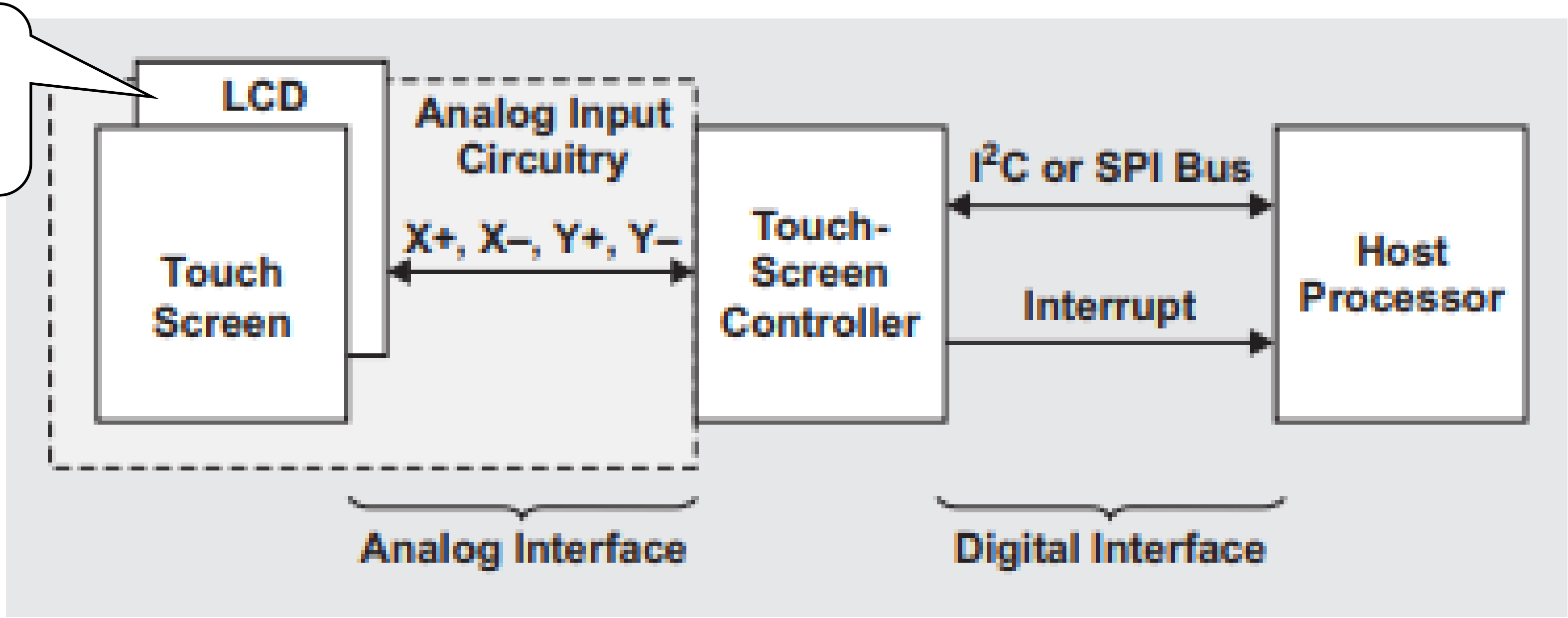
Touch Display

Paolo Bernardi

How Touch Screens work?



Display controller
(Chip-on-Glass)



Display types

- **LCD** – Liquid-Crystal-Display
- **TFT LCD** – Thin-Film-Transistor Liquid-Crystal Display
- **IPS LCD** – In-Plane Switching Liquid-Crystal Display
- **LED-backlit LCD** – Light-Emitting Diodes Liquid-Crystal Display

Constructed of flat panels that contain liquid crystals with light modulating properties.
This means that these liquid crystals use a backlight or reflector to emit light and produce either monochromatic or colored images.

Is a variant of a liquid-crystal display (LCD) that uses thin-film-transistor (TFT) technology to improve image qualities such as addressability and contrast. TFT-based displays have a transistor for each pixel on the screen.

They offer better viewing angles and consume less power. It is more expensive.

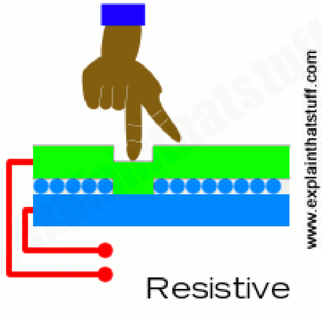
For TVs mainly, LED LCDs use an array of smaller, more efficient light-emitting diodes (LEDs) to illuminate the screen.
All LCD TVs now use LED lights and are colloquially called LED TVs.

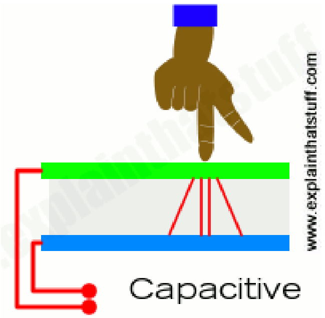
Touch Screens types

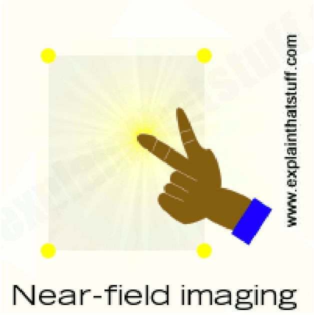
- Resistive
- Capacitive
- Infrared
- Surface Acoustic Wave
- Near field imaging
- Light pens

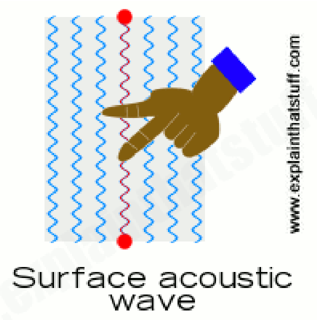
Resistive touchscreens (currently the most popular technology) work a bit like "transparent keyboards" overlaid on top of the screen.

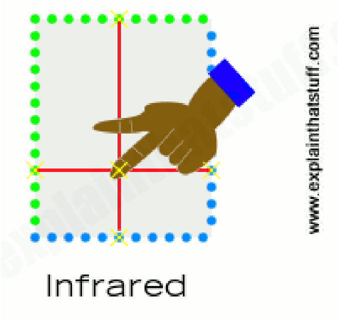
It is made from multiple layers of glass. The inner layer conducts electricity and so does the outer layer, the screen behaves like two conductors separated by an insulator—in other words, a capacitor. Capacitive screens can be touched in more than one place at once.



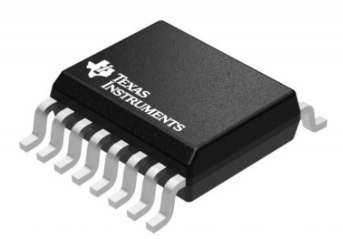


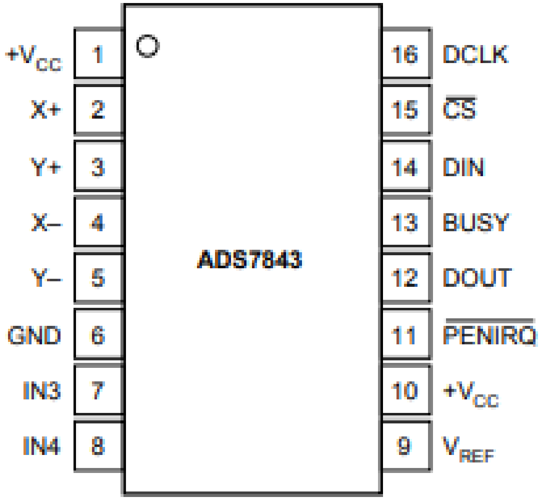


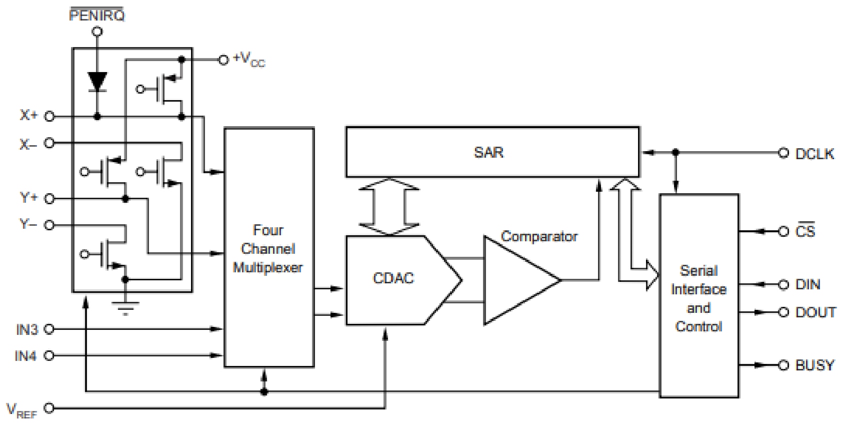


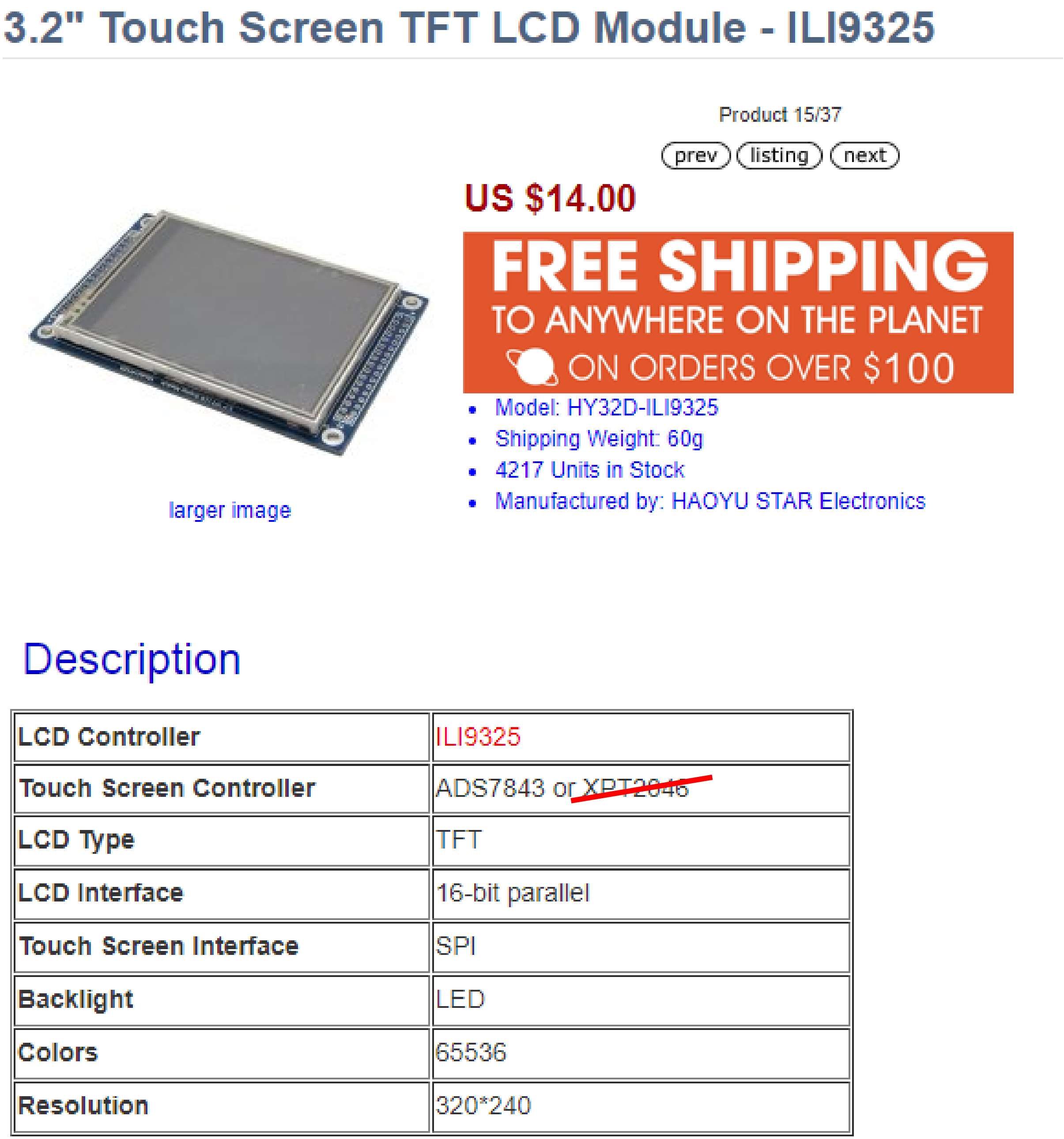


Our Hardware









3.2" Touch Screen TFT LCD Module - ILI9325

Product 15/37

larger image

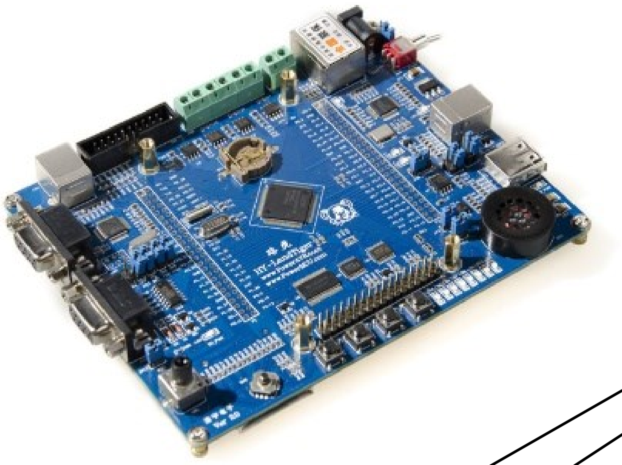
US $14.00

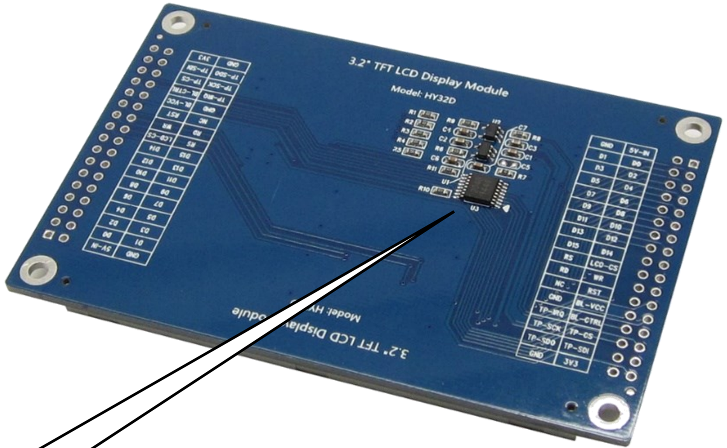
- Model: HY32D-ILI9325
- Shipping Weight: 60g
- 4217 Units in Stock
- Manufactured by: HAOYU STAR Electronics

