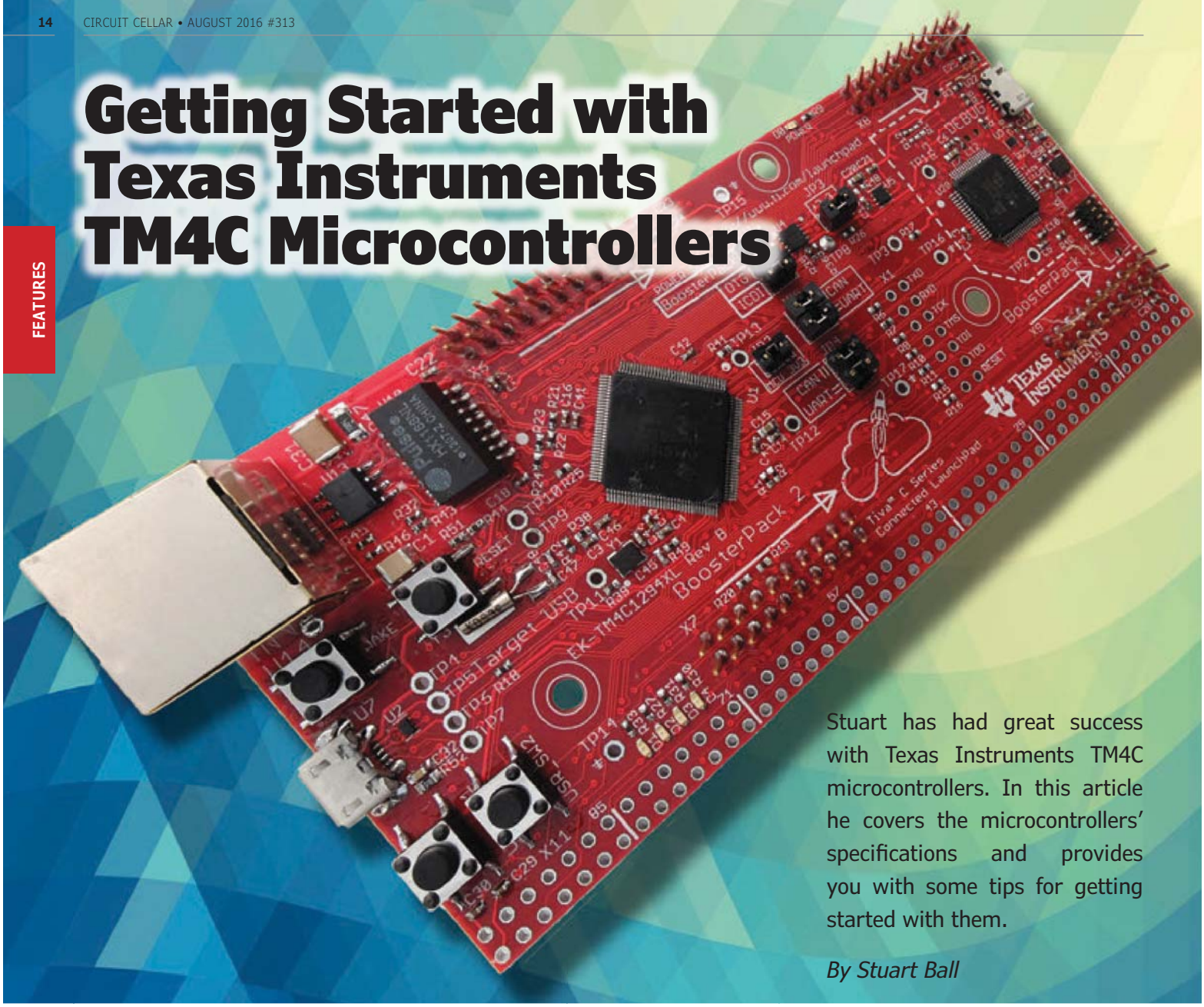
FEATURES

Getting Started with Texas Instruments TM4C Microcontrollers



Stuart has had great success with Texas Instruments TM4C microcontrollers. In this article he covers the microcontrollers' specifications and provides you with some tips for getting started with them.

By Stuart Ball

I've been experimenting with the Texas Instruments TM4C microcontrollers. This family of ARM-based parts offers a number of features and options, and perhaps my experience in getting started with them will benefit you.

