

Battery-Free Nonvolatile RAM

Do you have a RAM-hungry design in the works? If so, check out what Brian has to say about Ramtron International's integrated processor companions. They may be the perfect solution for your memory problems.

Today's microcontrollers come with a wealth of onboard peripherals, generally limited only by how much you want to pay for the device and how many pins are available on the chosen package. Current 8-bit microcontrollers are now available with increasingly large amounts of flash memory as well, so larger program code and the nonvolatile storage of configuration parameters are often not a concern anymore. A side effect of this increasing on-chip integration is the disappearance of parallel bus interfaces on many microcontrollers. This is generally not a big issue. If you have all of the necessary peripherals on-chip, why would you need the bus in the first place?

If there is one microcontroller commodity where I sometimes feel the pinch, it's RAM capacity. Granted, many 8-bit controllers are now sporting 1 to 2 KB of internal static RAM instead of the single 256-byte page that used to be the norm. This is generally adequate for the hardware stack and various other stacks and frames needed by C or Basic compilers. However, for many data acquisition and control programs, much larger amounts of RAM are needed for dynamic data and variables.

When you exceed the processor's internal RAM capacity, things tend to get complicated. Adding a standard static RAM is inexpensive enough, but it invokes other penalties. To start with, you'll lose at least 18 port lines for the memory interface (Address, Data plus Read and Write control lines). Also, you may have to slow down your MCU's clock rate to match the access times of inexpensive static RAM chips. Finally, you will also need to add an address latch "glue" chip to the design.

Another alternative is to upgrade to 16- or 32-bit processors (e.g., the ARM) that contain much more internal RAM. Then, of course, you have to tackle the learning curve of a new device, which may not be what you really want to do.

In this article, I'll introduce a new family of devices, Ramtron International's integrated processor companions (IPCs). These devices are different from the similarly named devices of the past, which either facilitated the connection of large memory arrays to microcontrollers or consolidated a useful variety of peripherals into one chip. Using this approach may solve some problems with your next RAM-hungry design.

FERROELECTRIC RAM CHIPS

Before discussing the IPC device, let's look at Ramtron's more general-purpose serial ferroelectric RAM (FRAM) devices. I stumbled upon the IPC devices after designing a Ramtron serial FRAM device into my current project. Until I found the Ramtron web site, I was beginning to think that there was no such thing as a serial RAM chip, apart from the tiny RAM found many serial real-time clock (RTC) chips.

FRAM is unique. It acts just like conventional RAM in that it can be written to and read from in less than 70 ns. Its endurance, or the number of times that it can be written to, is virtually unlimited, like conventional RAM. Actually, it is 10^{14} cycles. Ramtron did the math and determined that it would take 14 years of continuous writing to the devices (at a 15-MHz SPI clock rate) to reach this endurance limit. However, unlike either static or dynamic RAM, FRAM is truly nonvolatile (like EEPROM). When power

disappears, the data remains indefinitely (it's rated for 10-plus years). FRAM's fast write time (compared to EEPROM) is also advantageous because it's much less susceptible to noise during write operations, thereby vastly improving data integrity and reliability.

If you need fast FRAM access times, you'll have to stick with the parallel bus versions of the device that are available. However, for many applications, the SPI version of the device is plenty fast. Generally, it requires only a single additional port pin (i.e., Chip Select to interface the device) because the standard SPI lines (SCK, MISO, and MOSI) are often already present for any other SPI peripheral devices that are used.

At the maximum 15-MHz SPI clock rate of the device I used, it takes 0.5333 μ s to transfer a byte. The FRAM transfer protocol requires a 4-byte transfer for the first byte written and read, with each additional data byte written/read requiring only one additional byte transfer (assuming sequential access). You could, for example, transfer an 80-character string in and out of FRAM in 44 μ s (i.e., $(4 + 79) \times 0.5333 \mu$ s).

Ramtron correctly assumed that most designers don't want to reinvent the wheel, so it chose to make its serial FRAM chips pin-compatible with existing serial EEPROM chips. To make things even easier, it also made the command protocol basically identical to popular serial EEPROM chips. I took advantage of this to mount an Atmel AT25256 EEPROM on my board during development while I was awaiting the arrival of the Ramtron FM25256 FRAM part. When it arrived, it swapped in fine with no changes needed to the firmware.