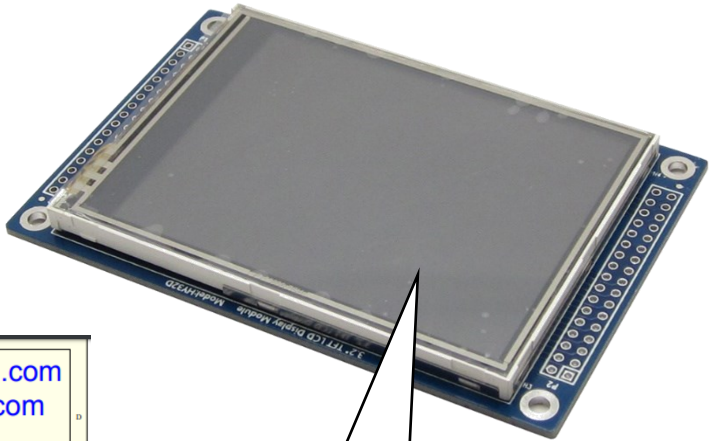
Description

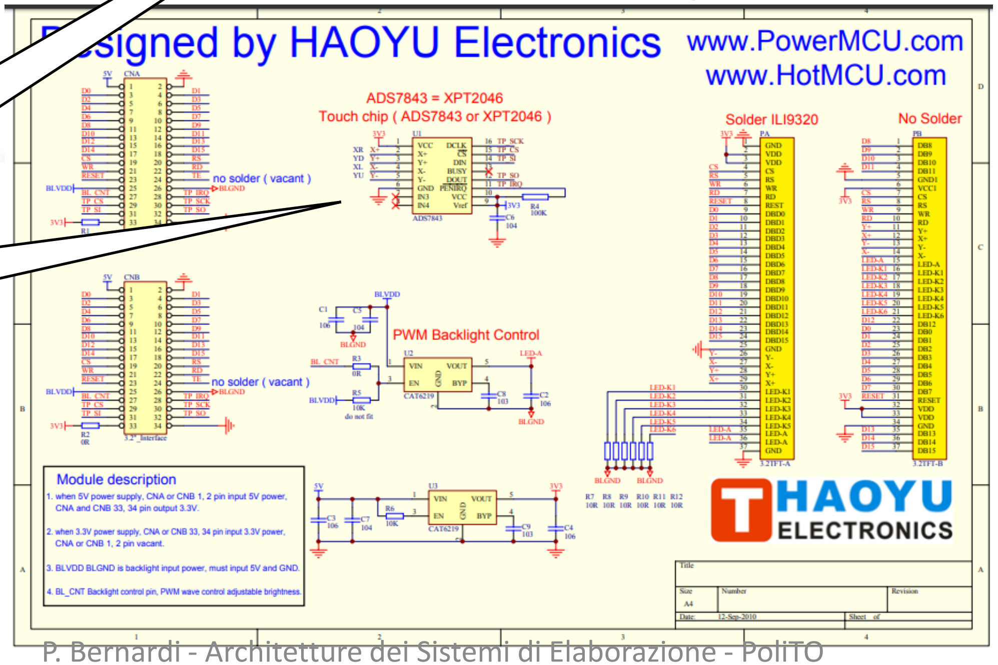
LCD Controller	ILI9325
Touch Screen Controller	ADS7843 or ~~XPT2046~~
LCD Type	TFT
LCD Interface	16-bit parallel
Touch Screen Interface	SPI
Backlight	LED
Colors	65536
Resolution	320*240

3.2" Touch Screen TFT LCD Module









Module description

1. when 5V power supply, CNA or CNB 1, 2 pin input 5V power, CNA and CNB 33, 34 pin output 3.3V.
2. when 3.3V power supply, CNA or CNB 33, 34 pin input 3.3V power, CNA or CNB 1, 2 pin vacant.
3. BLVDD BLGND is backlight input power, must input 5V and GND.
4. BL_CNT Backlight control pin, PWM wave control adjustable brightness.

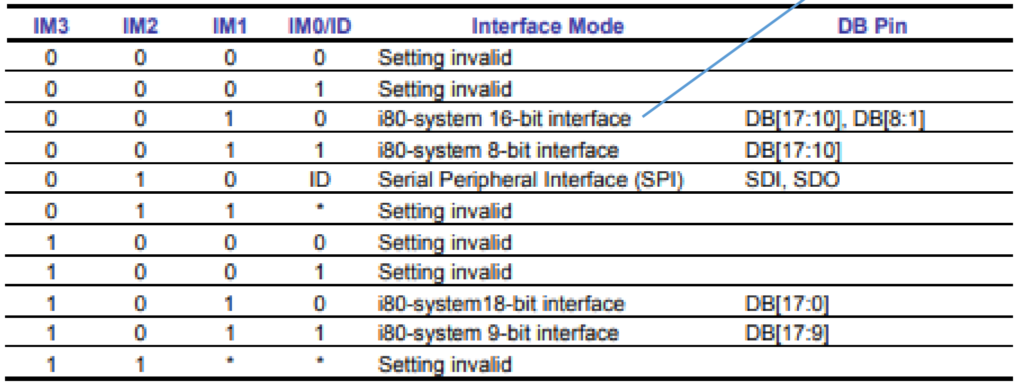
Touch Screen controller **ADS7843**

Display controller **ILI9325** (Chip-on-Glass)

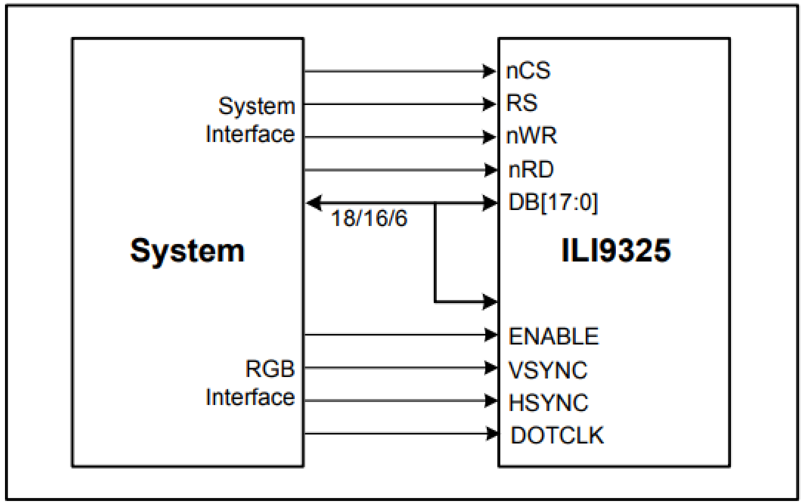
Display controller ILI9325

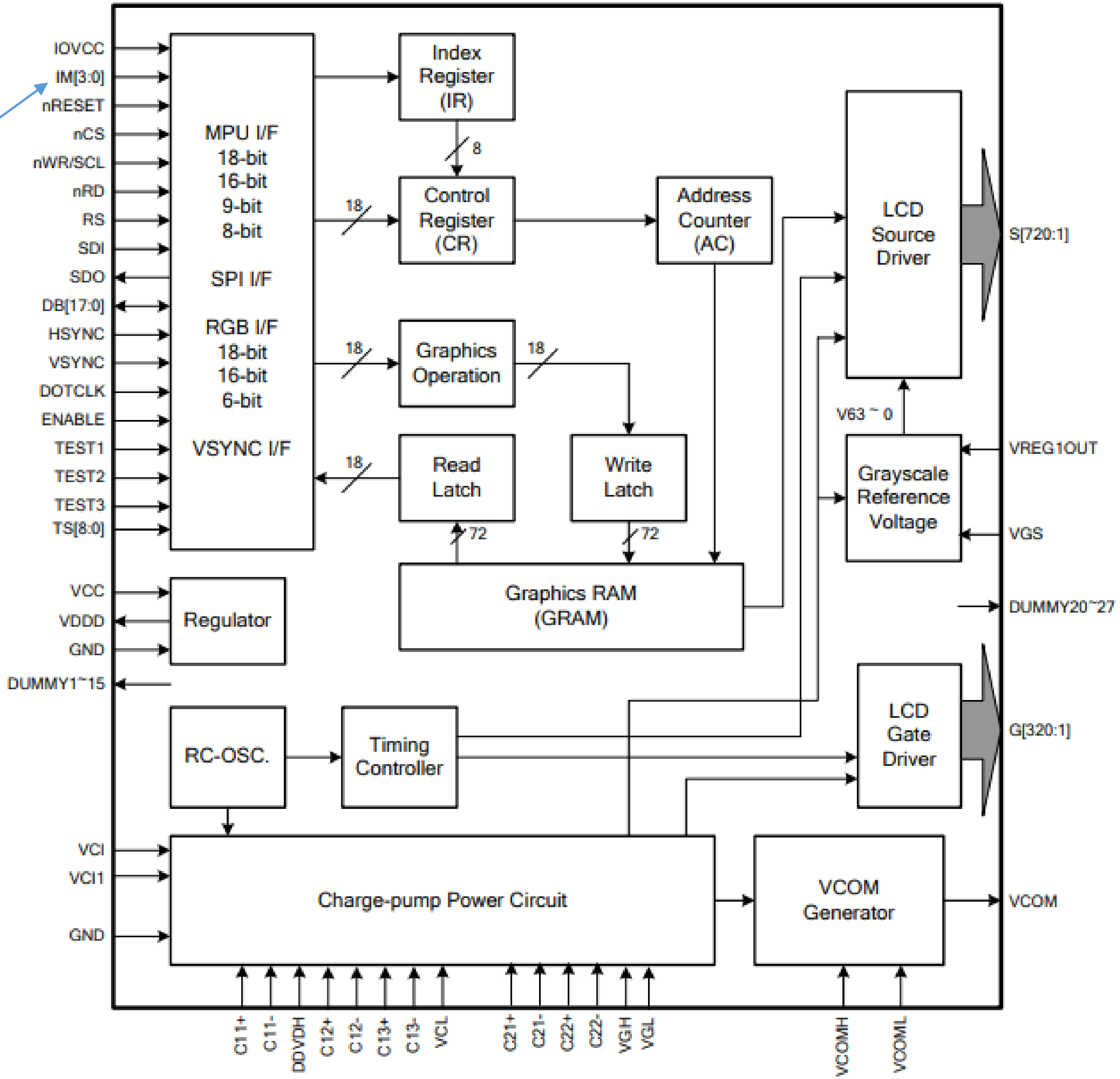
- Single chip solution for a TFT LCD display
- 240RGBx320-dot resolution capable with real 262,144 display color
- Incorporates 720-channel source driver and 320-channel gate driver
- Internal 172,800 bytes graphic RAM
- High-speed RAM burst write function
- System interfaces
- i80 system interface with 8-/ 9-/16-/18-bit bus width
- Serial Peripheral Interface (SPI)
- RGB interface with 6-/16-/18-bit bus width (VSYNC, HSYNC, DOTCLK, ENABLE, DB[17:0])
- VSYNC interface (System interface + VSYNC)

Block Diagram MPU side

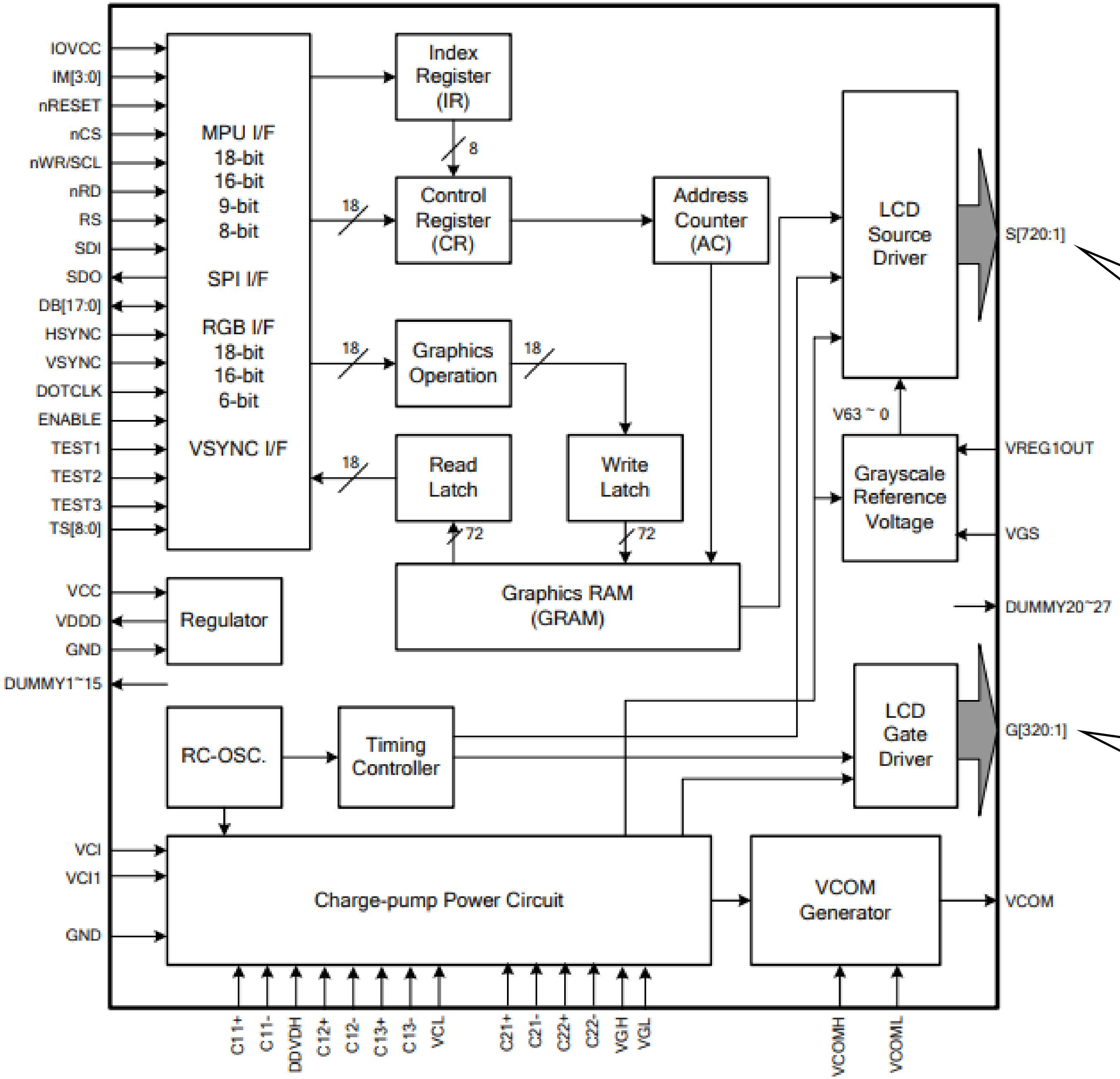


IM3	IM2	IM1	IM0/ID	Interface Mode	DB Pin
0	0	0	0	Setting invalid	
0	0	0	1	Setting invalid	
0	0	1	0	i80-system 16-bit interface	DB[17:10], DB[8:1]
0	0	1	1	i80-system 8-bit interface	DB[17:10]
0	1	0	ID	Serial Peripheral Interface (SPI)	SDI, SDO
0	1	1	*	Setting invalid	
1	0	0	0	Setting invalid	
1	0	0	1	Setting invalid	
1	0	1	0	i80-system18-bit interface	DB[17:0]
1	0	1	1	i80-system 9-bit interface	DB[17:9]
1	1	*	*	Setting invalid	

