GD32VF103 series of RISC-V MCU quick start

By **Tam Hanna**

GigaDevice Semiconductor Inc. has been producing the GD32F103 microcontroller for a number of years now. It contains the popular Arm Cortex-M3 CPU but last year they threw their gauntlet into the embedded microcontroller arena by introducing the GD32VF103 device (note the 'V' in the description) which has a RISC-V CPU. We took a closer look at it.

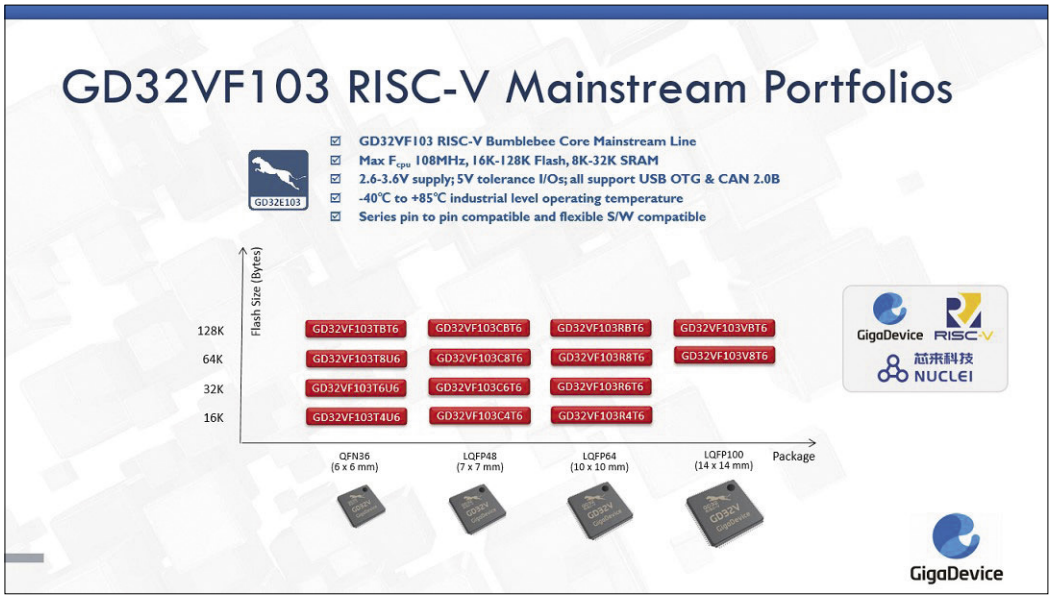

Figure 1: The RISC-V family is quite extensive and fully compatible with existing GD32 MCUs in software development and pin packaging. (Image source: GigaDevice)

GigaDevice added a new family of microcontrollers back in 2019. An important intention by GigaDevice in the design criteria was to promote a pain-free transition from other microcontroller architectures — the programming paradigms of the HAL are similar to those found for the GD32F devices. The current state of this family of devices is shown in **Figure 1**.

The RISC-V specification defines a standard ISA (Instruction Set Architecture), not an implementation for a specific MCU. The handling of exceptions, interrupts, peripherals and other niceties are left to the microcontroller developer's ingenuity. In the case of the GD32VF, GigaDevice decided to comply with the design specifications set by ARM designs.

Even if this does not lead to 1:1 code transferability (see [ARM2]), the learning curve is not quite so steep for those with some previous background experience. For this reason alone, we recommend taking a serious look at the system.

Setting up the tools

GigaDevice offers developers a very comfortable IDE — the command line level manipulation required by other providers is not necessary here. A small trap is waiting at this point though. You may be tempted to go to https://www.nucleisys.com/download.php and download the following packages individually:

- OpenOCD 64
- RISC-V Win32
- Nuclei Studio

At first glance this seems like a reasonable approach but in practice you end up with mismatches between the toolchain and IDE components. The correct version of the IDE and all its associated components can all be found at:

www.nucleisys.com/upload/file/2019/10/Nucleistudio/NucleiStudio_IDE_201909.rar

which also includes the toolchain. Extract the archive into the folder:

C:\NucleiStudio_IDE_201909

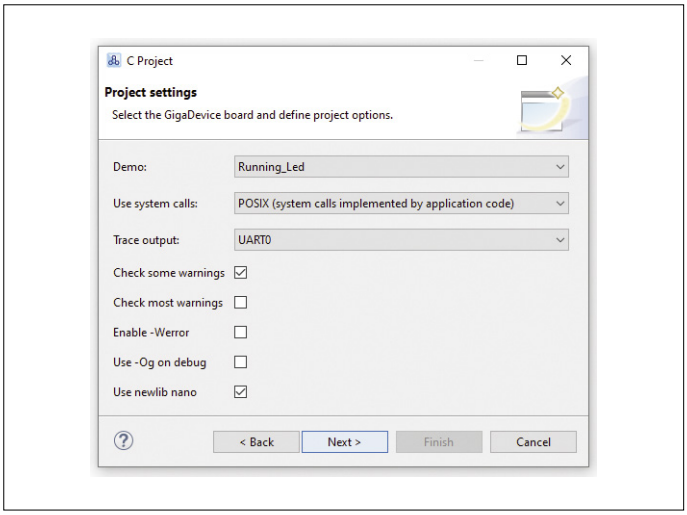

Figure 2: NucleiStudio configures some peripherals by default.

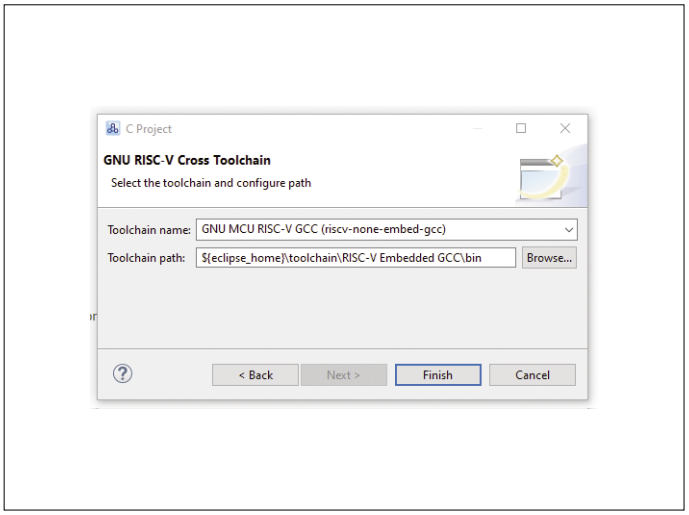

Figure 3: The default toolchain settings work here.

This is not required if using the GD-LINK debug probe which is hard soldered on all Gigadevice development boards; if you don't have a supported version of Java, you will find one in the archive. The author recommends you put the folders NucleiStudio and SerialDebugging_Tool directly in a subdirectory of C.
Start the IDE by clicking

C:\NucleiStudio_IDE_201909\NucleiStudio\eclipsec.exe

Then click File New Project and choose the template C/C++ C Project. The project intended for our MCU is named GigaDevice RISC-V Project; names can be assigned more or less arbitrarily. In the MCU selection we select the GD32VF103 device; the remaining settings remain as shown in **Figures 2 and 3**.
Make sure to minimize the welcome window by double-clicking its header. The next step is to click the hammer icon to start the first compilation process.

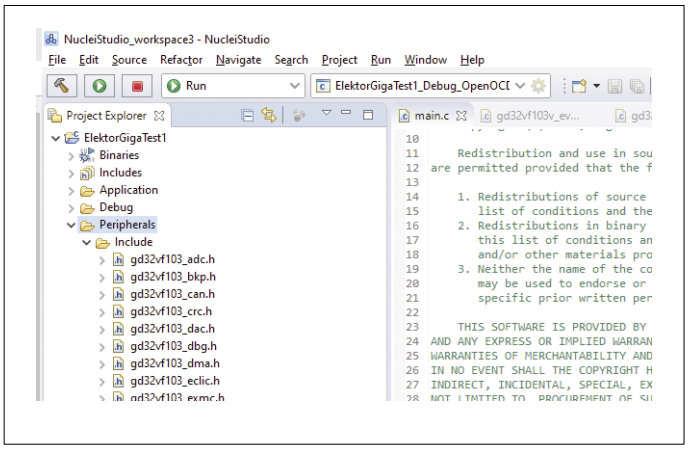

Figure 4: These controls replace the debug tools provided by Eclipse.

Eclipse is everywhere

At this point we can begin by running the example program. GigaDevice offer various evaluation boards to support their devices. For the purposes of this product review I will use the basic GD32VF103C-START board.
Connect the board to the workstation via the the GD-Link port. No external debugger and/or programmer is required as all our boards include the on-board GD-Link Debug Probe. If software has never been installed on this board before, LED1 will light up. The GD32VF103C-START and the Nucleo development boards from STM are quite similar. The GigaDevice board includes two Mini USB cables and various pin header strips which need to be soldered in place.
The small evaluation board is limited to the standard Arduino header pinout assignments and does not use the Morpho connector provided on the STM boards.
GigaDevice provides support in the same way Nuclei Studio generates an execution template for the newly created project skeleton - on the author's workstation it is named ElektorGigaTest1_Debug_OpenOCD. Note that the program must not be executed using the normal tools included with Eclipse.