THE TM4C LINE

There are two groups of parts in the TM4C line, the TM4C123x and TM4C129x. The TM4C129x parts generally have more capability; the TM4C123x parts are generally lower cost. TM4C123x and TM4c129x features include:

- ARM Cortex M4F processor at 80 MHz (TM4C123x) or 120 MHz (TM4C129x)
- 3.3-V operation (1.2-V core with on-chip regulator)
- JTAG
- Internal precision oscillator
- Support for external 32-kHz crystal and external megahertz-speed crystal
- Internal PLL
- Internal flash memory, SRAM, and EEPROM
- Internal ROM with peripheral driver functions
- Internal ROM supports download on startup
- UART, SSI, I²C, CAN, USB, Ethernet, LCD, and 1-Wire interfaces
- Multiple crypto modules, PWM, ADC, and DMA

Not every device has every feature, of course. The TM123x controllers have fewer features and less memory, in general, than the TM4C129x parts. For example, crypto modules and Ethernet and LCD interfaces are only available on the TM4C129x parts. Like most microcontroller families, there

are different parts optimized for low cost, low power, and high performance. There are over 40 parts in the product line, more if you include different packaging options such as BGA versus TQFP.

ROM-BASED BOOTLOADER

Most of the features of the TM4C are similar to other microcontrollers. But one of the most intriguing things about the TM4C series parts is the ROM bootloader. The ROM bootloader will monitor one USB port (USB 0), one of the UARTs (UART 0), SSI 0, I^2C 0, and Ethernet (if available) for connection to a host, and it can download firmware into the flash from any of those interfaces.

If there is no code loaded into the flash memory, the ROM bootloader will run by default. But one really useful feature of the ROM bootloader is that you can program a permanent register in the flash memory so that when a specific GPIO pin is low or high (programmable), the ROM bootloader will execute instead of the application code, even if application code is loaded. This lets you use the ROM bootloader even when you have code programmed into the part, just by grounding the specified GPIO port pin with a shunt jumper or switch.

Selection of a GPIO pin to enable the bootloader is done by writing to the BOOTCFG register. There is a field in the BOOTCFG register to select which port will be used to enable the downloader, and another field to select which bit of the port will be used. Another bit selects whether high or low on the selected GPIO pin will enable the ROM bootloader.

The BOOTCFG is a nonvolatile configuration register, so a specific process has to be followed to write it. Refer to the datasheet for the part you want to use. Note that the field to select the GPIO port is only 3 bits, so some ports are unavailable for enabling the ROM bootloader since there can be more than eight GPIO ports on the device. Also, obviously, if you have a device that does not support all the ports (such as a 64-pin device), you can't use a GPIO port that isn't there. If you program the BOOTCFG register incorrectly, a reset sequence using the JTAG pins is needed to clear it. The reset sequence is described in the datasheet under the JTAG interface section.

ROM-BASED DRIVERS

The TM4C parts also have driver firmware in ROM to control the peripherals. This includes configuring the GPIO ports as GPIO or alternate functions, sending and receiving data over the interfaces, and clock/system control functions. When you write your code, you can specify whether the ROM-based functions are used, or whether the functionality will be added to your application in flash. Using the ROM-based functions reduces the amount of flash memory used by your program.

DEVELOPMENT

Texas Instruments provides an integrated development environment, Code Composer Studio (CCS), that supports C/C++ language development. The TM4C parts are also supported by Keil and IAR, among others. TI also provides a utility, LM Flash, for downloading code to the parts. LM Flash will support the ROM bootloader download interfaces mentioned earlier. Conveniently, this includes the UART. You do need an RS-232-to-3.3-V level translator, of course, to use the UART.

Texas Instruments provides a software suite, TivaWare. This includes a peripheral driver library that has functions for most common things you will want to do with the peripherals. This includes enabling and configuring peripherals, sending and receiving data to/from peripherals, and so on. Functionality of the driver functions matches the capability of the peripherals. For example, the full functionality of the Ethernet media access controller (MAC) and USB interfaces is supported by TivaWare.

The TivaWare library is a separate download from the Texas Instruments website. TivaWare includes the capability to select and call the ROM-based library functions, as well as incorporate the equivalent functions into your flash code. Texas Instruments also has a free RTOS, TI-RTOS, which is compatible with the TM4C parts, although I have not used it.

LAUNCHPAD BOARDS

Texas Instruments provides a few low-cost development boards, some under $20, to support TM4C development. These are simple boards with switches, LEDs, and the ability to download code. You can also use the Launchpad boards to program your own custom boards via JTAG and to reset the device if you get it into a state where the ROM bootloader doesn't recognize the download GPIO pin. (I've done that.)