Note that you can write to and read from FRAM devices without giving any thought to the page size, which is a consideration when using EEPROMs. Thus, it's possible to simplify and speed up the firmware when replacing an EEPROM with a FRAM, although such code changes aren't strictly necessary.

I needed about 20 KB of RAM storage for dynamic waveform tables in my project, and the design already included a high-speed SPI ADC and DAC. Therefore, adding an FM25256 (32 K × 8) FRAM device required only one dedicated MCU port line to implement. Although the nonvolatile aspect of the FRAM devices wasn't important in my design, the fast access time was essential. At 32 KB, the FM25256 has the largest capacity of the family. The lesser devices contain 512 bytes, 2 KB, or 8 KB, respectively.

Table 1 shows the three FRAM families that are currently available. In terms of speed, the parallel devices are clearly the winners, followed by the SPI devices. The I²C devices (with their two-wire, multi-party bus) require the least in terms of processor pins and wiring, but they're considerably slower in operation.

FRAM & PERIPHERALS

Ramtron obviously thought it made sense to integrate its serial FRAM with a suite of useful peripherals in one package. These combinations are what make up the IPC family. The most comprehensive devices are in the 31x family. They contain FRAM, an RTC, a power monitor, a watchdog, early power failure notification, a unique serial number, and several event counters. Table 2

Feature	31x Family	32x Family	FM30C256
Memory	4–256 Kb	4–256 Kb	256 Kb
RTC	Yes	No	Yes
RTC Alarm	No	No	No
Power monitor	Yes	Yes	Yes
Watchdog	Yes	Yes	No
Power failure notification	Yes	Yes	No
Serial number	Yes	Yes	No
Battery switchover	Yes	Yes	Yes
Event counter	Yes	Yes	No

Table 2—The three Ramtron integrated processor companion devices have various features.

Feature	Parallel	SPI	I ² C
Density	8 K to 128 K × 8	4 K to 256 K × 8	4 K to 256 K × 8
Package(s)	28-pin, 32-pin SOIC, PDIP	8-pin SOIC, DFN	8-pin SOIC, DFN
Access time	60–120 ns	Varies with bus speed	Varies with bus speed
Bus speed	—	5–25 MHz	1 MHz
VDD	3.0–3.6 V and 5 V	2.7–3.6 and 5 V	2.7–3.6 V and 5 V

Table 1—Parallel versions of FRAM devices closely resemble static RAM in terms of packaging and access speed. Both the SPI and I²C serial family devices are packaged like serial EEPROM, and are basically pin- and software-compatible with them.

outlines the features available in each of the processor companion families.

All IPC chips interface to a processor via the two-wire I²C bus, and the FRAM capacity varies from 512 bytes to 32 KB. Virtually all of these devices are packaged in 14-pin SOIC packages. Although it isn't a consideration in my work, these devices are available in a green version for RoHS compliance.

Like any I²C device, the processor companion must have a manufacturer-defined slave address to uniquely identify itself on the bus. The processor companion is unique in that Ramtron chose to partition the device internally into two parts: memory and companion (all of the other functions). Each partition has its own unique slave address. The reasoning behind this decision isn't immediately obvious, but it turns out that there is a good reason for doing it this way. As I alluded to earlier when I described the SPI FRAM version, the overhead involved in accessing sequential memory locations is only a quarter of what's needed for an isolated or random access because the command, high-, and low-order address bytes needn't be resent.

On the I²C bus, giving the memory its own slave address allows access to memory to be unaffected by intervening access to either the RTC or the other companion peripherals. This can streamline memory access considerably if these other peripherals are also accessed frequently.

As you can see in Figure 1, the IPC device contains two address lines: A0 and A1. These pins are basically device select pins. The level of the two pins must match the values of bit1 and bit2 in the slave address that you use to access the device via the I²C bus.

This allows you to connect up to four IPC devices on a single

I²C bus, but I doubt that more than one of these devices would ever be needed in practice. If more FRAM were needed than what's available in one IPC, it would seem to be more reasonable to add I²C FRAM-only devices from the FM24Cxx family.

Let's take a closer look

at each of the companion functions. As you'll see, some have interesting features compared to those implemented in other devices.

RTC FUNCTION

It's certainly possible to implement an RTC function totally in the MCU itself if you use a timer interrupt. Many of Atmel's AVR processors, for example, contain a low-frequency oscillator block that's designed specifically for a 32,768-Hz watch crystal to support this function. The Renesas H8 family goes one step further by incorporating a full RTC block on chip.