Messages in the display window indicate the system status. The most important one at this stage of the setup is:

```
Info: JTAG tap: auto0.tap tap/device found:
   0x790007a3 (mfg: 0x3d1 (GigaDevice Semiconductor
   (Beijing)), part: 0x9000, ver: 0x7)
```

This indicates that the target device has been recognized and confirms that the USB communication link is working correctly. The code provided in the example is intended for use with the full-function GigaDevice evaluation board — if you use the small board only one LED is available to be turned on and off on the board. Now it is possible gain better insight into the API by right-clicking parts of the sample code and using the reflection function available in Eclipse.

What's what?

At this point, we open the main.c file and look at the entry point of the RISC-V software. GigaDevice — unlike Arduino — doesn't have a framework library so execution starts with the main function:

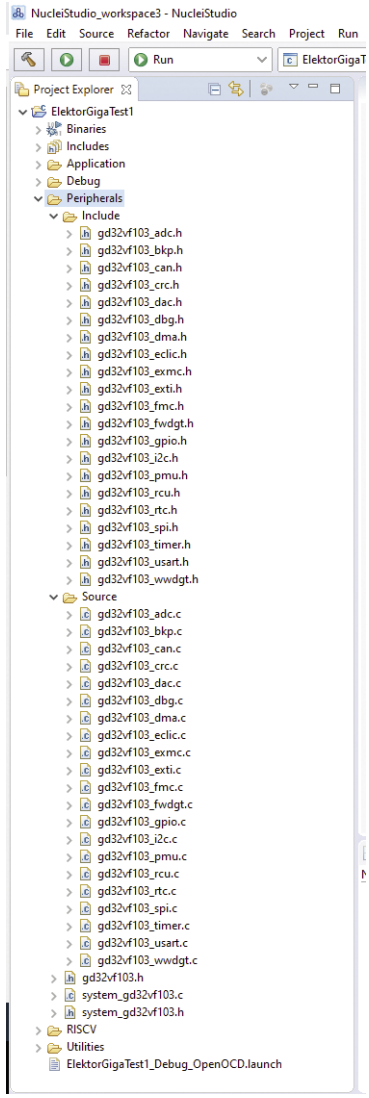

Figure 5: The peripherals subfolder contains the HAL library source code.

```
int main(void)
{
    gd_eval_led_init(LED1);
    gd_eval_led_init(LED2);
    . . .

    while(1){
        /* turn on led1, turn off led4 */
        gd_eval_led_on(LED1);
        gd_eval_led_off(LED4);
        delay_1ms(1000);
        . . .
```

In this fragment of the code we can see the first step, the 4 GPIO pins connected to LEDs are initialized and then switched on and off in an infinite *while* loop with a 1 second delay inserted each time two of the LEDs are switched. The larger evaluation board is fitted with four LEDs, which is why the supplied code example is larger than it needs to be for switching the LED on the more basic starter board I am using here.

Where is my CUBE?

The graphical code generator tool CubeMX used by Atollic and Co is not currently supported by GigaDevice. It will therefore be necessary for the developer to take care of configuring all the peripheral devices.

Also interesting is the *delay_1ms()* statement – it specifies a timebase, independent of the processor clock so that code written to run on a slower processor will not need to be changed to maintain the same software timing delays. The delay required is passed as a numeric parameter.
Constants like LED1 are defined in the file gd32vf103v_eval.h, which is intended for the larger board. The three elements corresponding to the GPIO port used with our board are as follows:

```
#define LED1_PIN                    GPIO_PIN_0
#define LED1_GPIO_PORT              GPIOC
#define LED1_GPIO_CLK               RCU_GPIOC
```

Let's first look at gd_eval_led_init, which takes care of the evaluation board initialization. Here we make sure the peripheral device is connected to the clock generator. This is followed by gpio_init, where the attributes of the LED GPIO port are assigned:

```
void gd_eval_led_init(led_typedef_enum lednum)
{
    /* enable the led clock */
    rcu_periph_clock_enable(GPIO_CLK[lednum]);
    /* configure led GPIO port */
    gpio_init(GPIO_PORT[lednum], GPIO_MODE_OUT_PP,
              GPIO_OSPEED_50MHZ, GPIO_PIN[lednum]);
```