In addition to the Launchpad boards, Texas Instruments offers BoosterPacks with various features such as ultrasonic ranging, motion control/sensings, temperature/humidity sensing, and QVGA display drivers. The entire infrastructure is similar in concept to Arduino, except, of course, that it's powered by the ARM Cortex M4F processor. Note that there are boards called BoosterPacks that are intended for Texas

Instruments development boards based on different microcontrollers. So if you are interested in these, you want to select boards that are compatible with the TM4C LaunchPad boards.

Visit the Texas Instruments website for a list of BoosterPacks. You can sort by those that are compatible with the TM4C LaunchPad boards. The universe of BoosterPacks is smaller than the number of available Arduino shields, of course, but the point is that many functions you might want to get started, are already available.

There is also an open-source platform called Energia (www.energia.nu) that makes LaunchPad boards even more Arduino-like, using the Wiring platform. I have not used this, so I will only point out the availability of it. Go to the Energia website for more information.

START DEVELOPMENT ON TM4C

The easiest way to get started with the TM4C, of course, is to purchase a LaunchPad board. Unlike some development boards, these are inexpensive and capable enough to do useful things. The boards bring out unused GPIO pins to headers so you can connect other boards, including the BoosterPack boards already mentioned. If you want to start with a LaunchPad board, there are user guides, quick start guides, and other documents that you can use to familiarize yourself with the available boards and their capabilities.

I ordered a LaunchPad, but I generally find that I get more learning from a new processor if I build something with it. I put a TM4C part on a TQFP breakout board that I ordered from a seller on eBay and connected some LEDs and switches and a few other things to I could test various features.

PERIPHERAL COMPLEXITY

If you are used to working with microcontrollers like the Atmel AVR parts, the TM4C may seem daunting. The peripherals for the TM4C are more complex, with more features. TM4C peripherals can be powered off and disabled to reduce system power, so they must be powered and enabled in the firmware before they can be used. To give an example of the difference, here is what you have to do to initialize an asynchronous serial port on an Atmel AVR microcontroller:

- Enable serial port (alternate pin functions of GPIO port)
- Write data rate registers to set baud rate (data rate)
- Configure serial port (parity, bits per word, etc.)

This typically can be done by writing three or four registers.

Contrast that with the steps needed to enable a serial port on a TM4C part:

- Enable and power on the GPIO peripheral
- Enable and power on the UART peripheral
- Configure the GPIO pins as UART pins
- Select UART clock source
- Configure UART parity, bits per word, etc.

These steps require calls to at least four different functions in the peripheral driver library, which in turn write to multiple registers. Plus, when a peripheral is powered on and enabled, you have to wait for it to be ready before you can use it (this only takes a few clock cycles).

In both devices, the UART pins are shared with GPIO pins and must be configured to function as the alternate usage (UART instead of GPIO). But the increased capability of the TM4C peripherals requires additional steps to fully configure the peripherals.

An example with even more contrast is configuring a timer to function as a simple rollover counter that is polled in the firmware. On the AVR, you can do this by writing one register (in the simplest case). It requires calling five functions in the TM4C peripheral library. Again, increased capability means increased configuration complexity.

PIN CHARACTERISTICS

Drive Level: The TM4C outputs default to 2-mA drive level. If you need more output current than that, such as might be needed to drive an LED, you have to reconfigure the pin to support higher current. Depending on the part you are using, the drive level can be configured to be 4 or 8 mA (on the TM4c123x), but the default is 2 mA. The pins can drive more than 8 mA, but the output voltage rises and the number of pins you can drive with higher current is limited. The TM4C129 parts have a default drive level of 2 mA as well, but the upper limit can be configured as 12 mA. As with the TM4C123x parts, the TM4C129x parts can exceed this maximum limit (up to 18 mA) on a limited number of pins.

NMI Pins: Generally, the GPIO pins default to be GPIO functionality and must be reconfigured for alternate functionality. But some pins default to NMI functionality. For example, Port F bit 0 and Port D bit 7 on the TM4C1233D5PM default to non-maskable interrupt (NMI) pins and must be unlocked and reconfigured to be used as a GPIO. This might seem minor, but if you do searches on the

www.embeddedARM.com

DON'T LET A NATURAL DISASTER BECOME A TOTAL DISASTER

Weather happens.
When it does, you can take comfort knowing your critical embedded system is taking care of itself when a power outage strikes.

Our TS-SILO equipped embedded systems offer up to a minute of operation after total power loss, time which can be used to gracefully shutdown, maintaining file system integrity.

