/\*!

\* @file Adafruit\_SPITFT.h

\*

\* Part of Adafruit's GFX graphics library. Originally this class was

\* written to handle a range of color TFT displays connected via SPI,

\* but over time this library and some display-specific subclasses have

\* mutated to include some color OLEDs as well as parallel-interfaced

\* displays. The name's been kept for the sake of older code.

\*

\* Adafruit invests time and resources providing this open source code,

\* please support Adafruit and open-source hardware by purchasing

\* products from Adafruit!

\*

\* Written by Limor "ladyada" Fried for Adafruit Industries,

\* with contributions from the open source community.

\*

\* BSD license, all text here must be included in any redistribution.

\*/

#ifndef \_ADAFRUIT\_SPITFT\_H\_

#define \_ADAFRUIT\_SPITFT\_H\_

#if !defined(\_\_AVR\_ATtiny85\_\_) // Not for ATtiny, at all

#include "Adafruit\_GFX.h"

#include <SPI.h>

// HARDWARE CONFIG ---------------------------------------------------------

#if defined(\_\_AVR\_\_)

typedef uint8\_t ADAGFX\_PORT\_t; ///< PORT values are 8-bit

#define USE\_FAST\_PINIO ///< Use direct PORT register access

#elif defined(ARDUINO\_STM32\_FEATHER) // WICED

typedef class HardwareSPI SPIClass; ///< SPI is a bit odd on WICED

typedef uint32\_t ADAGFX\_PORT\_t; ///< PORT values are 32-bit

#elif defined(\_\_arm\_\_)

#if defined(ARDUINO\_ARCH\_SAMD)

// Adafruit M0, M4

typedef uint32\_t ADAGFX\_PORT\_t; ///< PORT values are 32-bit

#define USE\_FAST\_PINIO ///< Use direct PORT register access

#define HAS\_PORT\_SET\_CLR ///< PORTs have set & clear registers

#elif defined(CORE\_TEENSY)

// PJRC Teensy 4.x

#if defined(\_\_IMXRT1052\_\_) || defined(\_\_IMXRT1062\_\_) // Teensy 4.x

typedef uint32\_t ADAGFX\_PORT\_t; ///< PORT values are 32-bit

// PJRC Teensy 3.x

#else

typedef uint8\_t ADAGFX\_PORT\_t; ///< PORT values are 8-bit

#endif

#define USE\_FAST\_PINIO ///< Use direct PORT register access

#define HAS\_PORT\_SET\_CLR ///< PORTs have set & clear registers

#else

// Arduino Due?

typedef uint32\_t ADAGFX\_PORT\_t; ///< PORT values are 32-bit

// USE\_FAST\_PINIO not available here (yet)...Due has a totally different

// GPIO register set and will require some changes elsewhere (e.g. in

// constructors especially).

#endif

#else // !ARM

// Probably ESP8266 or ESP32. USE\_FAST\_PINIO is not available here (yet)

// but don't worry about it too much...the digitalWrite() implementation

// on these platforms is reasonably efficient and already RAM-resident,

// only gotcha then is no parallel connection support for now.

typedef uint32\_t ADAGFX\_PORT\_t; ///< PORT values are 32-bit

#endif // end !ARM

typedef volatile ADAGFX\_PORT\_t \*PORTreg\_t; ///< PORT register type

#if defined(\_\_AVR\_\_)

#define DEFAULT\_SPI\_FREQ 8000000L ///< Hardware SPI default speed

#else

#define DEFAULT\_SPI\_FREQ 16000000L ///< Hardware SPI default speed

#endif

#if defined(ADAFRUIT\_PYPORTAL) || defined(ADAFRUIT\_PYPORTAL\_M4\_TITANO) || \

defined(ADAFRUIT\_PYBADGE\_M4\_EXPRESS) || \

defined(ADAFRUIT\_PYGAMER\_M4\_EXPRESS) || \

defined(ADAFRUIT\_MONSTER\_M4SK\_EXPRESS) || defined(NRF52\_SERIES) || \

defined(ADAFRUIT\_CIRCUITPLAYGROUND\_M0)

#define USE\_SPI\_DMA ///< Auto DMA

#else

//#define USE\_SPI\_DMA ///< If set,

// use DMA if available

#endif

// Another "oops" name -- this now also handles parallel DMA.