However, if this clock must continue to run during power failures (a pretty standard requirement!), provisions must be made for both battery-backed RAM and operating the processor in some sort of low-power mode that will keep the clock running under back-up battery power. For these reasons, it's often easier to use a dedicated, ultra-low-power RTC chip external to the MCU in order to implement the RTC function.

The IPC's RTC function operates in much the same way as stand-alone devices made by companies such as Dallas Semiconductor and Philips Semiconductors. It contains registers for seconds, minutes, hours, days, the date, the month, and the year. Unlike some other RTC devices, it contains a duplicate set of holding registers, so it's easy to read the current time by transferring the timing registers to this holding register bank, where it can be read without any ambiguity as to whether, for example, the seconds register rolled over after the minutes register was read. This is a common source of error on other devices if it isn't guarded against in the user's software routines.

Although the IPC contains nonvolatile RAM that will reliably store the time

information without any back-up power, it's still necessary to keep the clock counters running during a power outage. To this end, it is necessary to couple either a battery or a "supercap" to the device to keep the clock running. The processor companion is meant to work with either a NiCd or a lithium 3.6-V cell. It can be programmed to provide the necessary trickle charging for rechargeable cells. No series diodes for a battery switchover or charge current-limiting resistors are needed for the battery circuit (as is the case with some other available RTC devices).

All RTC devices require a 32,768-Hz watch crystal (sometimes called a tuning fork), and the IPC is no different. If your design calls for an accurate RTC, you may wonder why a common computer RTC circuit seldom keeps time as accurately as a cheap wristwatch that uses the same crystal as its time base. One reason lies in the frequency dependence with temperature inherent in the watch crystals. A wristwatch has a pretty good crystal oven working for it: you! Your body temperature is much more constant than the environment experienced by either an embedded processor or a PC.

Conventional RTC devices have no way to correct for this or even handle any initial frequency intolerances in the crystal itself, apart from the addition of a trimmer capacitor across the crystal. This must be adjusted basically by trial and error.

IPC, on the other hand, contains a nice frequency compensation scheme, which is digitally based. During calibration, the IPC can output a nominal 512-Hz signal (derived from the crystal) on one of its pins. Assuming the external MCU (or other device) can measure this accurately, it can calculate the deviation of this signal from 512 Hz and send out (using a look-up table) a correction byte that will correct the clock accordingly. This calibration can be done once or on an ongoing basis if necessary. The range of this digital calibration ranges from

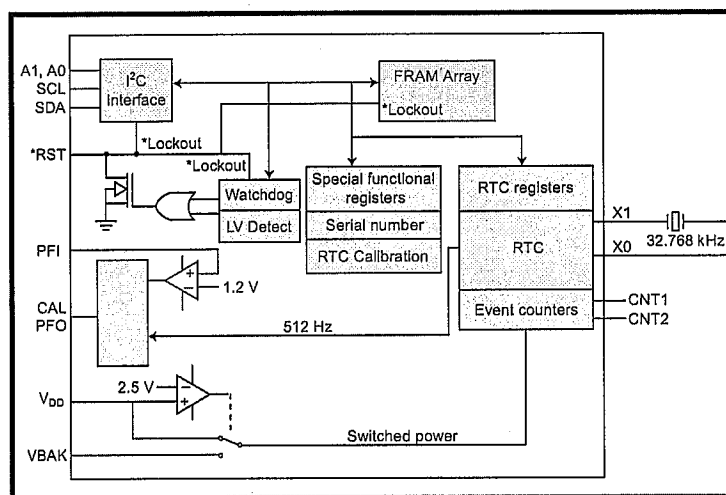


Figure 1—The IPC contains many useful peripherals in addition to its 512-byte to 32-KB FRAM array.

0.013% fast to 0.013% slow in 64 steps.

Generating an accurate 32,768-Hz clock signal, using a watch crystal is sometimes tricky in the presence of the other high-frequency signals commonly found in the proximity of a high-speed MCU. Such high-frequency noise can get mixed in with the legitimate 32,768-Hz clock and cause the RTC to run fast. I know from experience that PCB layout concerns are somewhat tricky when you try to do this using the on-board, low-frequency oscillator block in Atmel AVR chips. The same concerns apply with the IPC, but they are somewhat relaxed because there are fewer high-frequency signals around this device than there are with a full-blown processor.