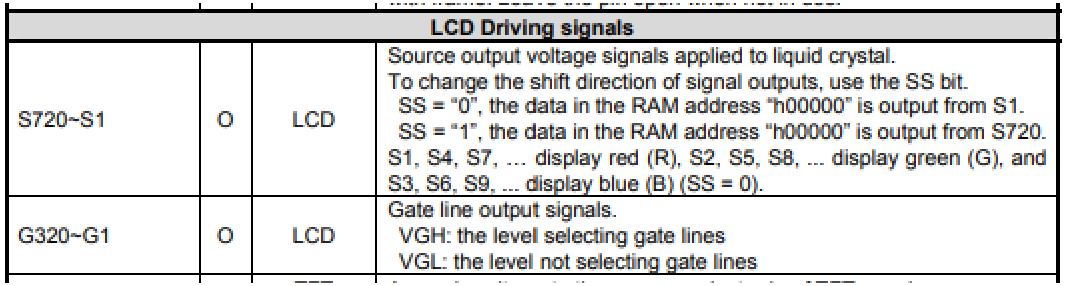




Block Diagram
Display side



Provides 3 analog values for every cell in a row (240 pixel)

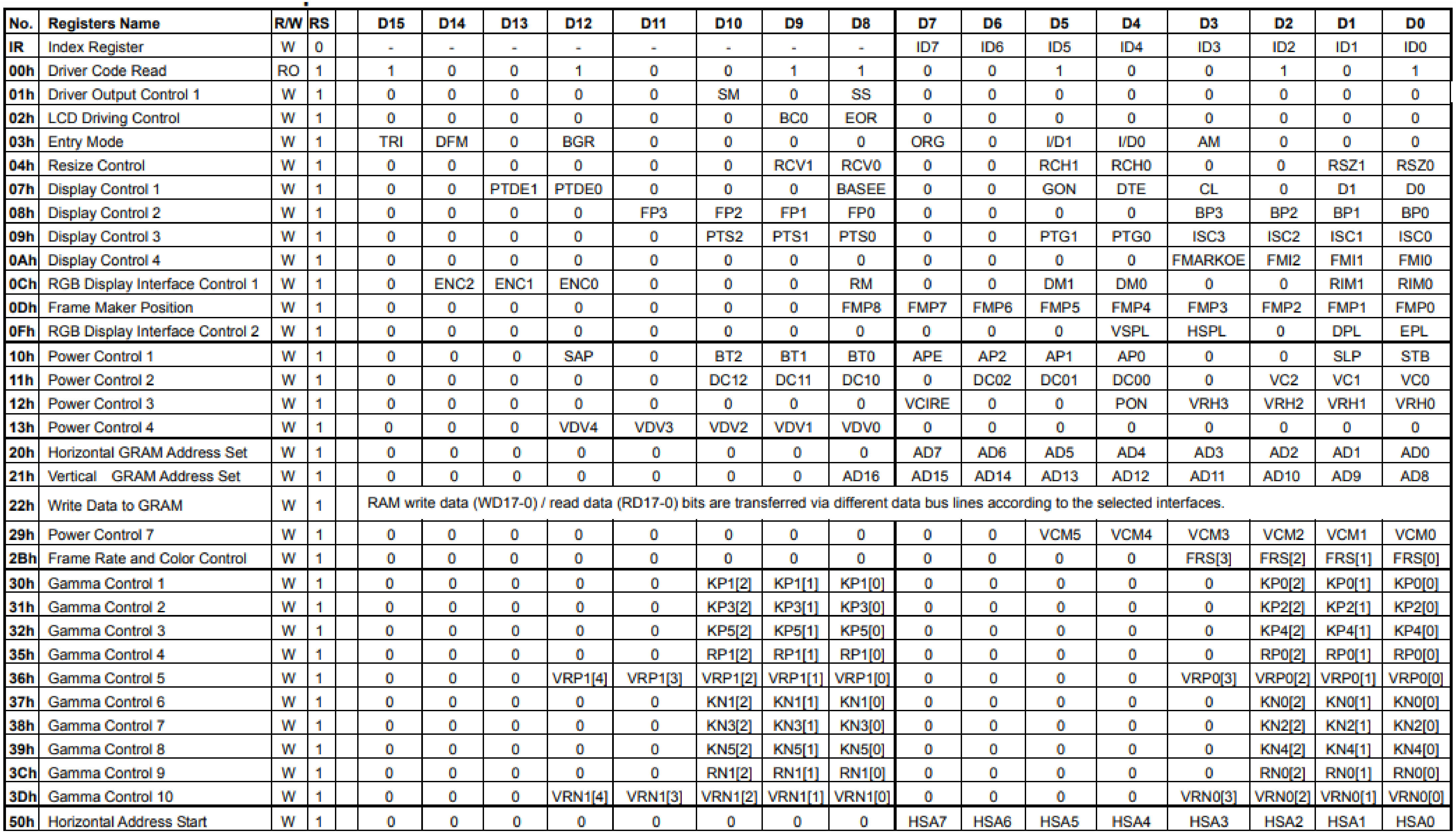


LCD Driving signals			
S720~S1	O	LCD	Source output voltage signals applied to liquid crystal. To change the shift direction of signal outputs, use the SS bit. SS = "0", the data in the RAM address "h00000" is output from S1. SS = "1", the data in the RAM address "h00000" is output from S720. S1, S4, S7, ... display red (R), S2, S5, S8, ... display green (G), and S3, S6, S9, ... display blue (B) (SS = 0).
G320~G1	O	LCD	Gate line output signals. VGH: the level selecting gate lines VGL: the level not selecting gate lines

Selection of the line to be displayed

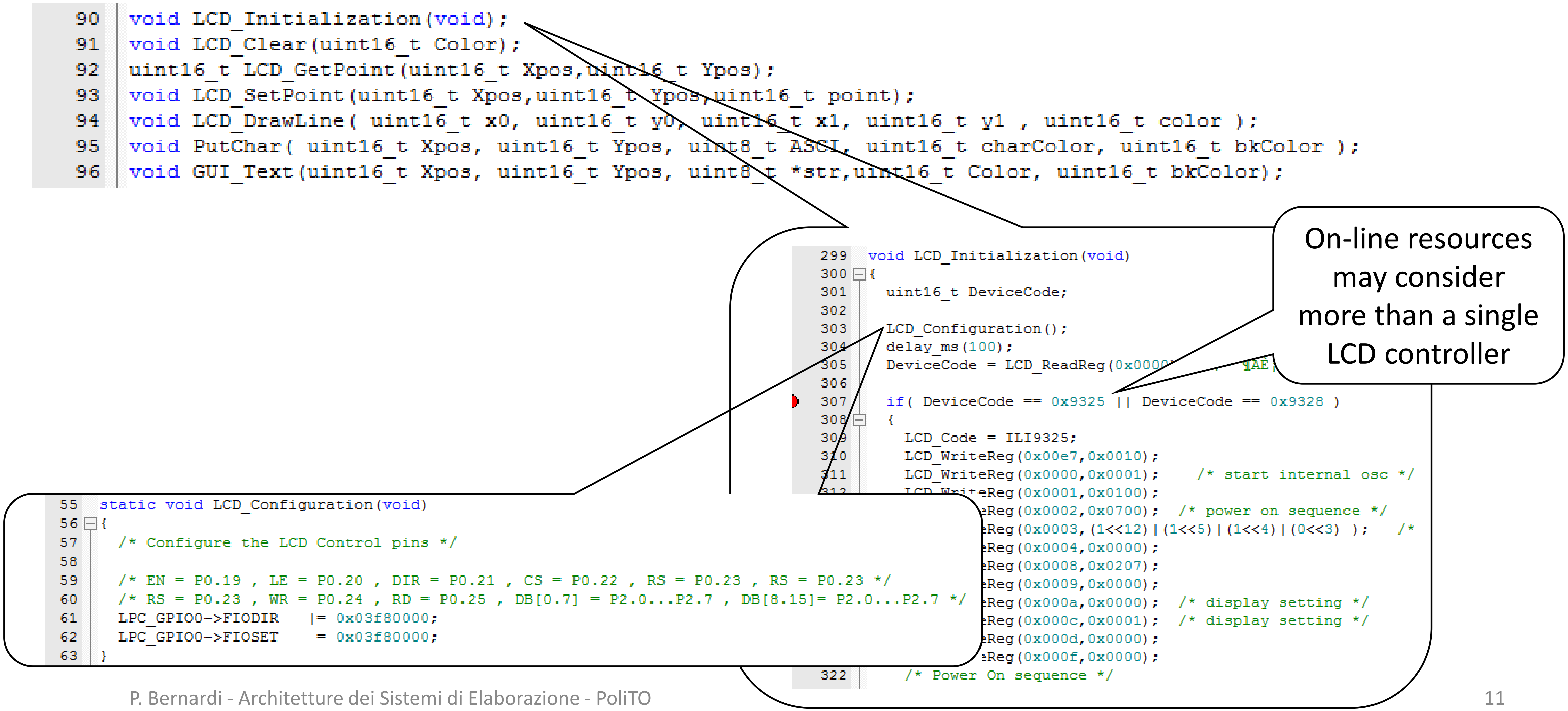
Updates a line at a time

Registers



No.	Registers Name	R/W	RS		D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
IR	Index Register	W	0		-	-	-	-	-	-	-	-	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
00h	Driver Code Read	RO	1		1	0	0	1	0	0	1	1	0	0	1	0	0	1	0	1
01h	Driver Output Control 1	W	1		0	0	0	0	0	SM	0	SS	0	0	0	0	0	0	0	0
02h	LCD Driving Control	W	1		0	0	0	0	0	0	BC0	EOR	0	0	0	0	0	0	0	0
03h	Entry Mode	W	1		TRI	DFM	0	BGR	0	0	0	0	ORG	0	I/D1	I/D0	AM	0	0	0
04h	Resize Control	W	1		0	0	0	0	0	0	RCV1	RCV0	0	0	RCH1	RCH0	0	0	RSZ1	RSZ0
07h	Display Control 1	W	1		0	0	PTDE1	PTDE0	0	0	0	BASEE	0	0	GON	DTE	CL	0	D1	D0
08h	Display Control 2	W	1		0	0	0	0	FP3	FP2	FP1	FP0	0	0	0	0	BP3	BP2	BP1	BP0
09h	Display Control 3	W	1		0	0	0	0	0	PTS2	PTS1	PTS0	0	0	PTG1	PTG0	ISC3	ISC2	ISC1	ISC0
0Ah	Display Control 4	W	1		0	0	0	0	0	0	0	0	0	0	0	0	FMARKOE	FMI2	FMI1	FMI0
0Ch	RGB Display Interface Control 1	W	1		0	ENC2	ENC1	ENC0	0	0	0	RM	0	0	DM1	DM0	0	0	RIM1	RIM0
0Dh	Frame Maker Position	W	1		0	0	0	0	0	0	0	FMP8	FMP7	FMP6	FMP5	FMP4	FMP3	FMP2	FMP1	FMP0
0Fh	RGB Display Interface Control 2	W	1		0	0	0	0	0	0	0	0	0	0	0	VSPL	HSPL	0	DPL	EPL
10h	Power Control 1	W	1		0	0	0	SAP	0	BT2	BT1	BT0	APE	AP2	AP1	AP0	0	0	SLP	STB
11h	Power Control 2	W	1		0	0	0	0	0	DC12	DC11	DC10	0	DC02	DC01	DC00	0	VC2	VC1	VC0
12h	Power Control 3	W	1		0	0	0	0	0	0	0	0	VCIRE	0	0	PON	VRH3	VRH2	VRH1	VRH0
13h	Power Control 4	W	1		0	0	0	VDV4	VDV3	VDV2	VDV1	VDV0	0	0	0	0	0	0	0	0
20h	Horizontal GRAM Address Set	W	1		0	0	0	0	0	0	0	0	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
21h	Vertical GRAM Address Set	W	1		0	0	0	0	0	0	0	AD16	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8
22h	Write Data to GRAM	W	1		RAM write data (WD17-0) / read data (RD17-0) bits are transferred via different data bus lines according to the selected interfaces.															
29h	Power Control 7	W	1		0	0	0	0	0	0	0	0	0	0	VCM5	VCM4	VCM3	VCM2	VCM1	VCM0
2Bh	Frame Rate and Color Control	W	1		0	0	0	0	0	0	0	0	0	0	0	0	FRS[3]	FRS[2]	FRS[1]	FRS[0]
30h	Gamma Control 1	W	1		0	0	0	0	0	KP1[2]	KP1[1]	KP1[0]	0	0	0	0	0	KP0[2]	KP0[1]	KP0[0]
31h	Gamma Control 2	W	1		0	0	0	0	0	KP3[2]	KP3[1]	KP3[0]	0	0	0	0	0	KP2[2]	KP2[1]	KP2[0]
32h	Gamma Control 3	W	1		0	0	0	0	0	KP5[2]	KP5[1]	KP5[0]	0	0	0	0	0	KP4[2]	KP4[1]	KP4[0]
35h	Gamma Control 4	W	1		0	0	0	0	0	RP1[2]	RP1[1]	RP1[0]	0	0	0	0	0	RP0[2]	RP0[1]	RP0[0]
36h	Gamma Control 5	W	1		0	0	0	VRP1[4]	VRP1[3]	VRP1[2]	VRP1[1]	VRP1[0]	0	0	0	0	VRP0[3]	VRP0[2]	VRP0[1]	VRP0[0]
37h	Gamma Control 6	W	1		0	0	0	0	0	KN1[2]	KN1[1]	KN1[0]	0	0	0	0	0	KN0[2]	KN0[1]	KN0[0]
38h	Gamma Control 7	W	1		0	0	0	0	0	KN3[2]	KN3[1]	KN3[0]	0	0	0	0	0	KN2[2]	KN2[1]	KN2[0]
39h	Gamma Control 8	W	1		0	0	0	0	0	KN5[2]	KN5[1]	KN5[0]	0	0	0	0	0	KN4[2]	KN4[1]	KN4[0]
3Ch	Gamma Control 9	W	1		0	0	0	0	0	RN1[2]	RN1[1]	RN1[0]	0	0	0	0	0	RN0[2]	RN0[1]	RN0[0]
3Dh	Gamma Control 10	W	1		0	0	0	VRN1[4]	VRN1[3]	VRN1[2]	VRN1[1]	VRN1[0]	0	0	0	0	VRN0[3]	VRN0[2]	VRN0[1]	VRN0[0]
50h	Horizontal Address Start	W	1		0	0	0	0	0	0	0	0	HSA7	HSA6	HSA5	HSA4	HSA3	HSA2	HSA1	HSA0

