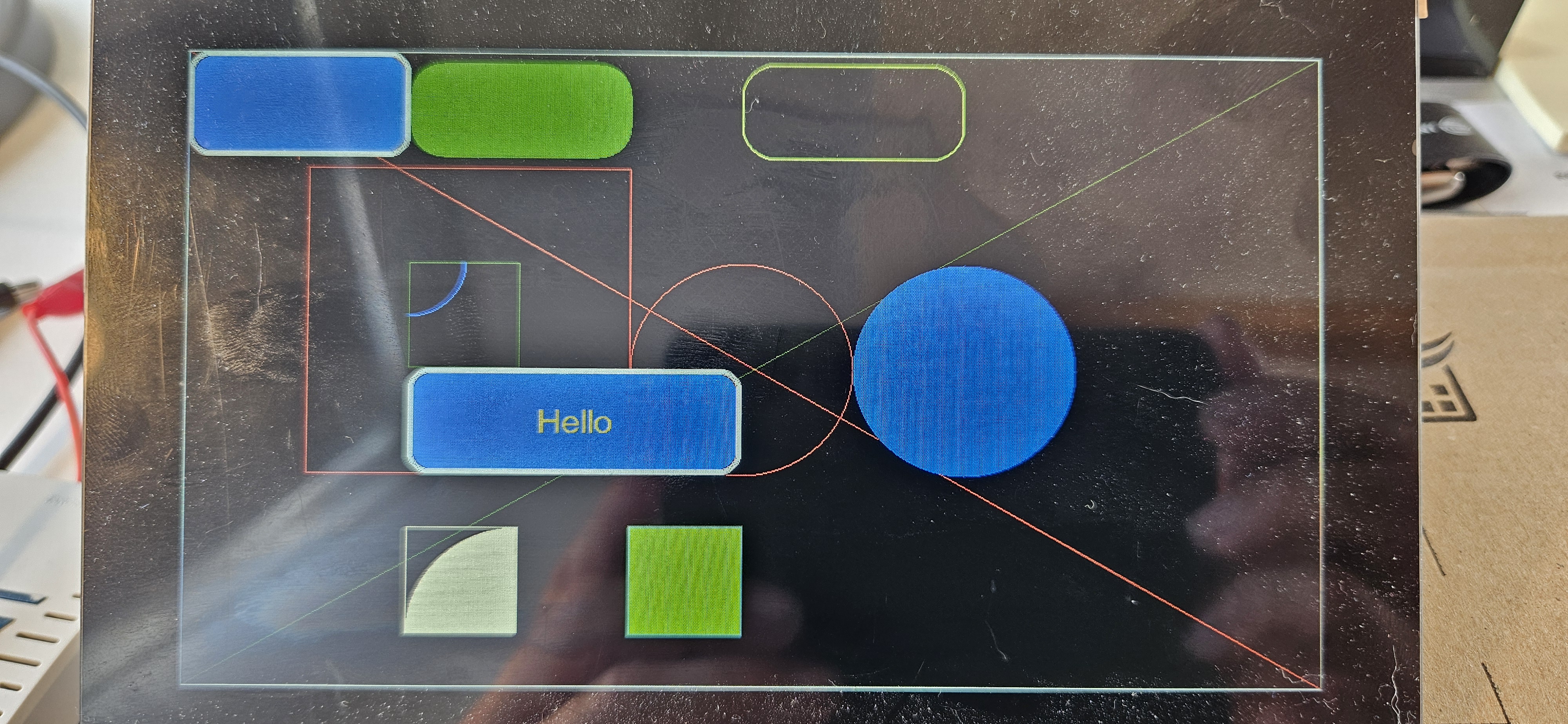
A GUI for the Riverdi STM32



An example of a C++ implementation for the Riverdi stm32 TFT 7” display

Søren Gullach | 04 June 2024

# Preface.

This is an attempt to make a C++ interface for the Riverdi STM32 (RVT70HSMNWC00-B), the goal is NOT to use the STM tool STM32Cube.