The value GPIO_OSPEED_50MHZ ensures that the output pin slew rate can support the output speed. For applications where timing constraints are less critical a slower frequency can be assigned to reduce the generation of switching RFI. The next statement resets (clears) the pin to make sure the program begins with the LED off:

```
    GPIO_BC(GPIO_PORT[lednum]) = GPIO_PIN[lednum];
}
```

The GPIO_BC macro gets us deeper into the framework. Switching on and off the LEDs on the evaluation board is done by the following two methods, which are basically limited to un-wrapping the data arrays using the LED as an index:

```
void gd_eval_led_on(led_typedef_enum lednum) {
    GPIO_BOP(GPIO_PORT[lednum]) = GPIO_PIN[lednum];
}
void gd_eval_led_off(led_typedef_enum lednum) {
    GPIO_BC(GPIO_PORT[lednum]) = GPIO_PIN[lednum];
}
```

Figure 6: The RISC-V core is a real powerhouse.

Their declaration can be found in the file gd32vf103_gpio.h.

The two register architecture known from other microcontrollers (one for setting bits and one for resetting bits) is also used here:

```
#define GPIO_BOP(gpiox)              REG32((gpiox) +
   0x10U)    /*!< GPIO port bit operation register */
#define GPIO_BC(gpiox)               REG32((gpiox) +
   0x14U)    /*!< GPIO bit clear register */
```

If you want to learn more about the GPIO-API, right-click the editor and select the header and source switch option in the context menu. You will find yourself in gd32vf103_gpio.c, which contains the full interaction methods and their implementations. More information can be found by taking a look at the hardware support library. As shown in **Figure 5**, their source code is located in a subfolder of the project skeleton.

To test how fast the output pin can be toggled we can build an endless loop in software. In order to reduce time spent making subroutine calls the loop is made up of just the bit set and reset instructions. The toggled pin (5) is initiated as a push-pull output and the output speed (slew rate) set to 50 MHz mode:

```
int main(void)
{
    rcu_periph_clock_enable(RCU_GPIOA);
    gpio_init(GPIOA, GPIO_MODE_OUT_PP,
          GPIO_OSPEED_50MHZ, GPIO_PIN_5);
    GPIO_BC(GPIOA) = GPIO_PIN_5;
    while(1){
       GPIO_BOP(GPIOA) = GPIO_PIN_5;
       GPIO_BC(GPIOA) = GPIO_PIN_5;
    }
}
```

Here the GPIO port is initialized and then the pin is continually set and reset in the *while* loop via the two registers mentioned above.

Note this is an endless loop and you will need to click on the red stop button symbol before attempting to re-run any new revision of the test code. The output is toggling at high frequency so it's necessary to keep all connections to the signal pin as short as possible and be aware of the scope probe capacitance. The output waveform can be seen in **Figure 6**.

Arm wrestle RISC-V

Open-source and royalty-free architectures such as RISC-V promise to bring a breath of fresh air to the microprocessor marketplace currently heavily dominated by Arm devices. This is good news for product developers and start ups that often don't have pockets deep enough to fund licensing fees necessary to build Arm-cored system-on-chip (SoC) designs.

Thanks to its similarity to existing layouts the GigaDevice RISC-V range of processors is a good start if you are keen to get some hands-on experience with the technology. From a software developer's point of view, the HALs are so similar that little effort is involved in changing to them.

The bottom line

Given all the advantages of the RISC-V controller it will no doubt be interesting to explore the capabilities of this device in an embedded environment. Whether it's just to satisfy your own curiosity or for more commercial reasons the devices from GigaDevice provide a good introduction to the technology.

An interesting development for the maker community is the recent appearance from Longan of their Nano board; it is of similar size to the STM Blue Pill development board but uses a GD32VF103CBT6 32-bit RISC-V humming along at 108 MHz while sipping just one third the power of an Arm Cortex-M3 core. All this for less than $5 (including a 0.96" 160 x 80 IPS RGB LCD). One thing's for sure; at this price and performance level, we can expect to see many more low-cost development boards and applications based on the RISC-V architecture entering the marketplace quite soon. ⏮

200094-01

Web Links

[1] GigaDevice Product Selection Guide: www.gigadevice.com/wp-content/uploads/2019/06/GigaDevice-Product-Selection-Guide.pdf

[2] GigaDevice GD32 Development Tools: www.gigadevice.com/products/microcontrollers/gd32-development-tools/