Already available libraries and functions



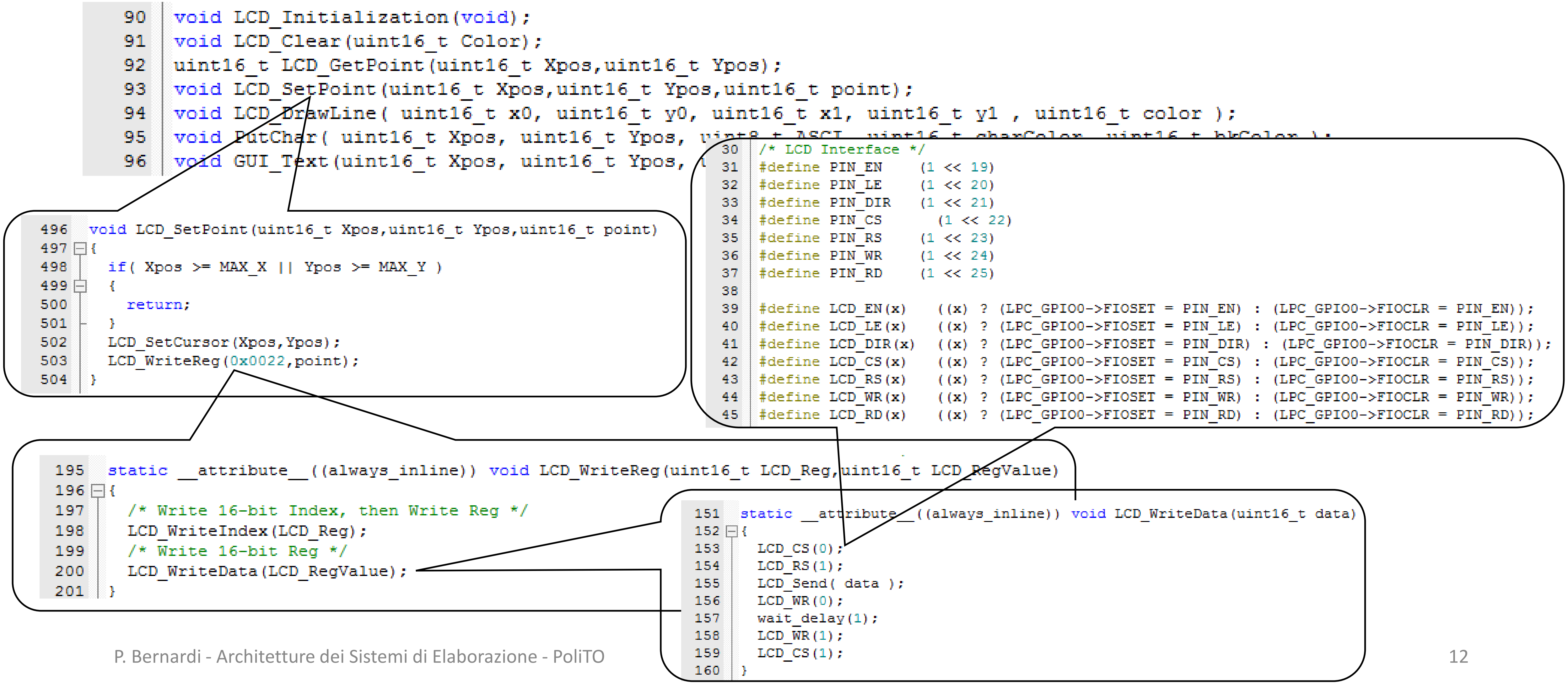
```
void LCD_Initialization(void);
void LCD_Clear(uint16_t Color);
uint16_t LCD_GetPoint(uint16_t Xpos,uint16_t Ypos);
void LCD_SetPoint(uint16_t Xpos,uint16_t Ypos,uint16_t point);
void LCD_DrawLine( uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1 , uint16_t color );
void PutChar( uint16_t Xpos, uint16_t Ypos, uint8_t ASCI, uint16_t charColor, uint16_t bkColor );
void GUI_Text(uint16_t Xpos, uint16_t Ypos, uint8_t *str,uint16_t Color, uint16_t bkColor);
```

```
void LCD_Initialization(void)
{
  uint16_t DeviceCode;

  LCD_Configuration();
  delay_ms(100);
  DeviceCode = LCD_ReadReg(0x0000                  ¶AE;

  if( DeviceCode == 0x9325 || DeviceCode == 0x9328 )
  {
    LCD_Code = ILI9325;
    LCD_WriteReg(0x00e7,0x0010);
    LCD_WriteReg(0x0000,0x0001);    /* start internal osc */
    LCD_WriteReg(0x0001,0x0100);
           eReg(0x0002,0x0700);  /* power on sequence */
           eReg(0x0003,(1<<12)|(1<<5)|(1<<4)|(0<<3) );   /*
           eReg(0x0004,0x0000);
           eReg(0x0008,0x0207);
           eReg(0x0009,0x0000);
           eReg(0x000a,0x0000);  /* display setting */
           eReg(0x000c,0x0001);  /* display setting */
           eReg(0x000d,0x0000);
           eReg(0x000f,0x0000);
    /* Power On sequence */
```

On-line resources may consider more than a single LCD controller

```
static void LCD_Configuration(void)
{
  /* Configure the LCD Control pins */

  /* EN = P0.19 , LE = P0.20 , DIR = P0.21 , CS = P0.22 , RS = P0.23 , RS = P0.23 */
  /* RS = P0.23 , WR = P0.24 , RD = P0.25 , DB[0.7] = P2.0...P2.7 , DB[8.15]= P2.0...P2.7 */
  LPC_GPIO0->FIODIR   |= 0x03f80000;
  LPC_GPIO0->FIOSET    = 0x03f80000;
}
```

Already available libraries and functions



```c
void LCD_Initialization(void);
void LCD_Clear(uint16_t Color);
uint16_t LCD_GetPoint(uint16_t Xpos,uint16_t Ypos);
void LCD_SetPoint(uint16_t Xpos,uint16_t Ypos,uint16_t point);
void LCD_DrawLine( uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1 , uint16_t color );
void PutChar( uint16_t Xpos, uint16_t Ypos, uint8_t ASCI, uint16_t charColor, uint16_t bkColor );
void GUI_Text(uint16_t Xpos, uint16_t Ypos,
```

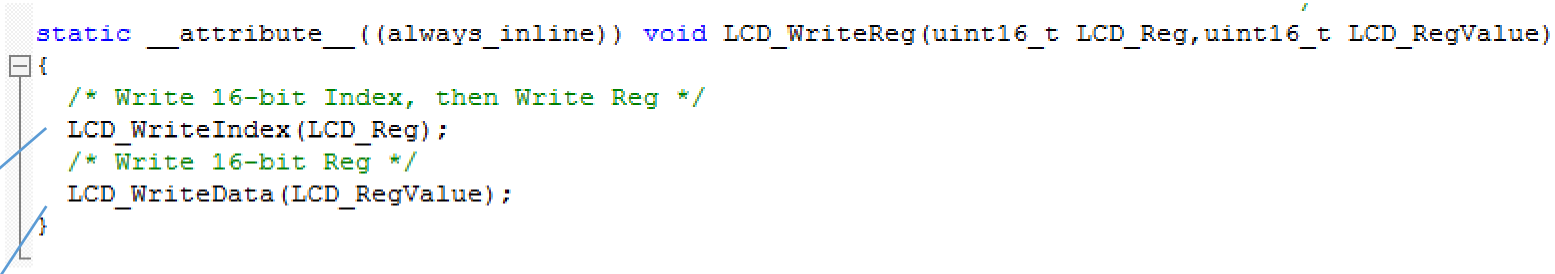
```c
void LCD_SetPoint(uint16_t Xpos,uint16_t Ypos,uint16_t point)
{
  if( Xpos >= MAX_X || Ypos >= MAX_Y )
  {
    return;
  }
  LCD_SetCursor(Xpos,Ypos);
  LCD_WriteReg(0x0022,point);
}
```

```c
/* LCD Interface */
#define PIN_EN    (1 << 19)
#define PIN_LE    (1 << 20)
#define PIN_DIR   (1 << 21)
#define PIN_CS      (1 << 22)
#define PIN_RS    (1 << 23)
#define PIN_WR    (1 << 24)
#define PIN_RD    (1 << 25)

#define LCD_EN(x)   ((x) ? (LPC_GPIO0->FIOSET = PIN_EN) : (LPC_GPIO0->FIOCLR = PIN_EN));
#define LCD_LE(x)   ((x) ? (LPC_GPIO0->FIOSET = PIN_LE) : (LPC_GPIO0->FIOCLR = PIN_LE));
#define LCD_DIR(x)  ((x) ? (LPC_GPIO0->FIOSET = PIN_DIR) : (LPC_GPIO0->FIOCLR = PIN_DIR));
#define LCD_CS(x)   ((x) ? (LPC_GPIO0->FIOSET = PIN_CS) : (LPC_GPIO0->FIOCLR = PIN_CS));
#define LCD_RS(x)   ((x) ? (LPC_GPIO0->FIOSET = PIN_RS) : (LPC_GPIO0->FIOCLR = PIN_RS));
#define LCD_WR(x)   ((x) ? (LPC_GPIO0->FIOSET = PIN_WR) : (LPC_GPIO0->FIOCLR = PIN_WR));
#define LCD_RD(x)   ((x) ? (LPC_GPIO0->FIOSET = PIN_RD) : (LPC_GPIO0->FIOCLR = PIN_RD));
```

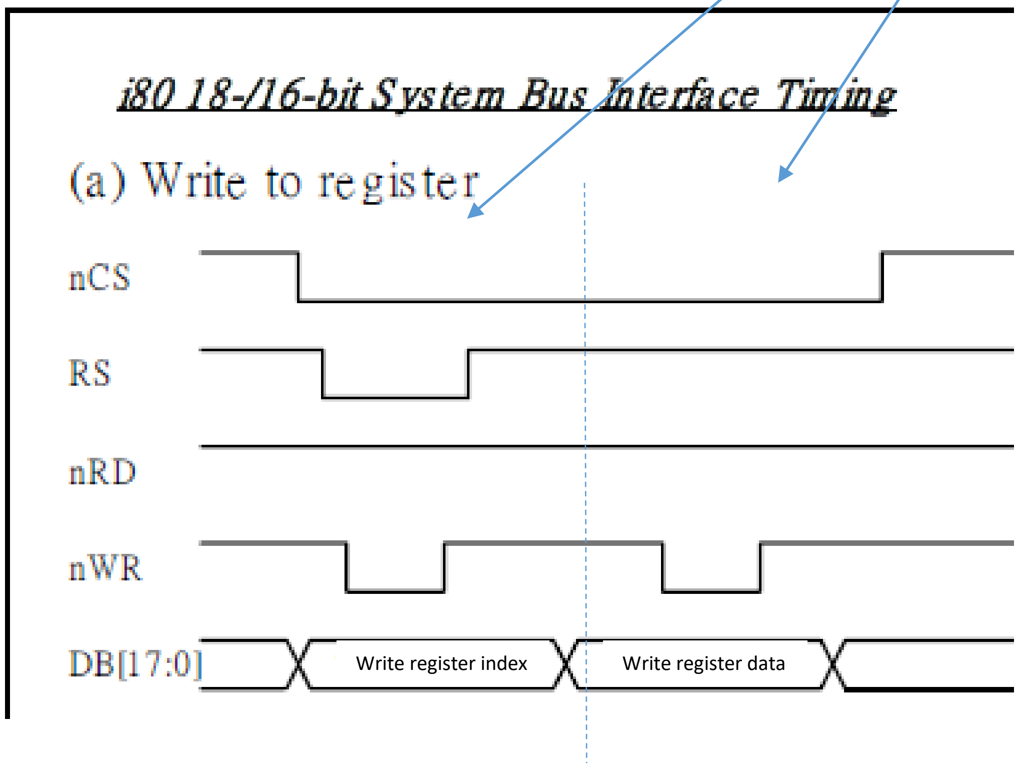
```c
static __attribute__((always_inline)) void LCD_WriteReg(uint16_t LCD_Reg,uint16_t LCD_RegValue)
{
  /* Write 16-bit Index, then Write Reg */
  LCD_WriteIndex(LCD_Reg);
  /* Write 16-bit Reg */
  LCD_WriteData(LCD_RegValue);
}
```

