Technical Article

Step 3 to Build a Smart Thermostat Using an MCU – What to do with all the Data?

Britta Ruelander

Bhargavi Nisarga

You've already started to develop your own smart thermostat by implementing the sensing and measurement chain for your application. Now, let's get going with processing and logging the digital sensor data.

Assuming that you have your analog-to-digital converter (ADC) up and running to sample the measured data, you might wonder what options you have to further work with this data. This third installment of a seven-part series will touch on three steps:

1. How to transfer data from one memory address to another.
2. How to perform further data processing to get meaningful information for your application.
3. How to store the relevant data.

Figure 1 shows a simplified microcontroller (MCU) block diagram with a central processing unit (CPU) and direct memory access (DMA) controller as the bus masters in the system.

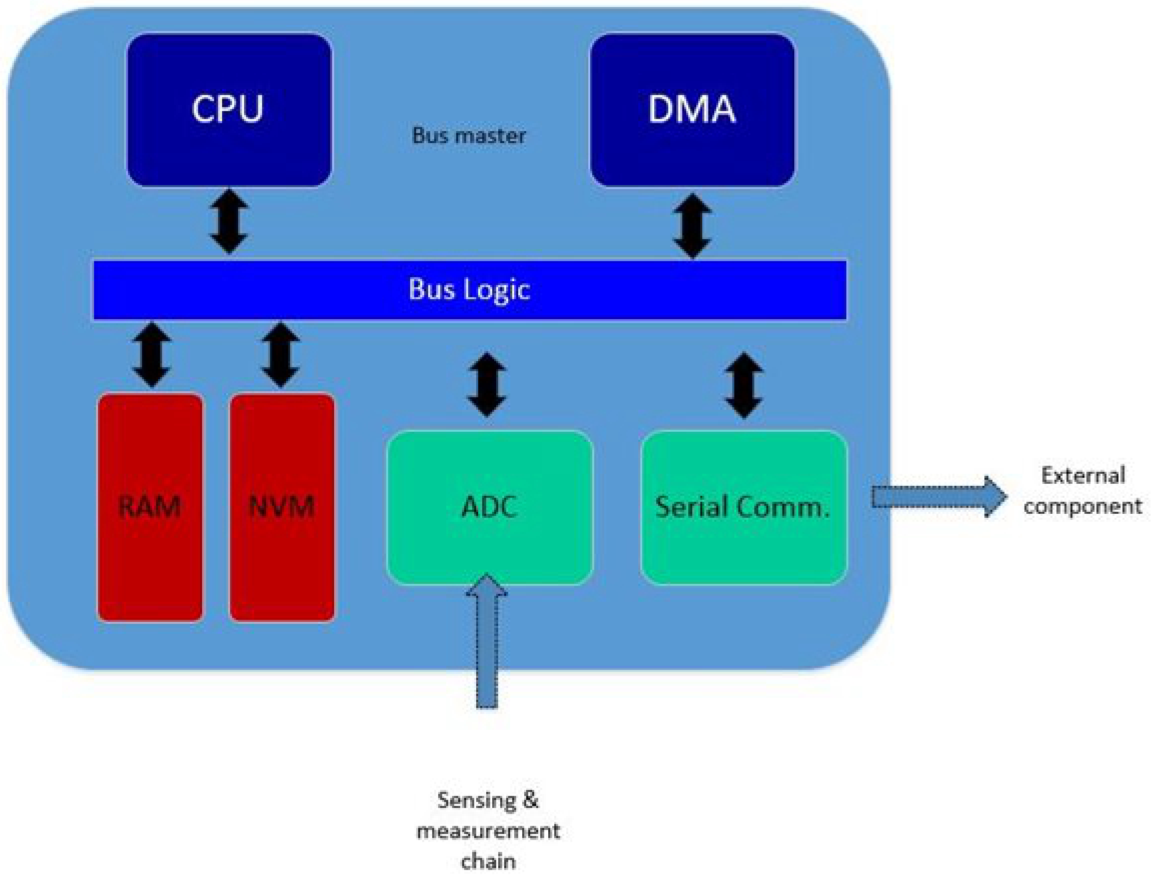

Figure 1. High-level Block Diagram of a Typical MCU

There are a few different data-flow scenarios involving the transfer of ADC data to on-chip and off-chip components:

- The ADC data is transferred via CPU or DMA to on-chip nonvolatile memory (NVM) storage.
- The ADC data is transferred via the CPU or DMA to an on-chip RAM. In this case, data can be further processed and then moved to on-chip NVM, or to the serial communication buffer via the CPU or DMA. Data can also move to external off-chip components via serial communication modules.
- The ADC data is transferred to a serial communication peripheral directly via the CPU or DMA.

In the case of our smart thermostat application, let’s focus on the second data-flow scenario, involving some data processing and averaging before sending the temperature measurement information to an external system, like a connectivity device that in turn connects to the cloud. On the other hand, if you want to collect a large amount of data to perform data analytics (for example, analyzing averages or minimums/maximums over time), you transfer measured data to an external system and then do the processing (such as cloud computing).

Now, let’s get started with the data transfer.

How to Move the (Conversion) Data

The second installment of this series provided information on conversion-mode options for ADC configuration. Most likely, you opted for the single conversion mode and thus have only one conversion data after every timer trigger. In typical MCU systems, you have two options to move the data from one memory address to another: by either using the CPU (Figure 2) or DMA (Figure 3). The advantage of using DMA is to perform the data transfer without involving (or even waking up) the CPU. Thus, you can save energy using DMA in a low-power mode while the CPU is sleeping.