Contact Technologic Systems at 480-837-5200 or visit us at www.embeddedARM.com

Single Board Computer

Starting at
$159
Qty 100

TS-7680
Low Power Industrial
Single Board Computer with
AC Power, Relays, Bluetooth and WiFI

The TS-7680 single board computer is the first product to feature TS-SILO technology. It's designed to provide not only extreme performance with low power and high reliability for the most demanding of applications, but also plenty of industry standard, rugged interfaces for easy integration and fast time to market.

Specifications include:

- NXP i.MX286 454 MHz ARM CPU
- 2 GB SLC or 4 GB MLC eMMC Flash Storage
- Wireless 802.11 b/g/n and Bluetooth 4.0 EDR
- Optional TS-SILO to provide 20-60 secs of power hold
- 24-position rugged screw terminal connector
- Power input allows 8 to 40 VDC and 10 to 28 VAC

DESIGNED ASSEMBLED 30 YEARS TESTED

CERTIFIED ISO 9001:2008 COMPANY

Technologic Systems

Internet, you find a lot of people apparently are tripped up by this. The specific pins are different on different devices, but the point is that NMI pins must be unlocked for use as normal GPIO pins. This requires writing a specific key value to a GPIO lock address and then writing the pin map to the GPIO commit register. The JTAG pins can be used as GPIO pins if JTAG is not needed. A similar unlock sequence is needed to enable GPIO functionality on those pins.

Crystals: The TM4C has an internal phase-locked loop (PLL) to multiply an external crystal to a higher frequency for the CPU and peripherals. The CPU can run at the frequency of the internal precision oscillator, an external crystal, or a multiple of the input clock.

The TM4C123x parts are intended to work with specific crystals, and the peripheral driver for the PLL and clock system has a list of 18 supported crystal frequencies, ranging from 5 to 25 MHz. This might seem limiting, but for operation of the USB port, only certain frequencies will provide the correct clock.

On other microcontrollers, I have used a crystal that is a multiple of the standard data rates, such as 14.745 MHz, to get precise baud rates for the UARTs. The TM4C123x parts do allow for a few standard data rate crystals, but it isn't necessary. The UARTs have fractional data rate dividers, so obtaining a precise data rate clock is not an issue. You can use a crystal, such as 12 MHz, that provides precise USB timing and precise data rate timing as well. This solves a problem that exists on some other microcontrollers, where you can either use USB or you can have an accurate data rate clock. On the TM4C parts, you can do both. The TM4C peripheral driver library has a setup function that is called with the system clock frequency and the desired data rate, and it configures the registers for you. No calculations needed.

The TM4C129x parts have a more traditional PLL configuration where you can directly program the PLL multiply/divide registers, instead of limiting you to predefined crystal frequencies. On both the TM4C123x and TM4C129x, the PLL is disabled at power-up or reset, and must be enabled and configured to be used. At power-up/reset, the part operates from the internal precision oscillator until the external crystal is selected and/or the PLL is configured and enabled.

Timers and PWM: The TM4C parts have multiple 32- and 64-bit timers. The 32 timers can be split into two 16-bit timers and have simple PWM capability. I split a 32-bit timer into two 16-bit timers and used one for an internal timer tick, and the other to generate an external clock for a peripheral, derived from the CPU clock. You could use this to drive an external charge pump, switching regulator, SPI/I2C peripheral, or anything that needs a clock or rudimentary PWM signal. Similarly, the 64-bit timers can be split into two 32-bit timers.

Some of the TM4C parts have full PWM modules, capable of multiple PWM outputs, dead-band generators for driving H-bridges (to prevent shoot-through), and other features. The PWM counters can be driven by the CPU clock or by a selectable binary sub multiple of the CPU clock.

SSI/I2C Ports: The parts include multiple SPI/SSI/I2C ports. The ports can be configured to operate as TI synchronous serial interface (SSI) ports, with word sizes up to 16 bits. I configured two of these ports to operate as inter-processor communication interface between two TM4C parts. The ability to select different word widths makes efficient use of this for simple transactions; the upper bits can be opcodes and the lower 8 bits can be byte-wide data. On many microcontrollers, the UART pins are shared with the SPI/I2C pins, so you can't use both at the same time. The TM4C parts give you SPI/I2C ports on different pins (for at least some of the ports).

Some of the I/O ports can be mapped to more than one pin. For example, on the TM4C1233D5PM, SSI 1 can be mapped to pins on port F or on port D. Similarly, on some devices a UART can be mapped to pins on two different ports. I haven't tried mapping the same peripheral to pins on two ports, so I don't know what would happen, but the datasheets caution against it. You've been warned.

circuitcellar.com/ccmaterials

RESOURCES

S. Cozart, "Differences Between Tiva C Series TM4C Microcontrollers," SPMA065, Texas Instruments, 2013, www.ti.com.cn/cn/lit/an/spma065/spma065.pdf.