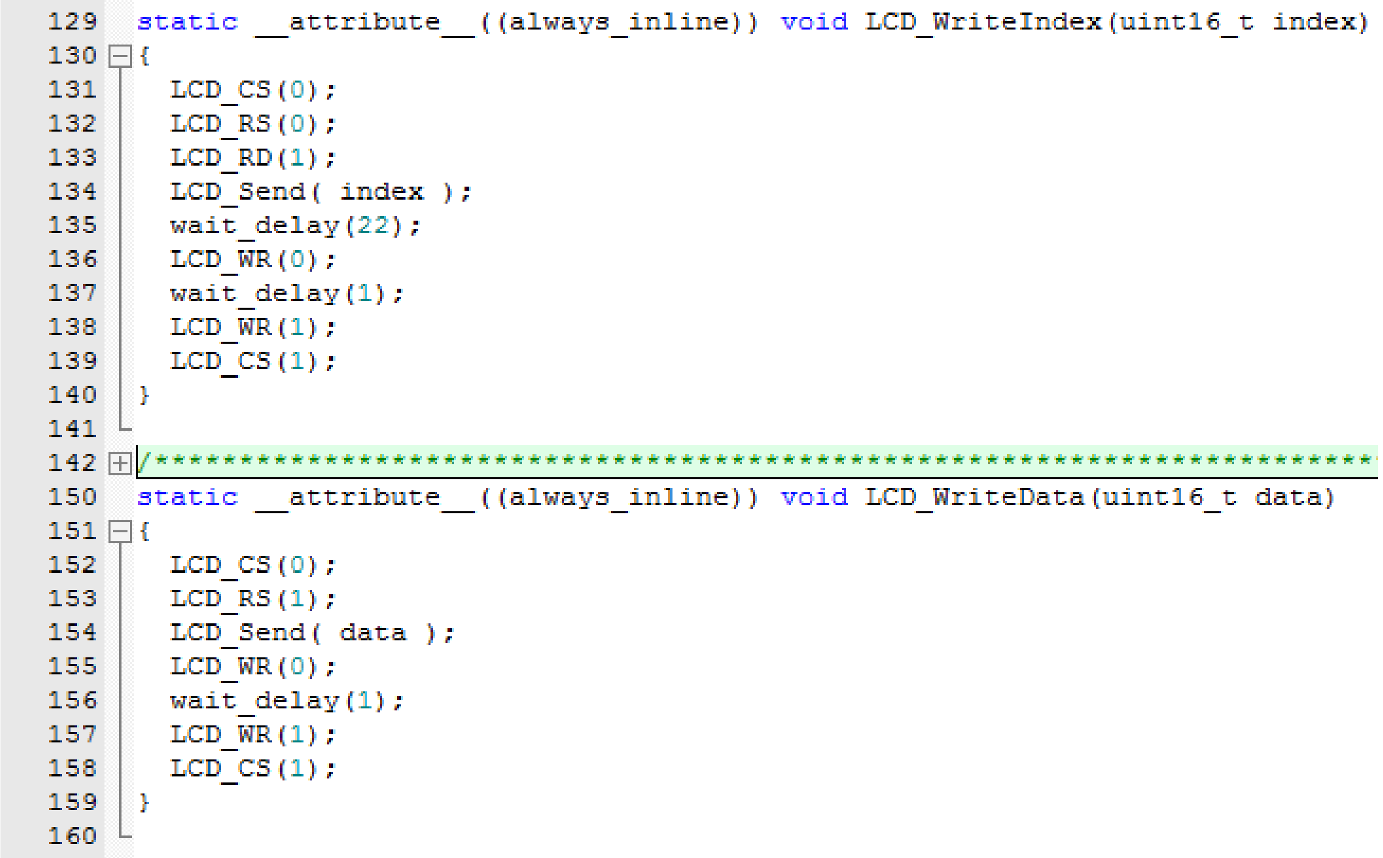
```c
static __attribute__((always_inline)) void LCD_WriteData(uint16_t data)
{
  LCD_CS(0);
  LCD_RS(1);
  LCD_Send( data );
  LCD_WR(0);
  wait_delay(1);
  LCD_WR(1);
  LCD_CS(1);
}
```

Communication timings



```
static __attribute__((always_inline)) void LCD_WriteReg(uint16_t LCD_Reg,uint16_t LCD_RegValue)
{
  /* Write 16-bit Index, then Write Reg */
  LCD_WriteIndex(LCD_Reg);
  /* Write 16-bit Reg */
  LCD_WriteData(LCD_RegValue);
}
```

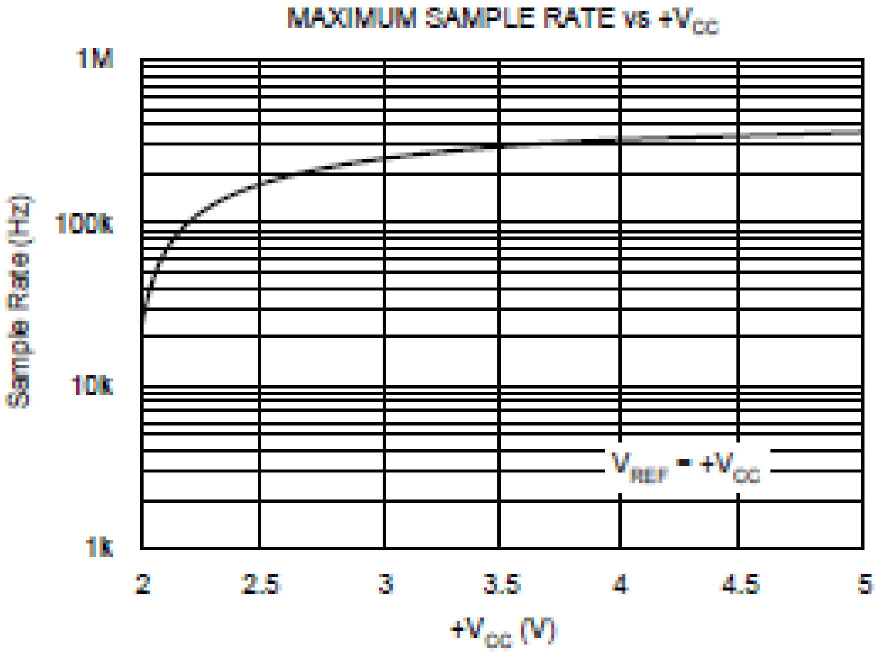




```
static __attribute__((always_inline)) void LCD_WriteIndex(uint16_t index)
{
  LCD_CS(0);
  LCD_RS(0);
  LCD_RD(1);
  LCD_Send( index );
  wait_delay(22);
  LCD_WR(0);
  wait_delay(1);
  LCD_WR(1);
  LCD_CS(1);
}

/**************************************************************************
static __attribute__((always_inline)) void LCD_WriteData(uint16_t data)
{
  LCD_CS(0);
  LCD_RS(1);
  LCD_Send( data );
  LCD_WR(0);
  wait_delay(1);
  LCD_WR(1);
  LCD_CS(1);
}
```

Touch Screen Controller - ADS7843

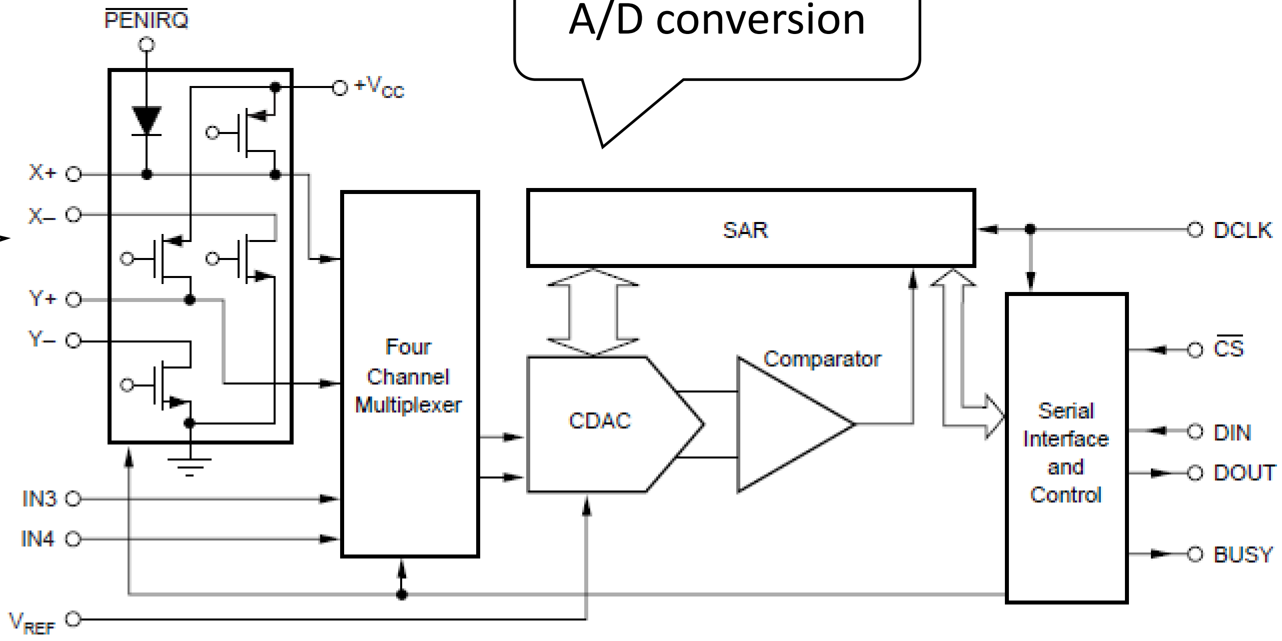
- 4-wire touch screen interface
- ratiometric conversion
- single supply: 2.7v to 5v
- up to 125kHz conversion rate
- serial interface
- programmable 8- or 12-bit resolution
- 2 auxiliary analog inputs
- full power-down control



Block diagram

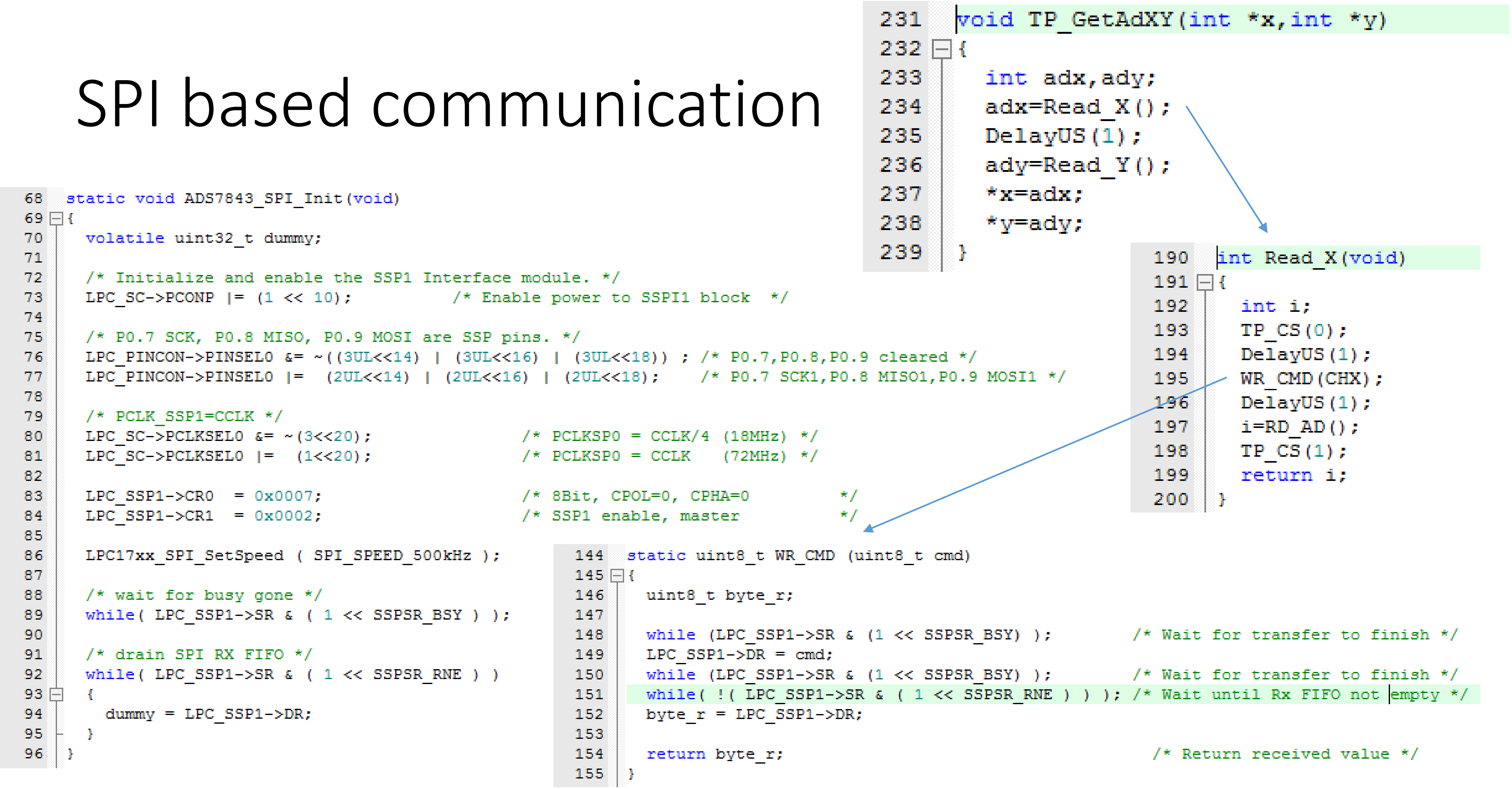
Touch panel side
Analog inputs

A/D conversion



MCU side
SPI protocol
Digital output

SPI based communication



```c
void TP_GetAdXY(int *x,int *y)
{
  int adx,ady;
  adx=Read_X();
  DelayUS(1);
  ady=Read_Y();
  *x=adx;
  *y=ady;
}
```

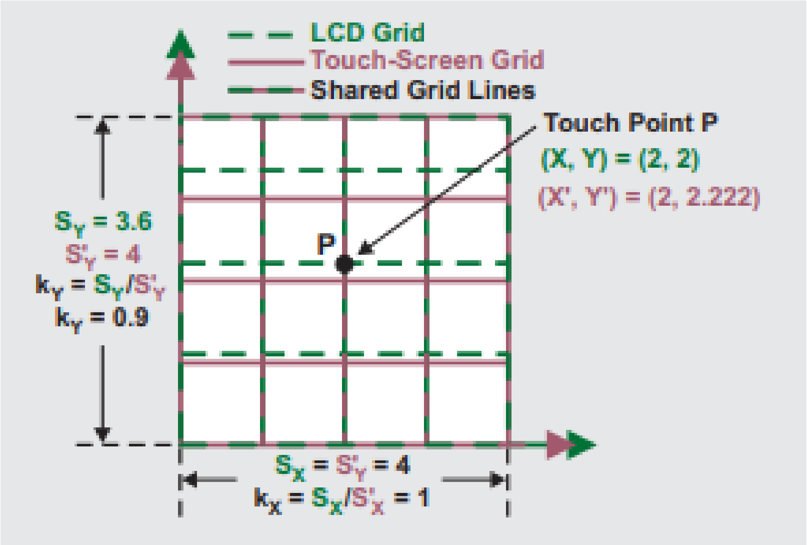
```c
int Read_X(void)
{
  int i;
  TP_CS(0);
  DelayUS(1);
  WR_CMD(CHX);
  DelayUS(1);
  i=RD_AD();
  TP_CS(1);
  return i;
}
```