// If DMA is enabled, Arduino sketch MUST #include <Adafruit\_ZeroDMA.h>

// Estimated RAM usage:

// 4 bytes/pixel on display major axis + 8 bytes/pixel on minor axis,

// e.g. 320x240 pixels = 320 \* 4 + 240 \* 8 = 3,200 bytes.

#if defined(USE\_SPI\_DMA) && (defined(\_\_SAMD51\_\_) || defined(ARDUINO\_SAMD\_ZERO))

#include <Adafruit\_ZeroDMA.h>

#endif

// This is kind of a kludge. Needed a way to disambiguate the software SPI

// and parallel constructors via their argument lists. Originally tried a

// bool as the first argument to the parallel constructor (specifying 8-bit

// vs 16-bit interface) but the compiler regards this as equivalent to an

// integer and thus still ambiguous. SO...the parallel constructor requires

// an enumerated type as the first argument: tft8 (for 8-bit parallel) or

// tft16 (for 16-bit)...even though 16-bit isn't fully implemented or tested

// and might never be, still needed that disambiguation from soft SPI.

/\*! For first arg to parallel constructor \*/

enum tftBusWidth { tft8bitbus, tft16bitbus };

// CLASS DEFINITION --------------------------------------------------------

/\*!

@brief Adafruit\_SPITFT is an intermediary class between Adafruit\_GFX

and various hardware-specific subclasses for different displays.

It handles certain operations that are common to a range of

displays (address window, area fills, etc.). Originally these were

all color TFT displays interfaced via SPI, but it's since expanded

to include color OLEDs and parallel-interfaced TFTs. THE NAME HAS

BEEN KEPT TO AVOID BREAKING A LOT OF SUBCLASSES AND EXAMPLE CODE.

Many of the class member functions similarly live on with names

that don't necessarily accurately describe what they're doing,

again to avoid breaking a lot of other code. If in doubt, read

the comments.

\*/

class Adafruit\_SPITFT : public Adafruit\_GFX {

public:

// CONSTRUCTORS --------------------------------------------------------

// Software SPI constructor: expects width & height (at default rotation

// setting 0), 4 signal pins (cs, dc, mosi, sclk), 2 optional pins

// (reset, miso). cs argument is required but can be -1 if unused --

// rather than moving it to the optional arguments, it was done this way

// to avoid breaking existing code (-1 option was a later addition).

Adafruit\_SPITFT(uint16\_t w, uint16\_t h, int8\_t cs, int8\_t dc, int8\_t mosi,

int8\_t sck, int8\_t rst = -1, int8\_t miso = -1);

// Hardware SPI constructor using the default SPI port: expects width &

// height (at default rotation setting 0), 2 signal pins (cs, dc),

// optional reset pin. cs is required but can be -1 if unused -- rather

// than moving it to the optional arguments, it was done this way to

// avoid breaking existing code (-1 option was a later addition).

Adafruit\_SPITFT(uint16\_t w, uint16\_t h, int8\_t cs, int8\_t dc,

int8\_t rst = -1);

#if !defined(ESP8266) // See notes in .cpp

// Hardware SPI constructor using an arbitrary SPI peripheral: expects

// width & height (rotation 0), SPIClass pointer, 2 signal pins (cs, dc)

// and optional reset pin. cs is required but can be -1 if unused.

Adafruit\_SPITFT(uint16\_t w, uint16\_t h, SPIClass \*spiClass, int8\_t cs,

int8\_t dc, int8\_t rst = -1);

#endif // end !ESP8266

// Parallel constructor: expects width & height (rotation 0), flag

// indicating whether 16-bit (true) or 8-bit (false) interface, 3 signal

// pins (d0, wr, dc), 3 optional pins (cs, rst, rd). 16-bit parallel

// isn't even fully implemented but the 'wide' flag was added as a

// required argument to avoid ambiguity with other constructors.