On the long run the intension is to make Windows program to setup the hardware and make/draw the GUI.

The project is made with Visual Studio and VisualGDB, GCC cpp17. On BareMetal.

To learn something about the STM32H7 we need the [STM32H7 datasheet](https://www.st.com/resource/en/reference_manual/rm0399-stm32h745755-and-stm32h747757-advanced-armbased-32bit-mcus-stmicroelectronics.pdf) and for the Riverdi display the [Riverdi STM32 datasheet](https://download.riverdi.com/RVT70HSMNWC00-B/DS_RVT70HSMNWC00-B_Rev.1.2.pdf?_gl=1*xr59zl*_ga*MTI0ODk0MDUwMy4xNzAwMzg2MDI5*_ga_7GXSNQBBQ3*MTcxMTg3MTYxOC42LjEuMTcxMTg3MTY2NS4xMy4wLjEzNTMyMjM2OTQ.).

I will in general not explain C++ the STM32H7 or Riverdi STM32 hardware.

Let’s make our hands dirty and get to the metal!

Content

[Preface. 1](#_Toc168394155)

[Chapter 1, The basic 1](#_Toc168394156)

[Organization of files. 1](#_Toc168394157)

# Chapter 1, The basic

How to get started? Out of the box the STM32H7 can “run” on basic settings, but we need to make some drivers to get any useful done.

1. A GPIO driver, to handle IO
2. A Systick driver to handle delays

With this we can make the LED blink.

1. A Clock driver to setup the clock system
2. Power driver

To make things run a little faster.

1. Timer drives
2. LTDC driver
3. DSI driver
4. FMC driver
5. I2C driver
6. DMS2D driver

To get anything on the screen

## Organization of files.

\Project The base project

\Project\osSTM32H7xx Files for OS/HW

\Project\glSTM32H7xx Files for the Graphic

All files starting with os- belongs to the STM32H7 hardware, to setup and control the STM environment.

Files starting with gl- is for the graphic library.

## LED Blink.

To make the LED, on IO GPIOJ 10, blink, we need to setup the IO port(s). This is done with the GPIO.h file, hwGPIO object.

And to setup the delay. Systick.h, hwSysTick object

From the Main file:

// create an IO pin for an LED connected to GPIOJ, pin 10

hwGPIO LedGUI(GPIOJ, 10, hwGPIO::eMode::Output, hwGPIO::ePP::PushPull);

int main()

{

// Initialize SysTick timer

hwSysTick::Init();

while (1)

{

// Toggle the LED

LedGUI.Toggle();

osDelay(1000); // Delay for x milliseconds

}

}

Compile and run 😊 We have made out first BareMetal drivers for the STM32H7.

## Clock System.

Now it’s going to be difficult, the clock system on the STM32H7 is dauting and to get to maximum speed we also need to handle the power modes!

See RCC.h and PWR.h.

At the time of writing the implementation have an unknown error with the power mode, resulting in that the HW will not always hot restart, needs a power recycle.

// Initialize hardware

Printf("HW Init CM7\n");

hwPWR::SupplyConfiguration(hwPWR::eSupplyConfigurations::C1); // Riverdi 7" uses this

hwSysClock::Setup(); // Setup RCC clock

// Update system core clock

hwSysClock::SystemCoreClockUpdate();

// Reinitialize SysTick with new clock settings

hwSysTick::Init();

Here is the hwSysClock::Setup() function

void hwSysClock::Setup()

{

// make sure MPUs run

CC.HSI(hwCC::eHSI::Mhz64); // on

CC.SYSClkSource(hwCC::eSysClkSource::HSI);

Printf("Selecting HSE\n");