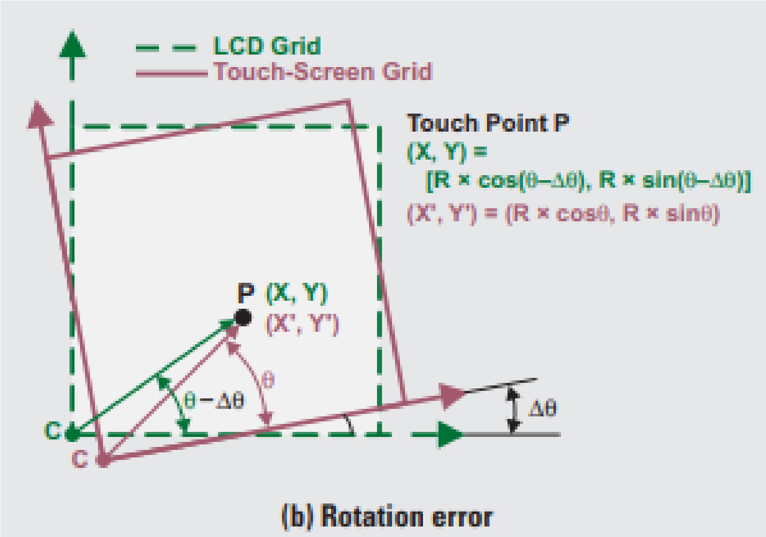
```c
static void ADS7843_SPI_Init(void)
{
  volatile uint32_t dummy;

  /* Initialize and enable the SSP1 Interface module. */
  LPC_SC->PCONP |= (1 << 10);          /* Enable power to SSPI1 block  */

  /* P0.7 SCK, P0.8 MISO, P0.9 MOSI are SSP pins. */
  LPC_PINCON->PINSEL0 &= ~((3UL<<14) | (3UL<<16) | (3UL<<18)) ; /* P0.7,P0.8,P0.9 cleared */
  LPC_PINCON->PINSEL0 |=  (2UL<<14) | (2UL<<16) | (2UL<<18);    /* P0.7 SCK1,P0.8 MISO1,P0.9 MOSI1 */

  /* PCLK_SSP1=CCLK */
  LPC_SC->PCLKSEL0 &= ~(3<<20);               /* PCLKSP0 = CCLK/4 (18MHz) */
  LPC_SC->PCLKSEL0 |=  (1<<20);               /* PCLKSP0 = CCLK   (72MHz) */

  LPC_SSP1->CR0  = 0x0007;                    /* 8Bit, CPOL=0, CPHA=0         */
  LPC_SSP1->CR1  = 0x0002;                    /* SSP1 enable, master          */

  LPC17xx_SPI_SetSpeed ( SPI_SPEED_500kHz );

  /* wait for busy gone */
  while( LPC_SSP1->SR & ( 1 << SSPSR_BSY ) );

  /* drain SPI RX FIFO */
  while( LPC_SSP1->SR & ( 1 << SSPSR_RNE ) )
  {
    dummy = LPC_SSP1->DR;
  }
}
```

```c
static uint8_t WR_CMD (uint8_t cmd)
{
  uint8_t byte_r;

  while (LPC_SSP1->SR & (1 << SSPSR_BSY) );        /* Wait for transfer to finish */
  LPC_SSP1->DR = cmd;
  while (LPC_SSP1->SR & (1 << SSPSR_BSY) );        /* Wait for transfer to finish */
  while( !( LPC_SSP1->SR & ( 1 << SSPSR_RNE ) ) ); /* Wait until Rx FIFO not empty */
  byte_r = LPC_SSP1->DR;

  return byte_r;                                    /* Return received value */
}
```

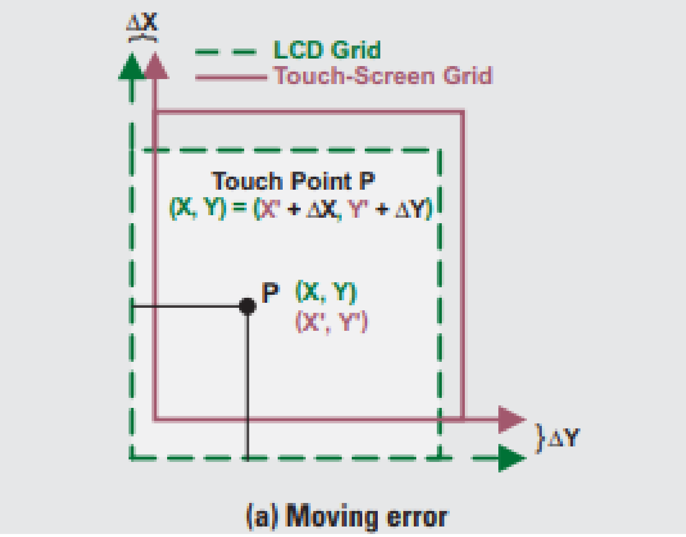
Touch-coordinate errors

- When pressure is applied to the touch screen, the touchscreen controller senses it and takes a measurement of the X and Y coordinates.
- Several sources of error can affect the accuracy and reliability of this measurement.





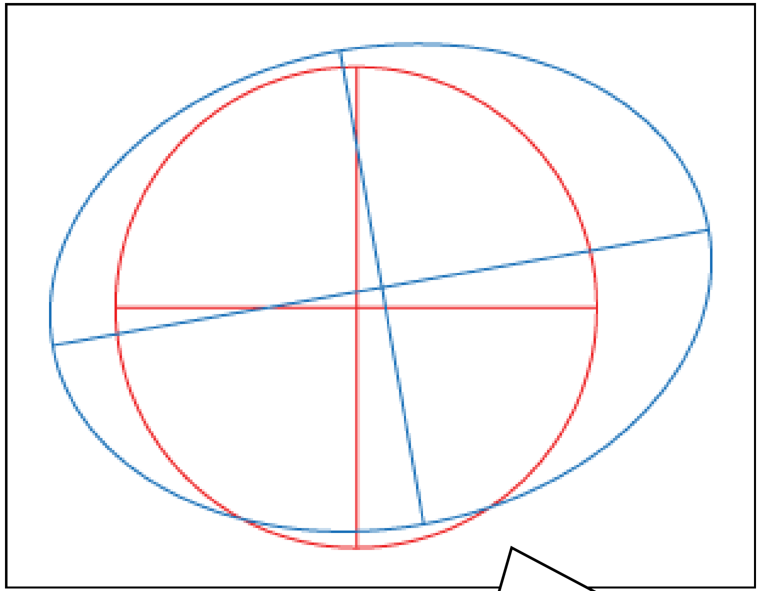
(b) Rotation error



(a) Moving error

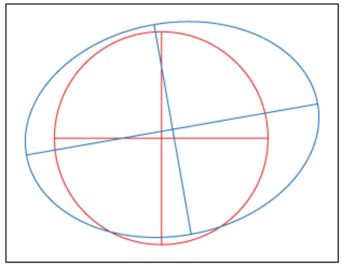
Calibration process

- When a finger is traced around the display circle (red line), the touch-panel system may give out the coordinates of an ellipse (blue line) instead of the circle
- This change of the shape from a circle to an ellipse can be explained by the following graphic transformations: translation, rotation, and scaling.



An exaggerated view of the distortion that might happen to a circle being displayed on an LCD touch-screen display

Calibration process (II)



- Intuition suggests that any coordinate point x, y in an x-y plane that has undergone a transformation should look like:

$$x_{NEW} = f1(x_{OLD}, y_{OLD}) + constant1$$

$$y_{NEW} = f2(x_{OLD}, y_{OLD}) + constant2$$

- where x_{NEW} and y_{NEW} are the transformed coordinates; x_{OLD} and y_{OLD} are the old coordinates;
  - f1() and f2() are functions that transform the old coordinates;
  - constants1 and 2 are just constants.

- If the transformation is linear, then functions f1() and f2() can be replaced by the following equations:

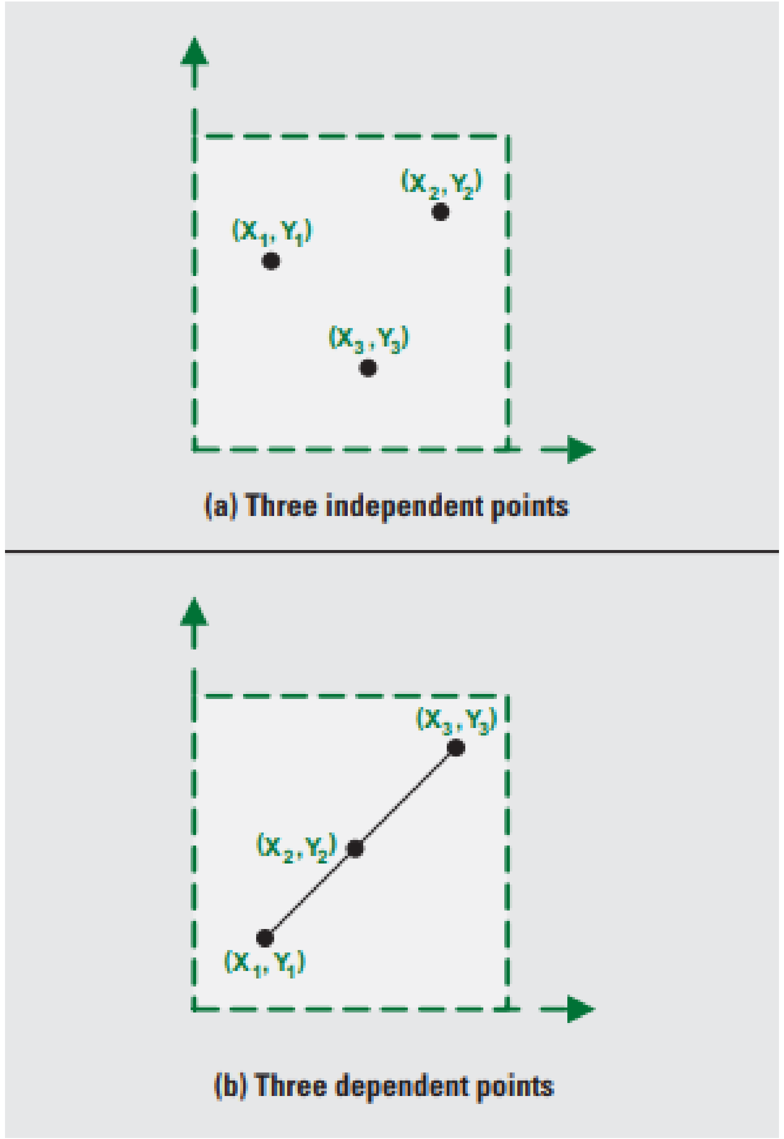
$$x_{NEW} = A\ x_{OLD} + B\ y_{OLD} + C$$

$$y_{NEW} = D\ x_{OLD} + E\ y_{OLD} + F$$

- Where A, B, C, D, E, and F are constant coefficients
  - f1() = A x_{OLD} +B y_{OLD}
  - f2() = D x_{OLD} + E y_{OLD}
  - constant1 and constant2 are C and F, respectively.

Three-points calibration method

- The goal of the touch-panel system calibration is to solve the Equations to derive values for A, B, C, D, E, and F
- Looking at these equations we know that there are six unknowns.
- Therefore, we will need six equations to solve for these unknowns and this can be achieved by doing a three-point calibration for a touch-panel system.

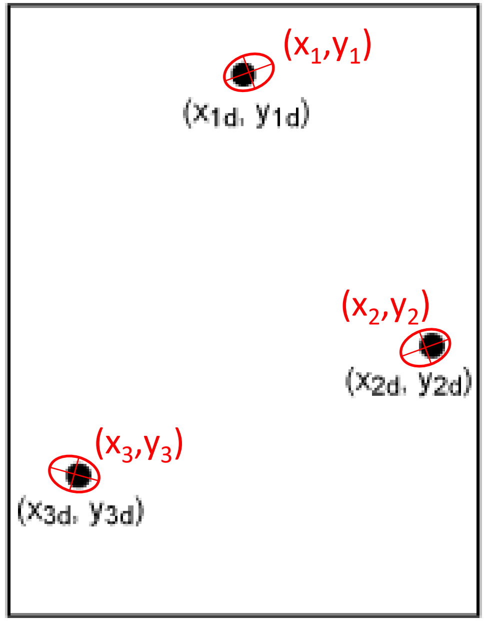


(a) Three independent points

(b) Three dependent points

Three-points calibration method (II)

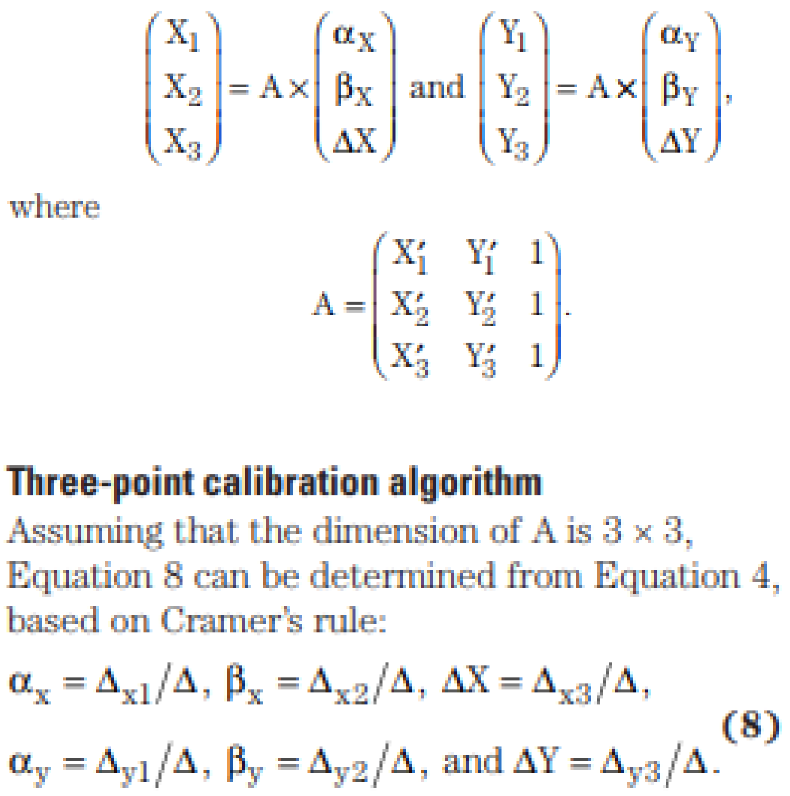
- Generate three pairs of (x, y) coordinates by touching the panel at the three pairs of display coordinates: (x_{1d}, y_{1d}), (x_{2d}, y_{2d}) and (x_{3d}, y_{3d}).
- If their corresponding touch-screen values (as presented by the touch-screen controller) are (x_1, y_1), (x_2, y_2), and (x_3, y_3), then the six unknowns can be solved by the equations shown below.
- These points must be independent of each other



$$x_{1d} = x_1 A + y_1 B + C$$
$$x_{2d} = x_2 A + y_2 B + C$$
$$x_{3d} = x_3 A + y_3 B + C$$

$$y_{1d} = x_1 D + y_1 E + F$$
$$y_{2d} = x_2 D + y_2 E + F$$
$$y_{3d} = x_3 D + y_3 E + F$$