Adafruit\_SPITFT(uint16\_t w, uint16\_t h, tftBusWidth busWidth, int8\_t d0,

int8\_t wr, int8\_t dc, int8\_t cs = -1, int8\_t rst = -1,

int8\_t rd = -1);

// DESTRUCTOR ----------------------------------------------------------

~Adafruit\_SPITFT(){};

// CLASS MEMBER FUNCTIONS ----------------------------------------------

// These first two functions MUST be declared by subclasses:

/\*!

@brief Display-specific initialization function.

@param freq SPI frequency, in hz (or 0 for default or unused).

\*/

virtual void begin(uint32\_t freq) = 0;

/\*!

@brief Set up the specific display hardware's "address window"

for subsequent pixel-pushing operations.

@param x Leftmost pixel of area to be drawn (MUST be within

display bounds at current rotation setting).

@param y Topmost pixel of area to be drawn (MUST be within

display bounds at current rotation setting).

@param w Width of area to be drawn, in pixels (MUST be >0 and,

added to x, within display bounds at current rotation).

@param h Height of area to be drawn, in pixels (MUST be >0 and,

added to x, within display bounds at current rotation).

\*/

virtual void setAddrWindow(uint16\_t x, uint16\_t y, uint16\_t w,

uint16\_t h) = 0;

// Remaining functions do not need to be declared in subclasses

// unless they wish to provide hardware-specific optimizations.

// Brief comments here...documented more thoroughly in .cpp file.

// Subclass' begin() function invokes this to initialize hardware.

// freq=0 to use default SPI speed. spiMode must be one of the SPI\_MODEn

// values defined in SPI.h, which are NOT the same as 0 for SPI\_MODE0,

// 1 for SPI\_MODE1, etc...use ONLY the SPI\_MODEn defines! Only!

// Name is outdated (interface may be parallel) but for compatibility:

void initSPI(uint32\_t freq = 0, uint8\_t spiMode = SPI\_MODE0);

void setSPISpeed(uint32\_t freq);

// Chip select and/or hardware SPI transaction start as needed:

void startWrite(void);

// Chip deselect and/or hardware SPI transaction end as needed:

void endWrite(void);

void sendCommand(uint8\_t commandByte, uint8\_t \*dataBytes,

uint8\_t numDataBytes);

void sendCommand(uint8\_t commandByte, const uint8\_t \*dataBytes = NULL,

uint8\_t numDataBytes = 0);

void sendCommand16(uint16\_t commandWord, const uint8\_t \*dataBytes = NULL,

uint8\_t numDataBytes = 0);

uint8\_t readcommand8(uint8\_t commandByte, uint8\_t index = 0);

uint16\_t readcommand16(uint16\_t addr);

// These functions require a chip-select and/or SPI transaction

// around them. Higher-level graphics primitives might start a

// single transaction and then make multiple calls to these functions

// (e.g. circle or text rendering might make repeated lines or rects)

// before ending the transaction. It's more efficient than starting a

// transaction every time.

void writePixel(int16\_t x, int16\_t y, uint16\_t color);

void writePixels(uint16\_t \*colors, uint32\_t len, bool block = true,

bool bigEndian = false);

void writeColor(uint16\_t color, uint32\_t len);

void writeFillRect(int16\_t x, int16\_t y, int16\_t w, int16\_t h,

uint16\_t color);

void writeFastHLine(int16\_t x, int16\_t y, int16\_t w, uint16\_t color);

void writeFastVLine(int16\_t x, int16\_t y, int16\_t h, uint16\_t color);

// This is a new function, similar to writeFillRect() except that

// all arguments MUST be onscreen, sorted and clipped. If higher-level

// primitives can handle their own sorting/clipping, it avoids repeating

// such operations in the low-level code, making it potentially faster.