CC.HSE(hwCC::eHSE::External, HSE\_VALUE); // 16MHz

CC.PLLSource(hwCC::ePLL\_SRC::HSE); // 16MHz

CC.PLL1.PllInClk(4'000'000); // 16/4 = 4MHz source for PLL1

#if C1\_CLK == 480

Printf("Selecting 480 MHz\n");

hwPWR::VOS(hwPWR::eVOSs::VOS0);

CC.PLL1.SetupInteger(240, 2, 24, 2); // (4\*240)/2 = 480 MHz

#endif

#if C1\_CLK == 400

Printf("Selecting 400 MHz\n");

hwPWR::VOS(hwPWR::eVOSs::VOS1);

CC.PLL1.SetupInteger(200, 2, 24, 2); // (4\*200)/2 = 400 MHz

#endif

#if C1\_CLK == 300

Printf("Selecting 300 MHz\n");

hwPWR::VOS(hwPWR::eVOSs::VOS2);

CC.PLL1.SetupInteger(150, 2, 24, 2); // (4\*150)/2 = 300 MHz

#endif

#if C1\_CLK == 200

Printf("Selecting 200 MHz\n");

hwPWR::VOS(hwPWR::eVOSs::VOS3);

CC.PLL1.SetupInteger(100, 2, 24, 2); // (4\*100)/2 = 200 MHz

#endif

CC.PLL1.Enable();

// LTDC clk

CC.PLL3.DivM(12); // 16/12 = 1.333MHz source for PLL3

CC.PLL3.SetupInteger(246, 6, 6, 6); // (1.333\*231)/6 = 51.333 MHz

// (1.333\*246)/6 = 54,6653 MHz

CC.PLL3.Enable();

// FMC clk

CC.PLL2.DivM(4); // 16/4 = 4MHz source for PLL2

CC.PLL2.SetupInteger(180, 4, 11, 3); // (4\*180)/3 = 240 MHz

CC.PLL2.Enable();

CC.SYSClkSource(hwCC::eSysClkSource::PLL1); // pll P\_clk;

CC.SCGU.P1Clk(hwCC::hwSCGU::ePDivisor::P\_2, false);

CC.SCGU.P2Clk(hwCC::hwSCGU::ePDivisor::P\_2, false);

CC.SCGU.P3Clk(hwCC::hwSCGU::ePDivisor::P\_2);

CC.SCGU.P4Clk(hwCC::hwSCGU::ePDivisor::P\_2);

CC.HSI(hwCC::eHSI::Off); // off

CC.CSI(hwCC::eHSE::Off); // off

}

Simple uhh.

## The Riverdi STM32.

Before we get into the graphics, we need to power up the display backlight and the graphic HW interface.

#### Backlight.

TFTDisplay.h and TFTDisplay object handles the backlight.

Riverdi STM32 uses a buck HW driver connected to TIM15 CH1, the PWM can adjust the intensity.

Timers.h defines the timer drivers.

TimBackLight(Timer15::eInputClk::Internal)

This configures it

TimBackLight.Prescaler(TimBackLight.FindPrescalerValue(5000'00)); // 1MHz

TimBackLight.Periode(999);

TimBackLight.Mode(1, Timer::eMode::PWM1); // set CCR1 to PWM1

TimBackLight.Compare(1, 200); // set CCR1 to xxx

The function

void Intencity(uint16\_t value)

{

if (value == 0)

{

TimBackLight.Stop();

return;

}

TimBackLight.Compare(1, value);

TimBackLight.Start();

}

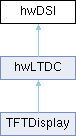
Handles intensity.

#### DSI and LTDC

Et billede, der indeholder tekst, skærmbillede, Rektangel, flashhukommelse

Automatisk genereret beskrivelseThe HW uses the DSI interface with 2 active lanes. To feed the DSI, STM32H7 have the LTDC interface.

The code is in hwDSI and hwLTDC



Look at the code, it took some hard work to get it working 😊 and is not fully tested to the corners.

TFTDisplay calls hwLTDC::Init to set it all up.

### Touch controller

To be continued