From TI datasheet



$$\begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} = A \times \begin{pmatrix} \alpha_X \\ \beta_X \\ \Delta X \end{pmatrix} \text{ and } \begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} = A \times \begin{pmatrix} \alpha_Y \\ \beta_Y \\ \Delta Y \end{pmatrix},$$

where

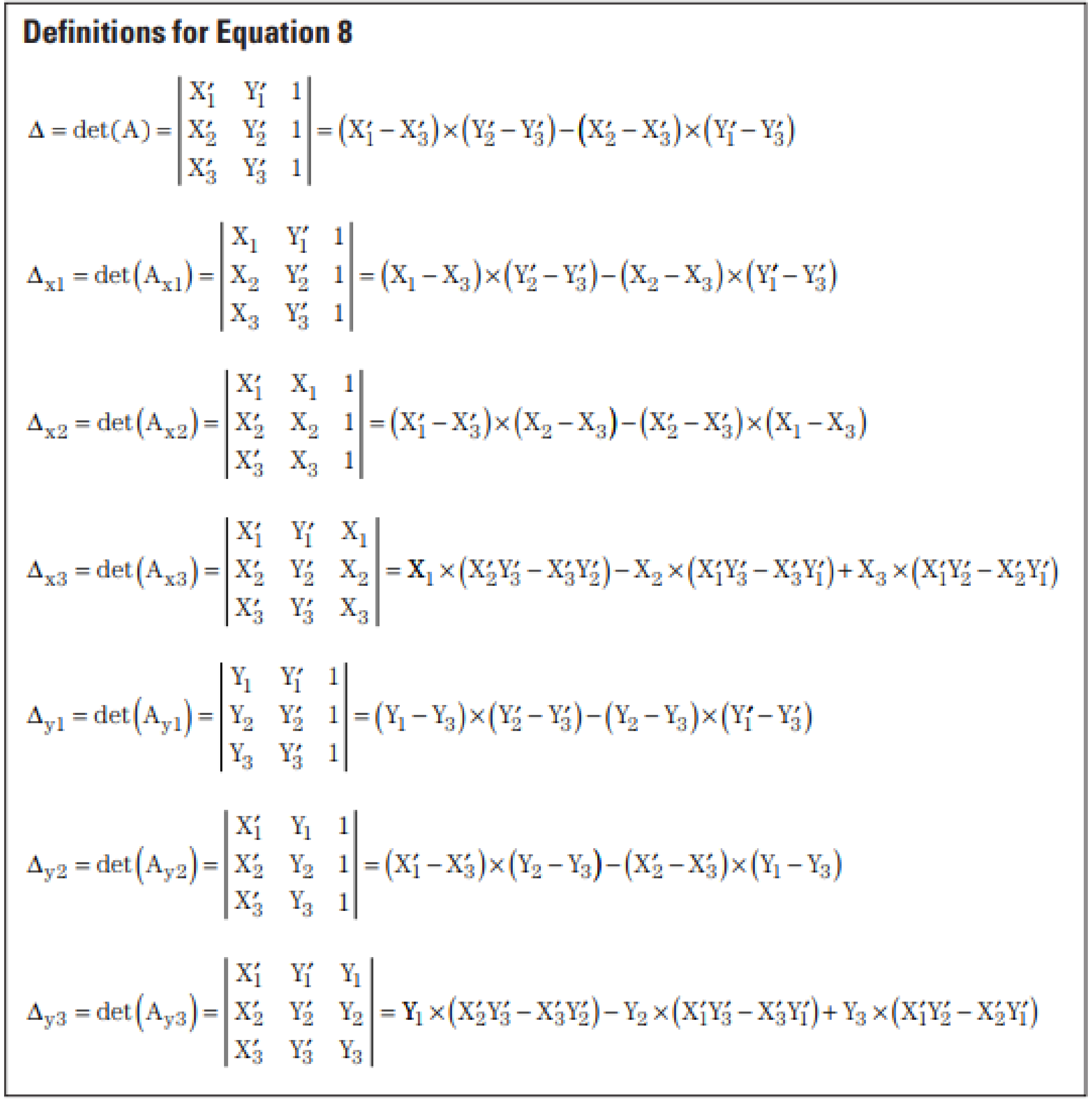
$$A = \begin{pmatrix} X'_1 & Y'_1 & 1 \\ X'_2 & Y'_2 & 1 \\ X'_3 & Y'_3 & 1 \end{pmatrix}.$$

Three-point calibration algorithm

Assuming that the dimension of A is 3 × 3, Equation 8 can be determined from Equation 4, based on Cramer's rule:

$$\alpha_x = \Delta_{x1}/\Delta,\ \beta_x = \Delta_{x2}/\Delta,\ \Delta X = \Delta_{x3}/\Delta,$$
$$\alpha_y = \Delta_{y1}/\Delta,\ \beta_y = \Delta_{y2}/\Delta, \text{ and } \Delta Y = \Delta_{y3}/\Delta. \quad \textbf{(8)}$$

http://www.ti.com/lit/an/slyt277/slyt277.pdf



Definitions for Equation 8

$$\Delta = \det(A) = \begin{vmatrix} X'_1 & Y'_1 & 1 \\ X'_2 & Y'_2 & 1 \\ X'_3 & Y'_3 & 1 \end{vmatrix} = (X'_1 - X'_3) \times (Y'_2 - Y'_3) - (X'_2 - X'_3) \times (Y'_1 - Y'_3)$$

$$\Delta_{x1} = \det(A_{x1}) = \begin{vmatrix} X_1 & Y'_1 & 1 \\ X_2 & Y'_2 & 1 \\ X_3 & Y'_3 & 1 \end{vmatrix} = (X_1 - X_3) \times (Y'_2 - Y'_3) - (X_2 - X_3) \times (Y'_1 - Y'_3)$$

$$\Delta_{x2} = \det(A_{x2}) = \begin{vmatrix} X'_1 & X_1 & 1 \\ X'_2 & X_2 & 1 \\ X'_3 & X_3 & 1 \end{vmatrix} = (X'_1 - X'_3) \times (X_2 - X_3) - (X'_2 - X'_3) \times (X_1 - X_3)$$

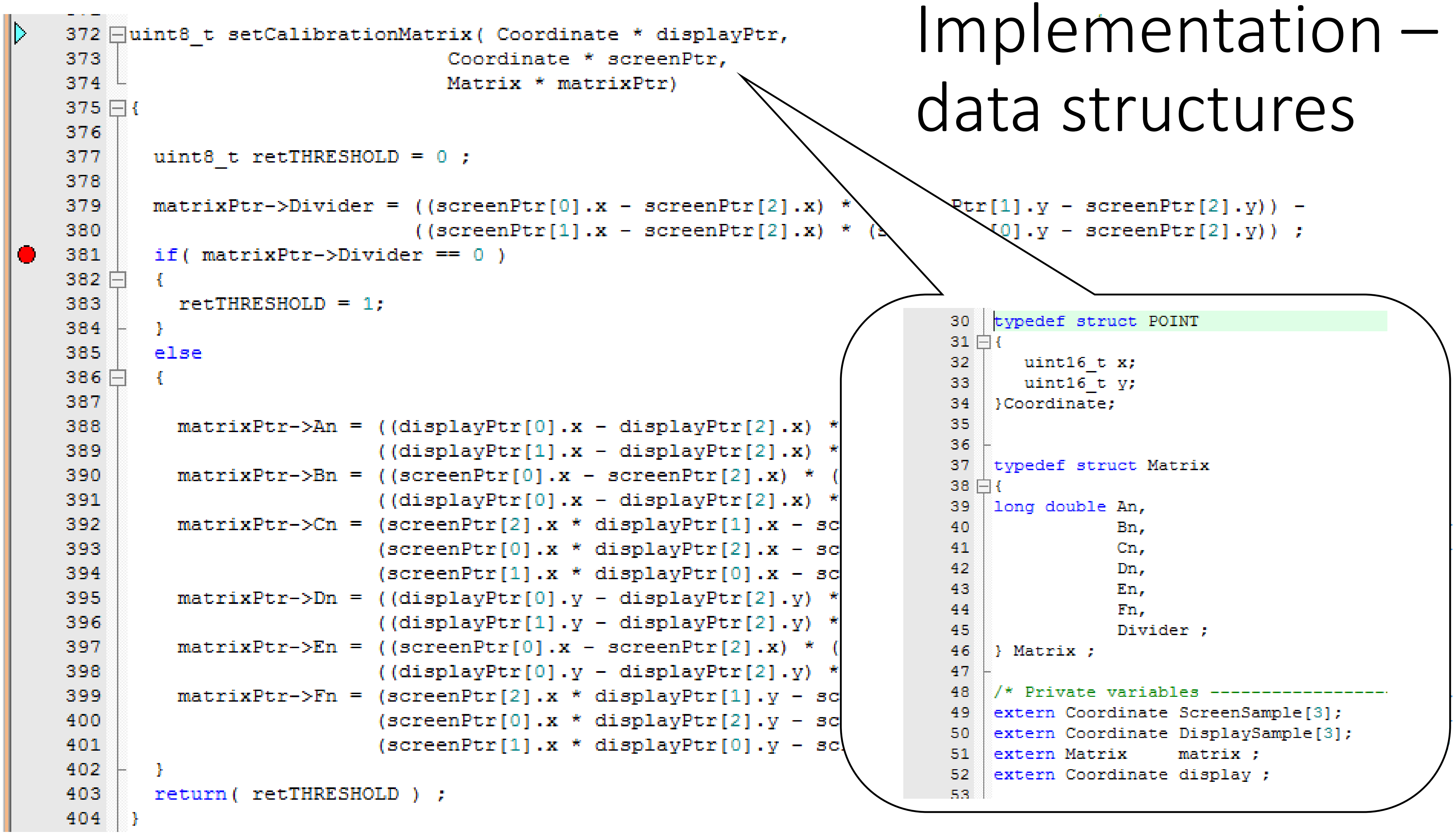
$$\Delta_{x3} = \det(A_{x3}) = \begin{vmatrix} X'_1 & Y'_1 & X_1 \\ X'_2 & Y'_2 & X_2 \\ X'_3 & Y'_3 & X_3 \end{vmatrix} = X_1 \times (X'_2Y'_3 - X'_3Y'_2) - X_2 \times (X'_1Y'_3 - X'_3Y'_1) + X_3 \times (X'_1Y'_2 - X'_2Y'_1)$$

$$\Delta_{y1} = \det(A_{y1}) = \begin{vmatrix} Y_1 & Y'_1 & 1 \\ Y_2 & Y'_2 & 1 \\ Y_3 & Y'_3 & 1 \end{vmatrix} = (Y_1 - Y_3) \times (Y'_2 - Y'_3) - (Y_2 - Y_3) \times (Y'_1 - Y'_3)$$

$$\Delta_{y2} = \det(A_{y2}) = \begin{vmatrix} X'_1 & Y_1 & 1 \\ X'_2 & Y_2 & 1 \\ X'_3 & Y_3 & 1 \end{vmatrix} = (X'_1 - X'_3) \times (Y_2 - Y_3) - (X'_2 - X'_3) \times (Y_1 - Y_3)$$

$$\Delta_{y3} = \det(A_{y3}) = \begin{vmatrix} X'_1 & Y'_1 & Y_1 \\ X'_2 & Y'_2 & Y_2 \\ X'_3 & Y'_3 & Y_3 \end{vmatrix} = Y_1 \times (X'_2Y'_3 - X'_3Y'_2) - Y_2 \times (X'_1Y'_3 - X'_3Y'_1) + Y_3 \times (X'_1Y'_2 - X'_2Y'_1)$$