// CALLING THIS WITH UNCLIPPED OR NEGATIVE VALUES COULD BE DISASTROUS.

inline void writeFillRectPreclipped(int16\_t x, int16\_t y, int16\_t w,

int16\_t h, uint16\_t color);

// Another new function, companion to the new non-blocking

// writePixels() variant.

void dmaWait(void);

// Used by writePixels() in some situations, but might have rare need in

// user code, so it's public...

bool dmaBusy(void) const; // true if DMA is used and busy, false otherwise

void swapBytes(uint16\_t \*src, uint32\_t len, uint16\_t \*dest = NULL);

// These functions are similar to the 'write' functions above, but with

// a chip-select and/or SPI transaction built-in. They're typically used

// solo -- that is, as graphics primitives in themselves, not invoked by

// higher-level primitives (which should use the functions above).

void drawPixel(int16\_t x, int16\_t y, uint16\_t color);

void fillRect(int16\_t x, int16\_t y, int16\_t w, int16\_t h, uint16\_t color);

void drawFastHLine(int16\_t x, int16\_t y, int16\_t w, uint16\_t color);

void drawFastVLine(int16\_t x, int16\_t y, int16\_t h, uint16\_t color);

// A single-pixel push encapsulated in a transaction. I don't think

// this is used anymore (BMP demos might've used it?) but is provided

// for backward compatibility, consider it deprecated:

void pushColor(uint16\_t color);

using Adafruit\_GFX::drawRGBBitmap; // Check base class first

void drawRGBBitmap(int16\_t x, int16\_t y, uint16\_t \*pcolors, int16\_t w,

int16\_t h);

void invertDisplay(bool i);

uint16\_t color565(uint8\_t r, uint8\_t g, uint8\_t b);

// Despite parallel additions, function names kept for compatibility:

void spiWrite(uint8\_t b); // Write single byte as DATA

void writeCommand(uint8\_t cmd); // Write single byte as COMMAND

uint8\_t spiRead(void); // Read single byte of data

void write16(uint16\_t w); // Write 16-bit value as DATA

void writeCommand16(uint16\_t cmd); // Write 16-bit value as COMMAND

uint16\_t read16(void); // Read single 16-bit value

// Most of these low-level functions were formerly macros in

// Adafruit\_SPITFT\_Macros.h. Some have been made into inline functions

// to avoid macro mishaps. Despite the addition of code for a parallel

// display interface, the names have been kept for backward

// compatibility (some subclasses may be invoking these):

void SPI\_WRITE16(uint16\_t w); // Not inline

void SPI\_WRITE32(uint32\_t l); // Not inline

// Old code had both a spiWrite16() function and SPI\_WRITE16 macro

// in addition to the SPI\_WRITE32 macro. The latter two have been

// made into functions here, and spiWrite16() removed (use SPI\_WRITE16()

// instead). It looks like most subclasses had gotten comfortable with

// SPI\_WRITE16 and SPI\_WRITE32 anyway so those names were kept rather

// than the less-obnoxious camelcase variants, oh well.

// Placing these functions entirely in the class definition inlines

// them implicitly them while allowing their use in other code:

/\*!

@brief Set the chip-select line HIGH. Does NOT check whether CS pin

is set (>=0), that should be handled in calling function.

Despite function name, this is used even if the display

connection is parallel.

\*/

void SPI\_CS\_HIGH(void) {

#if defined(USE\_FAST\_PINIO)

#if defined(HAS\_PORT\_SET\_CLR)

#if defined(KINETISK)

\*csPortSet = 1;

#else // !KINETISK

\*csPortSet = csPinMask;

#endif // end !KINETISK

#else // !HAS\_PORT\_SET\_CLR

\*csPort |= csPinMaskSet;

#endif // end !HAS\_PORT\_SET\_CLR

#else // !USE\_FAST\_PINIO

digitalWrite(\_cs, HIGH);

#endif // end !USE\_FAST\_PINIO

}

/\*!

@brief Set the chip-select line LOW. Does NOT check whether CS pin

is set (>=0), that should be handled in calling function.

Despite function name, this is used even if the display

connection is parallel.