POWER SUPERVISION

The processor companion contains a conventional power supervisor function that issues a *RESET signal when the V_{CC} supply falls below a given threshold. The actual threshold is programmable in four steps from 2.6 through 4.4 V using two bits (VTP0 and VTP1) in the companion control register.

The processor companion also contains an early power-fail comparator. It consists of a comparator with a fixed 1.2-V source on its negative input. Its positive input connects to your power supply's main storage capacitor through a resistive divider (ahead of the V_{CC} regulator circuit). Connected as such, it can sense the drop in power supply voltage arising from a power failure/brownout considerably sooner than it would

show up at the output of the V_{CC} regulator.

If you connect the comparator's output to the MCU NMI interrupt, it can give the MCU a reasonable amount of advance notice time to handle critical clean-up tasks before V_{CC} drops below the threshold value, which puts the MCU in a reset state. Some MCUs contain an uncommitted comparator, which can similarly be configured to perform this function.

WATCHDOG COMPANION

A dog is man's best friend, so it seems natural to include a watchdog as one of the IPC's companion functions. The IPC watchdog is reasonably versatile, with timeout intervals programmable in 100-ms steps from 100 ms up to 3 s. Like most watchdog circuits, this one is an RC-based timer, so its actual timing varies markedly with temperature, and the above ranges reflect the minimum time intervals. Like most watchdog timers, there is a watchdog enable bit that must be set to initiate watchdog operation. Another register bit group must be written with a prescribed value to restart the watchdog timer before it hits zero and causes an MCU reset by dropping its *RST line.

The watchdog seems, in some ways, to be less rigorous in its operation than the watchdog circuits built into many MCUs. For example, some MCU watchdogs require the MCU to write two distinct predefined values to a watchdog register within just a few machine cycles. Although this scheme is a fairly bulletproof watchdog solution, it may be difficult at times to achieve this timing from code generated by a high-level language compiler. Because the IPC's watchdog can be accessed by only a legitimate I²C transfer to the correct slave address, addressing the correct register, and using a prescribed value, it is still a robust solution that doesn't involve tricky timing specifications. Therefore, it may be easier to use than some built-in MCU watchdog solutions.

SERIAL NUMBER COMPANION

If your design calls for a unique serial number to be embedded in the electronic circuitry, it can be a bit of a nuisance to implement. Although most MCUs contain flash program memory and data EEPROM, the flash program memory is likely going to be gang-programmed (i.e., there's no differences from one unit to the next). The MCU's data EEPROM can be programmed with a serial number, but this requires some sort of individualized download/programming operation to be performed on each unit, which may not be feasible in some situations. An alternative to this that requires no individualized programming at the board level is to use the unique serial number lasered into Dallas Semiconductor's 1-Wire family of devices, for example.

At first glance, I thought that the Ramtron IPC family contained a unique 64-bit serial number. But that's not the case. The IPC's 64-bit serial number is strictly a block of eight nonvolatile FRAM bytes that can be written, as often as required, until the serial number lock bit is set. After that, those 8 bytes are readable but can't ever be altered again.

Effectively, the serial number companion function seems equivalent to a small, 8-byte block of MCU EEPROM, which can be individually write-protected. As such, it isn't quite as useful as the factory-lasered serial numbers contained in the 1-Wire devices.

DUAL EVENT COUNTERS

Both the 31x and 32x families contain two independent 16-bit up counters. What may make them more useful to you than the common counter/timer blocks found in most MCUs is that the IPC's event counter registers are implemented as FRAM, so they retain their values even when the power is off.

In many applications, an event counter is by definition something that must be maintained over power on and off cycles. An EEPROM may be used to achieve this, but the limited write endurance of the EEPROM cell may make it unsuitable. FRAM event counter registers have all of the advantages of battery-backed RAM without

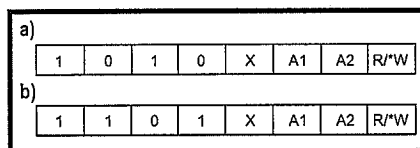


Figure 2a—Using the standard I²C convention, the FRAM array's slave address is defined as shown. b—In a similar fashion, the Integrated Companion devices' slave address is defined as shown.

the hassle of the battery. To add versatility, the count edge polarity can be programmed in the IPC, and the two 16-bit counters can be concatenated into one 32-bit register by setting the CC bit (bit 2) in the event count control register.