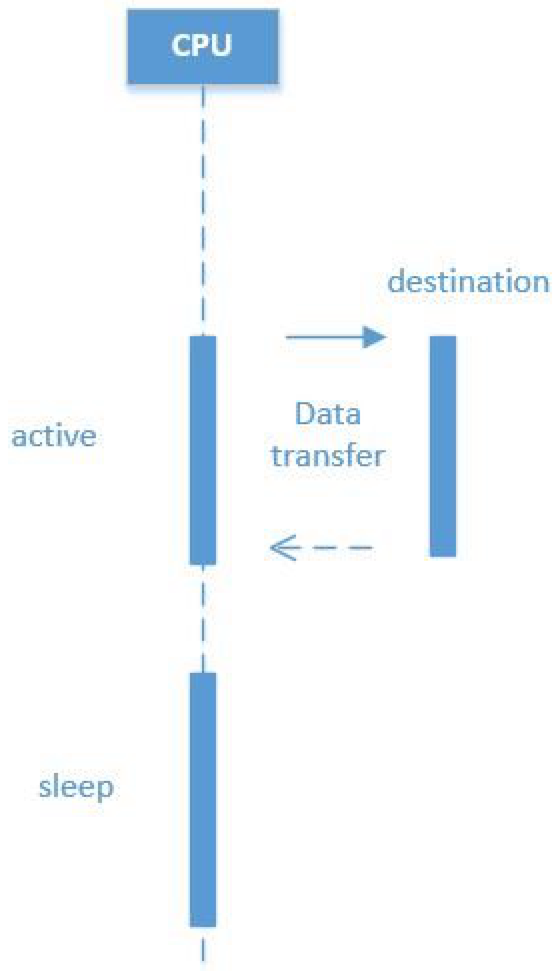

Figure 2. Data Transfer Executed by the CPU

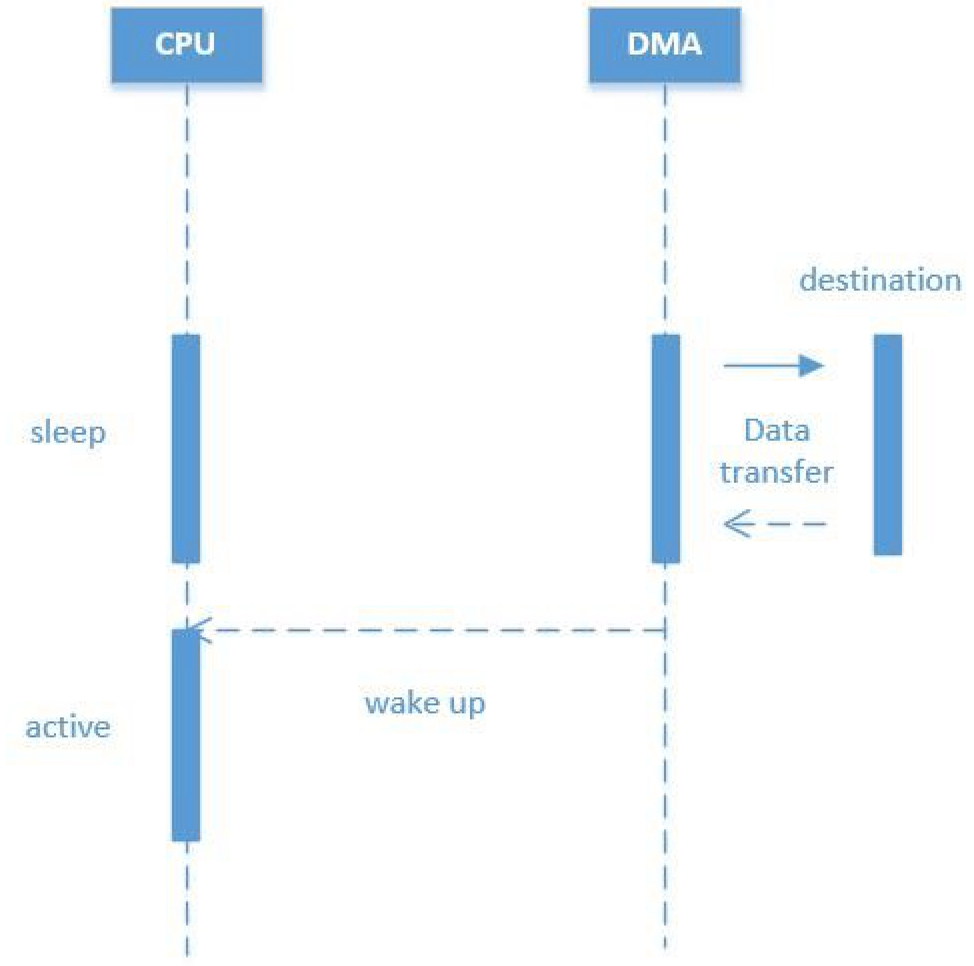

Figure 3. Data Transfer Executed by DMA While the CPU Is Sleeping

For smaller data sizes, DMA is not absolutely necessary, but it is still an option. You can opt to use the CPU or DMA to move the data to memory every time you take an ADC measurement. For transferring larger data sizes (for example, transferring all data collected in one day to a cloud or to external memory for backup), consider using DMA.

To configure the DMA, you need to set the right DMA transfer mode, distinguishing the transfer of a single data point or block of data and selecting whether it is acceptable to allow block transfer with CPU activity interleaved. Configure the size and width (byte/word/long word) of the data being transferred, plus the source and destination address of the transfer. Afterwards, just select the signal that will trigger the DMA transfer and you can start.

In a temperature-monitoring application, you will typically average multiple data points during measurement. In that case, you can use DMA to move the amount of data you want to use for the average and only wake the CPU after the transfer of all of the data points.