\*/

void SPI\_CS\_LOW(void) {

#if defined(USE\_FAST\_PINIO)

#if defined(HAS\_PORT\_SET\_CLR)

#if defined(KINETISK)

\*csPortClr = 1;

#else // !KINETISK

\*csPortClr = csPinMask;

#endif // end !KINETISK

#else // !HAS\_PORT\_SET\_CLR

\*csPort &= csPinMaskClr;

#endif // end !HAS\_PORT\_SET\_CLR

#else // !USE\_FAST\_PINIO

digitalWrite(\_cs, LOW);

#endif // end !USE\_FAST\_PINIO

}

/\*!

@brief Set the data/command line HIGH (data mode).

\*/

void SPI\_DC\_HIGH(void) {

#if defined(USE\_FAST\_PINIO)

#if defined(HAS\_PORT\_SET\_CLR)

#if defined(KINETISK)

\*dcPortSet = 1;

#else // !KINETISK

\*dcPortSet = dcPinMask;

#endif // end !KINETISK

#else // !HAS\_PORT\_SET\_CLR

\*dcPort |= dcPinMaskSet;

#endif // end !HAS\_PORT\_SET\_CLR

#else // !USE\_FAST\_PINIO

digitalWrite(\_dc, HIGH);

#endif // end !USE\_FAST\_PINIO

}

/\*!

@brief Set the data/command line LOW (command mode).

\*/

void SPI\_DC\_LOW(void) {

#if defined(USE\_FAST\_PINIO)

#if defined(HAS\_PORT\_SET\_CLR)

#if defined(KINETISK)

\*dcPortClr = 1;

#else // !KINETISK

\*dcPortClr = dcPinMask;

#endif // end !KINETISK

#else // !HAS\_PORT\_SET\_CLR

\*dcPort &= dcPinMaskClr;

#endif // end !HAS\_PORT\_SET\_CLR

#else // !USE\_FAST\_PINIO

digitalWrite(\_dc, LOW);

#endif // end !USE\_FAST\_PINIO

}

protected:

// A few more low-level member functions -- some may have previously

// been macros. Shouldn't have a need to access these externally, so

// they've been moved to the protected section. Additionally, they're

// declared inline here and the code is in the .cpp file, since outside

// code doesn't need to see these.

inline void SPI\_MOSI\_HIGH(void);

inline void SPI\_MOSI\_LOW(void);

inline void SPI\_SCK\_HIGH(void);

inline void SPI\_SCK\_LOW(void);

inline bool SPI\_MISO\_READ(void);

inline void SPI\_BEGIN\_TRANSACTION(void);

inline void SPI\_END\_TRANSACTION(void);

inline void TFT\_WR\_STROBE(void); // Parallel interface write strobe

inline void TFT\_RD\_HIGH(void); // Parallel interface read high

inline void TFT\_RD\_LOW(void); // Parallel interface read low

// CLASS INSTANCE VARIABLES --------------------------------------------

// Here be dragons! There's a big union of three structures here --

// one each for hardware SPI, software (bitbang) SPI, and parallel

// interfaces. This is to save some memory, since a display's connection

// will be only one of these. The order of some things is a little weird

// in an attempt to get values to align and pack better in RAM.

#if defined(USE\_FAST\_PINIO)

#if defined(HAS\_PORT\_SET\_CLR)

PORTreg\_t csPortSet; ///< PORT register for chip select SET

PORTreg\_t csPortClr; ///< PORT register for chip select CLEAR

PORTreg\_t dcPortSet; ///< PORT register for data/command SET

PORTreg\_t dcPortClr; ///< PORT register for data/command CLEAR

#else // !HAS\_PORT\_SET\_CLR

PORTreg\_t csPort; ///< PORT register for chip select

PORTreg\_t dcPort; ///< PORT register for data/command

#endif // end HAS\_PORT\_SET\_CLR

#endif // end USE\_FAST\_PINIO

#if defined(\_\_cplusplus) && (\_\_cplusplus >= 201100)

union {

#endif

struct { // Values specific to HARDWARE SPI:

SPIClass \*\_spi; ///< SPI class pointer

#if defined(SPI\_HAS\_TRANSACTION)

SPISettings settings; ///< SPI transaction settings

#else

uint32\_t \_freq; ///< SPI bitrate (if no SPI transactions)

#endif

uint32\_t \_mode; ///< SPI data mode (transactions or no)

} hwspi; ///< Hardware SPI values

struct { // Values specific to SOFTWARE SPI:

#if defined(USE\_FAST\_PINIO)

PORTreg\_t misoPort; ///< PORT (PIN) register for MISO

#if defined(HAS\_PORT\_SET\_CLR)

PORTreg\_t mosiPortSet; ///< PORT register for MOSI SET

PORTreg\_t mosiPortClr; ///< PORT register for MOSI CLEAR

PORTreg\_t sckPortSet; ///< PORT register for SCK SET

PORTreg\_t sckPortClr; ///< PORT register for SCK CLEAR

#if !defined(KINETISK)

ADAGFX\_PORT\_t mosiPinMask; ///< Bitmask for MOSI

ADAGFX\_PORT\_t sckPinMask; ///< Bitmask for SCK

#endif // end !KINETISK

#else // !HAS\_PORT\_SET\_CLR

PORTreg\_t mosiPort; ///< PORT register for MOSI

PORTreg\_t sckPort; ///< PORT register for SCK

ADAGFX\_PORT\_t mosiPinMaskSet; ///< Bitmask for MOSI SET (OR)

ADAGFX\_PORT\_t mosiPinMaskClr; ///< Bitmask for MOSI CLEAR (AND)

ADAGFX\_PORT\_t sckPinMaskSet; ///< Bitmask for SCK SET (OR bitmask)

ADAGFX\_PORT\_t sckPinMaskClr; ///< Bitmask for SCK CLEAR (AND)

#endif // end HAS\_PORT\_SET\_CLR

#if !defined(KINETISK)

ADAGFX\_PORT\_t misoPinMask; ///< Bitmask for MISO

#endif // end !KINETISK

#endif // end USE\_FAST\_PINIO

int8\_t \_mosi; ///< MOSI pin #

int8\_t \_miso; ///< MISO pin #

int8\_t \_sck; ///< SCK pin #

} swspi; ///< Software SPI values

struct { // Values specific to 8-bit parallel:

#if defined(USE\_FAST\_PINIO)

#if defined(\_\_IMXRT1052\_\_) || defined(\_\_IMXRT1062\_\_) // Teensy 4.x

volatile uint32\_t \*writePort; ///< PORT register for DATA WRITE

volatile uint32\_t \*readPort; ///< PORT (PIN) register for DATA READ

#else

volatile uint8\_t \*writePort; ///< PORT register for DATA WRITE

volatile uint8\_t \*readPort; ///< PORT (PIN) register for DATA READ

#endif

#if defined(HAS\_PORT\_SET\_CLR)

// Port direction register pointers are always 8-bit regardless of

// PORTreg\_t -- even if 32-bit port, we modify a byte-aligned 8 bits.

#if defined(\_\_IMXRT1052\_\_) || defined(\_\_IMXRT1062\_\_) // Teensy 4.x

volatile uint32\_t \*dirSet; ///< PORT byte data direction SET

volatile uint32\_t \*dirClr; ///< PORT byte data direction CLEAR

#else

volatile uint8\_t \*dirSet; ///< PORT byte data direction SET

volatile uint8\_t \*dirClr; ///< PORT byte data direction CLEAR

#endif

PORTreg\_t wrPortSet; ///< PORT register for write strobe SET

PORTreg\_t wrPortClr; ///< PORT register for write strobe CLEAR

PORTreg\_t rdPortSet; ///< PORT register for read strobe SET

PORTreg\_t rdPortClr; ///< PORT register for read strobe CLEAR

#if !defined(KINETISK)