K. Krakow, S. Johnson, and J. Guy, "System Design Guidelines for the TM4C123x Family of Tiva C Series Microcontrollers," SPMA059, 2013, www.ti.com/lit/an/spma059/spma059.pdf.

Texas Instruments, "Tiva C Series Development and Evaluation Kits for Code Composer Studio," SPMU352, 2013, www.ti.com.cn/cn/lit/ml/spmu352/spmu352.pdf.

———, "Tiva C Series TM4C123x: ROM User's Guide," SPMU367, 2014, www.ti.com/lit/ug/spmu367/spmu367.pdf.

———, "TivaWare Peripheral Driver Library," SW-TM4C-DRL-UG-2.1.1.71, 2015, www.ti.com/general/docs/lit/getliterature.

ENVIRONMENT SETUP

There are a few things that I learned the hard way in setting up the environment. The comments that follow all apply to the free CCS software provided by Texas Instruments. I haven't used tools from the other vendors.

You need to add the TivaWare peripheral driver library includes into the CCS environment. If you want to use the ROM-based functions, you have to include the ROM header files in the code, and you need to include the ROM library in the linker include path.

Most of the peripheral functions in the TivaWare library have both ROM and flash-based versions. To specify the ROM-based version, you prefix the function invocation with `ROM_`. But you don't always know if the specific function you want is available in the ROM. So if you use the MAP_ prefix instead, the compiler/linker will use the ROM version if it is available, and the flash version if not.

There is a `#define` that must be included to use the ROM libraries, and it tells the compiler what processor is being used, so it knows how to map ROM-based calls. If you use the `MAP_` prefix without this, the flash version of the functions will be selected because the compiler doesn't know any ROM functions are available. An example looks like this:

```
#define TARGET_IS_TM4C123_RB1
```

The `RB1` part of that (there is a TM4C129 define instead if you are using a TM4C129 part) depends on the specific silicon revision you are using. The Silicon Errata document describes which revision you have. This is the one thing I don't like about these parts. If the silicon revision changes, you might need to change your code. However, since the ROM functions are accessed via a table in the ROM, many if not most silicon changes won't affect compiled code. But be aware that this `#define` is needed and that it must be specified either in CCS or in the source code, prior to including the TivaWare ROM functions. Of course, if you don't need the space in flash, you can use the flash-based functions instead of the ROM-based functions.

If you want to create binary files that can be downloaded via the serial port, you need to add a line to the CCS Build post-link steps to invoke the binary converter to convert the .out file from the linker to a .bin for LMflash. I included the line I used here. Adjust as needed. This all has to be on one line:

```
${CCE_INSTALL_ROOT}/utils/tiobj2bin/tiobj
    2bin""${BuildArtifactFileName}"
${BuildArtifactFileBaseName}.bin"
"${CG_
    TOOL_ROOT}/bin/armofd" "${CG_
TOOLROOT}
    /bin/armhex"
"${CCE_INSTALL_ROOT}/utils/tiobj2bin/
mk
    hex4bin"
```

Like the issue of PF0 configuration, this may seem very specific for a general article, but it took me a while to find this. Note that this could change when the CCS toolchain is updated by Texas Instruments.

RECOMMEND RESOURCES

The datasheets for the individual parts are over 1,000 pages, so most of us aren't going to read the entire thing before starting a design. But there are a number of other documents that provide useful information before starting a design. Refer to the Resources list at the end of this article. It isn't an exhaustive list of what is available, but it's a good place to get started.

Texas Instruments also has a forum where a lot of design and development questions get asked and answered. Go to e2e.ti.com and select the TM4C microcontrollers forum. One of the first links is to a page titled "Diagnosing Common Development Problems and Tips & Info for TM4C Devices," which describes common issues. This includes the GPIO pin unlocking that I mentioned earlier.

GET STARTED

If you need to migrate to microcontrollers with more capability than 8-bit parts provide, or if you just want to experiment with ARM-based Cortex microcontrollers, the TI TM4C family is a good place to start. The ROM-based bootloader and the LaunchPad environment make a relatively painless way to get going. The free Texas Instruments toolchain is adequate and I haven't found it to be missing anything essential.

ABOUT THE AUTHOR

Stuart Ball is a registered professional engineer with a BSEE and an MBA. He has more than 30 years of experience in electronics design. He is currently a principal engineer at Seagate Technologies.