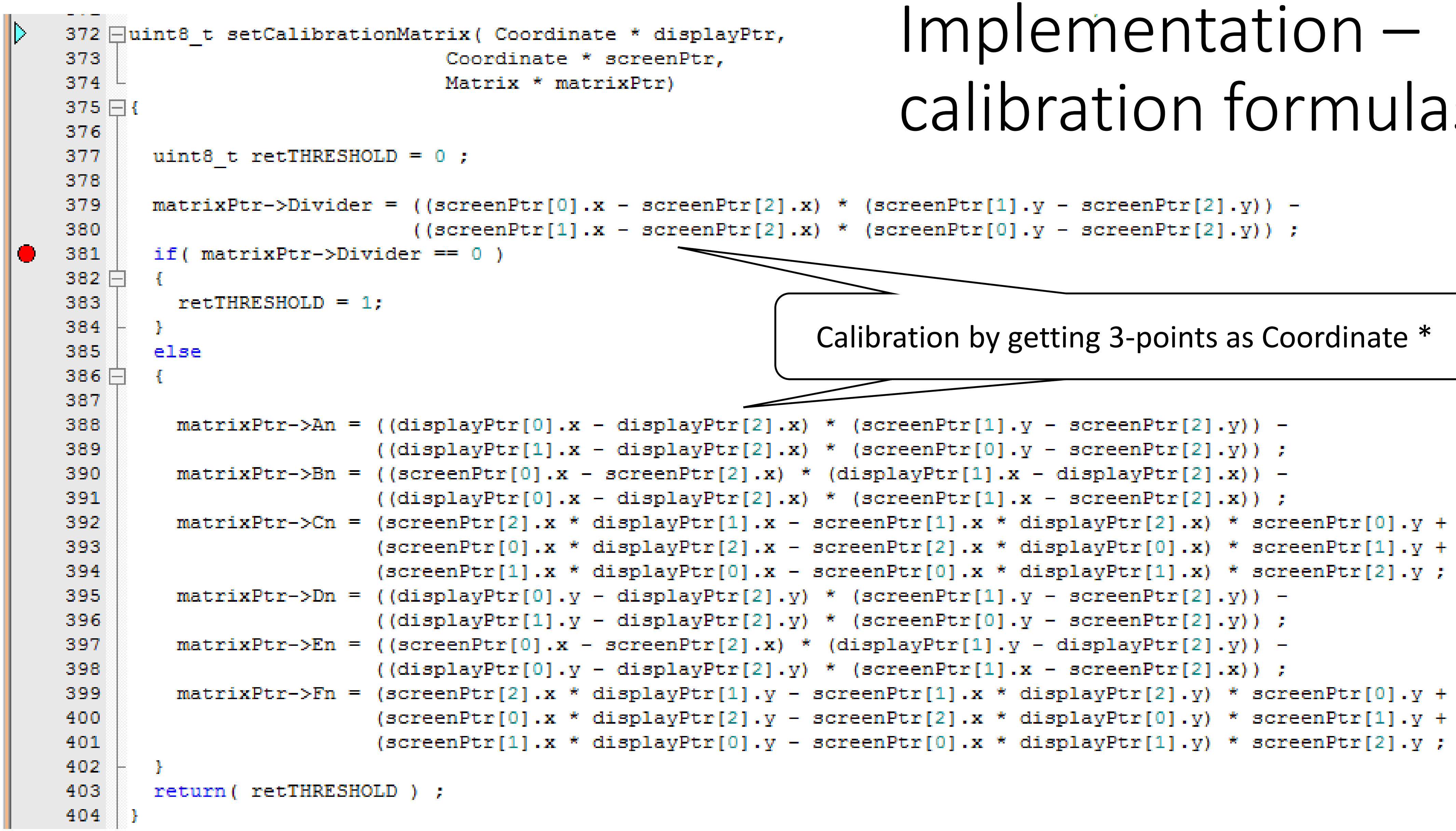
Implementation – data structures



```
uint8_t setCalibrationMatrix( Coordinate * displayPtr,
                              Coordinate * screenPtr,
                              Matrix * matrixPtr)
{

  uint8_t retTHRESHOLD = 0 ;

  matrixPtr->Divider = ((screenPtr[0].x - screenPtr[2].x) *          Ptr[1].y - screenPtr[2].y)) -
                       ((screenPtr[1].x - screenPtr[2].x) * (s          [0].y - screenPtr[2].y)) ;
  if( matrixPtr->Divider == 0 )
  {
    retTHRESHOLD = 1;
  }
  else
  {

    matrixPtr->An = ((displayPtr[0].x - displayPtr[2].x) *
                    ((displayPtr[1].x - displayPtr[2].x) *
    matrixPtr->Bn = ((screenPtr[0].x - screenPtr[2].x) * (
                    ((displayPtr[0].x - displayPtr[2].x) *
    matrixPtr->Cn = (screenPtr[2].x * displayPtr[1].x - sc
                    (screenPtr[0].x * displayPtr[2].x - sc
                    (screenPtr[1].x * displayPtr[0].x - sc
    matrixPtr->Dn = ((displayPtr[0].y - displayPtr[2].y) *
                    ((displayPtr[1].y - displayPtr[2].y) *
    matrixPtr->En = ((screenPtr[0].x - screenPtr[2].x) * (
                    ((displayPtr[0].y - displayPtr[2].y) *
    matrixPtr->Fn = (screenPtr[2].x * displayPtr[1].y - sc
                    (screenPtr[0].x * displayPtr[2].y - sc
                    (screenPtr[1].x * displayPtr[0].y - sc
  }
  return( retTHRESHOLD ) ;
}
```

```
typedef struct POINT
{
   uint16_t x;
   uint16_t y;
}Coordinate;


typedef struct Matrix
{
long double An,
            Bn,
            Cn,
            Dn,
            En,
            Fn,
            Divider ;
} Matrix ;

/* Private variables ------------------
extern Coordinate ScreenSample[3];
extern Coordinate DisplaySample[3];
extern Matrix     matrix ;
extern Coordinate display ;
```

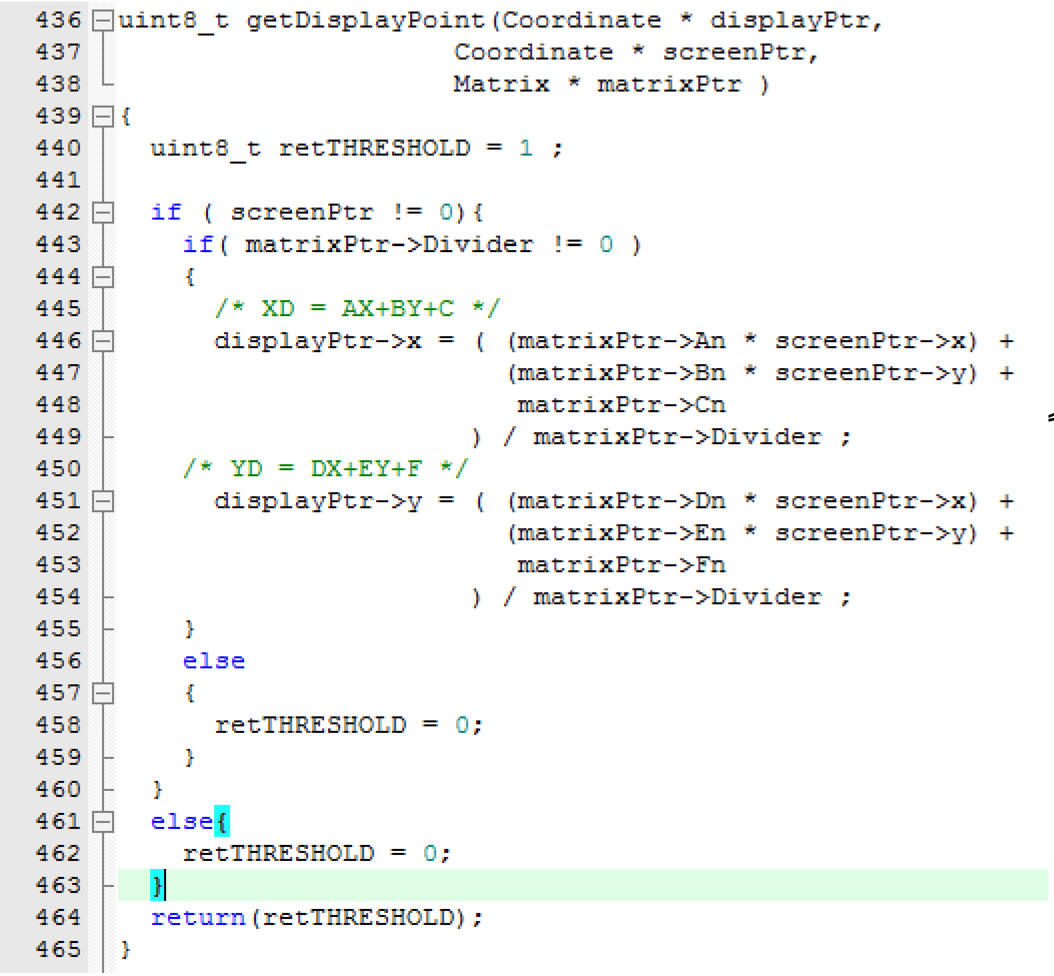
Implementation – calibration formulas



```
uint8_t setCalibrationMatrix( Coordinate * displayPtr,
                          Coordinate * screenPtr,
                          Matrix * matrixPtr)
{

  uint8_t retTHRESHOLD = 0 ;

  matrixPtr->Divider = ((screenPtr[0].x - screenPtr[2].x) * (screenPtr[1].y - screenPtr[2].y)) -
                       ((screenPtr[1].x - screenPtr[2].x) * (screenPtr[0].y - screenPtr[2].y)) ;
  if( matrixPtr->Divider == 0 )
  {
    retTHRESHOLD = 1;
  }
  else
  {

    matrixPtr->An = ((displayPtr[0].x - displayPtr[2].x) * (screenPtr[1].y - screenPtr[2].y)) -
                    ((displayPtr[1].x - displayPtr[2].x) * (screenPtr[0].y - screenPtr[2].y)) ;
    matrixPtr->Bn = ((screenPtr[0].x - screenPtr[2].x) * (displayPtr[1].x - displayPtr[2].x)) -
                    ((displayPtr[0].x - displayPtr[2].x) * (screenPtr[1].x - screenPtr[2].x)) ;
    matrixPtr->Cn = (screenPtr[2].x * displayPtr[1].x - screenPtr[1].x * displayPtr[2].x) * screenPtr[0].y +
                    (screenPtr[0].x * displayPtr[2].x - screenPtr[2].x * displayPtr[0].x) * screenPtr[1].y +
                    (screenPtr[1].x * displayPtr[0].x - screenPtr[0].x * displayPtr[1].x) * screenPtr[2].y ;
    matrixPtr->Dn = ((displayPtr[0].y - displayPtr[2].y) * (screenPtr[1].y - screenPtr[2].y)) -
                    ((displayPtr[1].y - displayPtr[2].y) * (screenPtr[0].y - screenPtr[2].y)) ;
    matrixPtr->En = ((screenPtr[0].x - screenPtr[2].x) * (displayPtr[1].y - displayPtr[2].y)) -
                    ((displayPtr[0].y - displayPtr[2].y) * (screenPtr[1].x - screenPtr[2].x)) ;
    matrixPtr->Fn = (screenPtr[2].x * displayPtr[1].y - screenPtr[1].x * displayPtr[2].y) * screenPtr[0].y +
                    (screenPtr[0].x * displayPtr[2].y - screenPtr[2].x * displayPtr[0].y) * screenPtr[1].y +
                    (screenPtr[1].x * displayPtr[0].y - screenPtr[0].x * displayPtr[1].y) * screenPtr[2].y ;
  }
  return( retTHRESHOLD ) ;
}
```

Calibration by getting 3-points as Coordinate *

Implementation – adjust point position

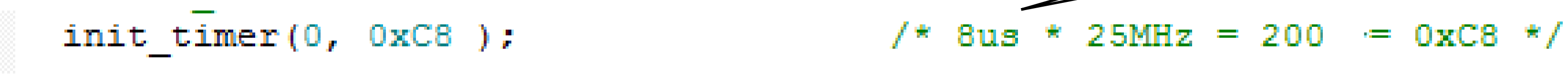


```
uint8_t getDisplayPoint(Coordinate * displayPtr,
                     Coordinate * screenPtr,
                     Matrix * matrixPtr )
{
  uint8_t retTHRESHOLD = 1 ;

  if ( screenPtr != 0){
    if( matrixPtr->Divider != 0 )
    {
      /* XD = AX+BY+C */
      displayPtr->x = ( (matrixPtr->An * screenPtr->x) +
                        (matrixPtr->Bn * screenPtr->y) +
                         matrixPtr->Cn
                      ) / matrixPtr->Divider ;
    /* YD = DX+EY+F */
      displayPtr->y = ( (matrixPtr->Dn * screenPtr->x) +
                        (matrixPtr->En * screenPtr->y) +
                         matrixPtr->Fn
                      ) / matrixPtr->Divider ;
    }
    else
    {
      retTHRESHOLD = 0;
    }
  }
  else{
    retTHRESHOLD = 0;
  }
  return(retTHRESHOLD);
}
```

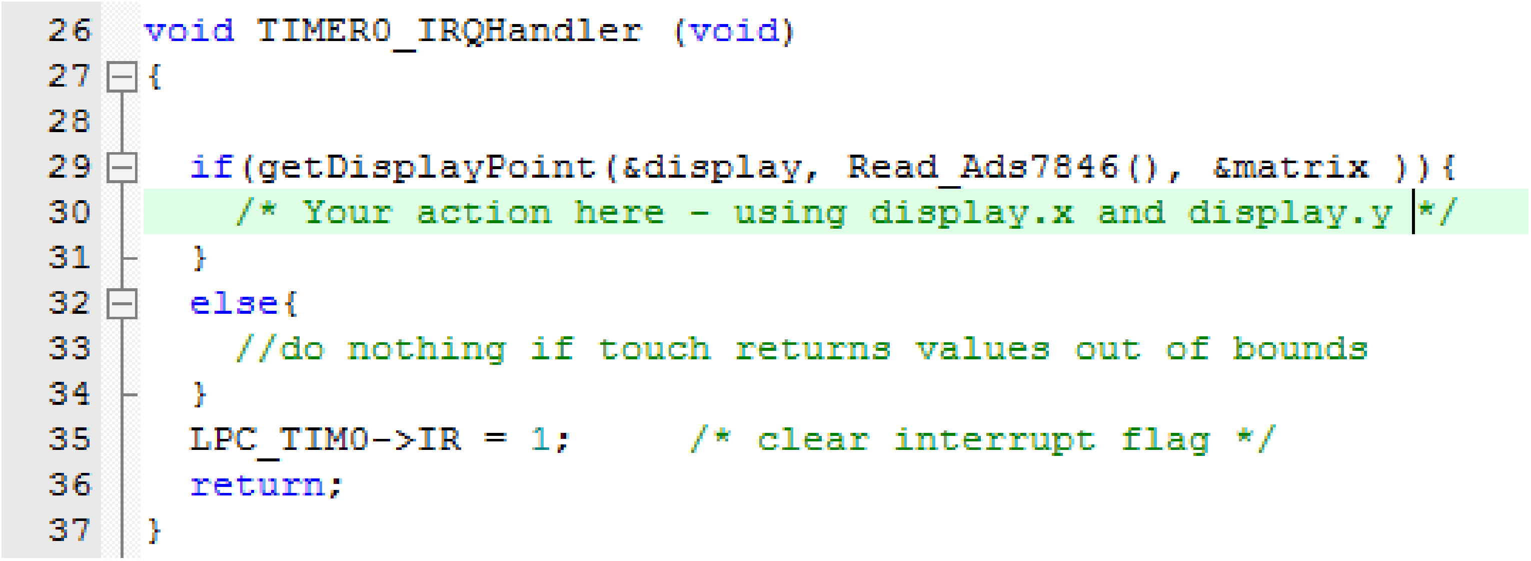
This function uses the calibration values.

How to use TP with timer

125kHz conversion rate
Period = $(125kHz)^{-1}$ = 8us



```
init_timer(0, 0xC8 );                          /* 8us * 25MHz = 200  = 0xC8 */
```



```
void TIMER0_IRQHandler (void)
{

  if(getDisplayPoint(&display, Read_Ads7846(), &matrix )){
    /* Your action here - using display.x and display.y */
  }
  else{
    //do nothing if touch returns values out of bounds
  }
  LPC_TIM0->IR = 1;      /* clear interrupt flag */
  return;
}
```