ADAGFX\_PORT\_t wrPinMask; ///< Bitmask for write strobe

#endif // end !KINETISK

ADAGFX\_PORT\_t rdPinMask; ///< Bitmask for read strobe

#else // !HAS\_PORT\_SET\_CLR

// Port direction register pointer is always 8-bit regardless of

// PORTreg\_t -- even if 32-bit port, we modify a byte-aligned 8 bits.

volatile uint8\_t \*portDir; ///< PORT direction register

PORTreg\_t wrPort; ///< PORT register for write strobe

PORTreg\_t rdPort; ///< PORT register for read strobe

ADAGFX\_PORT\_t wrPinMaskSet; ///< Bitmask for write strobe SET (OR)

ADAGFX\_PORT\_t wrPinMaskClr; ///< Bitmask for write strobe CLEAR (AND)

ADAGFX\_PORT\_t rdPinMaskSet; ///< Bitmask for read strobe SET (OR)

ADAGFX\_PORT\_t rdPinMaskClr; ///< Bitmask for read strobe CLEAR (AND)

#endif // end HAS\_PORT\_SET\_CLR

#endif // end USE\_FAST\_PINIO

int8\_t \_d0; ///< Data pin 0 #

int8\_t \_wr; ///< Write strobe pin #

int8\_t \_rd; ///< Read strobe pin # (or -1)

bool wide = 0; ///< If true, is 16-bit interface

} tft8; ///< Parallel interface settings

#if defined(\_\_cplusplus) && (\_\_cplusplus >= 201100)

}; ///< Only one interface is active

#endif

#if defined(USE\_SPI\_DMA) && \

(defined(\_\_SAMD51\_\_) || \

defined(ARDUINO\_SAMD\_ZERO)) // Used by hardware SPI and tft8

Adafruit\_ZeroDMA dma; ///< DMA instance

DmacDescriptor \*dptr = NULL; ///< 1st descriptor

DmacDescriptor \*descriptor = NULL; ///< Allocated descriptor list

uint16\_t \*pixelBuf[2]; ///< Working buffers

uint16\_t maxFillLen; ///< Max pixels per DMA xfer

uint16\_t lastFillColor = 0; ///< Last color used w/fill

uint32\_t lastFillLen = 0; ///< # of pixels w/last fill

uint8\_t onePixelBuf; ///< For hi==lo fill

#endif

#if defined(USE\_FAST\_PINIO)

#if defined(HAS\_PORT\_SET\_CLR)

#if !defined(KINETISK)

ADAGFX\_PORT\_t csPinMask; ///< Bitmask for chip select

ADAGFX\_PORT\_t dcPinMask; ///< Bitmask for data/command

#endif // end !KINETISK

#else // !HAS\_PORT\_SET\_CLR

ADAGFX\_PORT\_t csPinMaskSet; ///< Bitmask for chip select SET (OR)

ADAGFX\_PORT\_t csPinMaskClr; ///< Bitmask for chip select CLEAR (AND)

ADAGFX\_PORT\_t dcPinMaskSet; ///< Bitmask for data/command SET (OR)

ADAGFX\_PORT\_t dcPinMaskClr; ///< Bitmask for data/command CLEAR (AND)

#endif // end HAS\_PORT\_SET\_CLR

#endif // end USE\_FAST\_PINIO

uint8\_t connection; ///< TFT\_HARD\_SPI, TFT\_SOFT\_SPI, etc.

int8\_t \_rst; ///< Reset pin # (or -1)

int8\_t \_cs; ///< Chip select pin # (or -1)

int8\_t \_dc; ///< Data/command pin #

int16\_t \_xstart = 0; ///< Internal framebuffer X offset

int16\_t \_ystart = 0; ///< Internal framebuffer Y offset

uint8\_t invertOnCommand = 0; ///< Command to enable invert mode

uint8\_t invertOffCommand = 0; ///< Command to disable invert mode

uint32\_t \_freq = 0; ///< Dummy var to keep subclasses happy

};

#endif // end \_\_AVR\_ATtiny85\_\_

#endif // end \_ADAFRUIT\_SPITFT\_H\_