Smart Conditioning – How to Process the Data

The second installment of this series outlined various temperature sensors, with the final decision to go with the thermistor option. The ADC conversion results represent the thermistor analog values in a digital format. In order to retrieve a meaningful temperature value, you need to apply the voltage-to-temperature formula to the conversion result. Thermistors come in various sizes, are very sensitive to temperature changes and easy to interface with MCUs. However, since the thermistor response to temperature changes is very nonlinear, correlating the thermistor voltage to an actual temperature value entails either computing the nonlinear transfer function or using a look-up table (LUT)-based approach. The LUT-based approach requires using linear extrapolation between two temperature points in the LUT to interpret the temperature.

Thermistor data sheets typically include a temperature-to-respective thermistor resistance LUT for several temperature points. Using the table in the data sheet, for a given temperature value you can determine the actual analog voltage present at the ADC input channel. This analog input value correlates to a specific ADC code. The blog post, “It’s in the math: how to convert ADC code to a voltage (part 1)” discusses analog-to-digital conversion parameters in detail.

The LUT stored on the MCU will include a list of temperature points mapped to the respective ADC digital codes. Upon taking a specific temperature measurement using the on-chip ADC, the application firmware can determine between which two temperature points the measured digital code falls into within the LUT. Once you know this, you can extrapolate the temperature value between the two points using the point slope formulae. The point slope formulae involve multiply-and-divide operations that can be computationally intensive, requiring a lot of CPU cycles. Consider MCUs where some of the computations are handled in hardware, which then requires less CPU involvement and enables faster and more energy-efficient computation. For example, many MSP430™ MCUs have a hardware multiplier that can enable 8-, 16-, 24- and 32-bit operations and achieve multiplication results within a few clock cycles.

If required, you can also consider averaging the temperature values measured before using the LUT. For averaging values in the order of multiples of 2, consider using shift-right instructions, which are less computationally intensive.

How to Store Data without Losing It at Power Loss

In order to not lose data upon power loss, you will need to store it in NVM. MCUs typically offer embedded NVMs to store your code and data.

In most cases, MCUs offer flash memories to store your data when there is no power to the device. The data access – irrespective of if you want to read, write or erase – is blockwise and handled in blocks of 64B, 128B, and so forth, based on device. The main disadvantage is the limited number of writes/erases (dependent on technology between 3,000 and several 100,000 cycles).

TI offers microcontroller families with embedded FRAM NVM. These families offer faster memory access (especially when writing), unified memory access (that is, no blockwise access), lower energy consumption compared to flash writes (as charge pump is not involved for erase/write operations) and increased write/erase endurance compared to flash. Since FRAM memory has faster write access times and does not require erasing the memory before writing, it enables faster writes to store data during a power-loss event. Faster writes reduce the time that interrupts are delayed or even blocked in your system.

As a wrap-up, here are the key takeaways of the third installment of this series:

- In typical MCUs, you can either use the CPU or DMA to transfer data from one memory to another.
- Data processing on-chip can be computationally intensive. Consider hardware features and accelerators, which enable faster and more energy-efficient computations.
- On-chip and off-chip NVMs can store data in the system. If you want to store data without losing it at power loss, consider MCUs with FRAM NVM.

Stay tuned for the fourth installment of this series, which will cover the human machine interface.

Additional Resources

- Read the first and second installments of this series:
  - “How to build a smart thermostat using an MCU – 7 steps to make it happen!”
  - “Step 2 to build a smart thermostat using an MCU – It’s all about the sensing.”
  - "Step 4 to building a smart thermostat using an MCU – adding HMI."
  - "Step 5 to build a smart thermostat using an MCU – adding network connectivity."
  - "Step 6 to build a smart thermostat using an MCU: energy optimization."
- See the MSP430 MCU Users Guide for more details on how configuring DMA works.
- Check out the different types of TI temperature sensors.
- Get more details about FRAM and its benefits:

  - Understanding FRAM Technology
  - "MSP430 MCUs feature FRAM with a touch of security."
  - "What is compute through power loss?"
- Download this reference design: Intelligent System State Restoration after Failure with Compute Through Power Loss Utility