Call me fussy, but I wish I could have reconfigured the dual event counters into a single up/down counter. Then, adding a small amount of glue logic inside the chip would have allowed a rotary encoder to be connected up and its position read directly via the I²C bus. Of course, I'd like to see this feature on MCUs as well. So far, I've only run into it on Renesas Technology M16C MCUs.

COMPANION ACCESS

I won't get into all the details of the I²C bus transfer protocol here because it's well documented in the FRAM datasheets. However, an overview of some specifics for this particular device is in order.

The required slave address for accessing the FRAM in the device is shown in Figure 2a. A1 and A2 must match the state of the two address pins on the device. The R/*W bit defines whether a read or a write operation is to follow.

For write operations, after the appropriate slave address has been issued, you must send 2 bytes representing the desired memory address (most significant byte first). Then you must issue as many sequential data bytes as there are memory bytes to be written. This is followed by an I²C stop condition. Note that there are no page boundaries to worry about, so the entire memory array can be written in one burst if desired.

Memory reads can be done in one of two ways, and such transfers can be either single-byte or multi-byte. The FRAM device has a latch that contains the last memory address read. Therefore,

it's possible to send just the FRAM's slave address byte (with bit 0 = 1 for a read operation) and immediately begin reading memory data without specifying any address at all. Done this way, the FRAM will first increment its last memory location read pointer and then send out the memory data residing at this location. If you continue to perform I²C reads, the FRAM will return consecutive memory locations all the way through the entire memory array (and rolling over to 0000 if you read past the top of memory).

The only way to terminate this multi-byte read operation is to issue a NACK at the end of the last required memory read followed by an I²C stop condition. This allows for an efficient memory read operation because only 1 byte is transferred per memory location read.

Alternatively, you can read from a specific memory location by issuing the following sequence: a start condition, a slave address (R/*W bit set to 0), the MSB of the desired memory location, the LSB of desired memory location, a start condition, a slave address (R/*W bit set to 1), an I²C read operation (returns desired memory location), a NACK condition, and a stop condition. Similarly, in this mode, you can perform multiple reads of sequential memory locations by issuing multiple I²C read operations prior to the NACK and stop conditions, which are only issued after the last required memory location has been read.

A different slave address is used to access the companion functions (see Figure 2b). In this case, instead of specifying an FRAM location, the companions are accessed via 25 special function register (SFRs) in the address range of 0x00–0x18. Only one address byte is needed to access any of these registers. Therefore, an SFR write is performed like a memory write, except only one address byte is sent instead of two. Similarly, to read an SFR, the sequence for memory read is followed, using only 1 byte for the SFR address instead of the two needed for memory access.

The processor companion device contains separate pointers for both the last memory location read and the last

SFR register read. In practice, this means that you can interrupt a multi-byte FRAM read with an access to the RTC SFRs, for example, without having to reload the read pointer after the interruption. This makes for more efficient memory transfer operations in the presence of recurring SFR operations (e.g., those typically done in a background ISR in support of an RTC). This shortcut applies only to FRAM read operations. Write operations require that the memory address pointer be reloaded if an intervening SFR access is performed.

IDEAL SOLUTION

My work at Dalhousie University allows me more freedom to switch among processor families than would be practical if I were working in industry. *Circuit Cellar* contests also tempt me to try new microcontrollers. That being said, I still prefer to use the smallest, least expensive MCU that's feasible for any particular design. In my situation, that generally boils down to 8-bit MCUs. About the only times I've

been stymied using an 8-bit MCU were when I needed much more RAM than was available in such devices.

The Ramtron serial FRAMs, particularly the SPI versions, are an ideal solution. They're fast and require almost no MCU I/O port pins. Last but not least, they're dead easy to wire up on a circuit board.

It's easy to be an armchair quarterback, but in my opinion, Ramtron could have picked a somewhat better mix of companions for the IPC. The RTC is well implemented and the dual nonvolatile event counters are unique, but the watchdog and power-fail reset/monitor modules are generally already available in any MCU you'll find today. For example, a temperature sensor/alarm function with FRAM storage of the alarm parameters and possibly a logging function would be a unique addition.

Also, because the device already uses a 32,768 watch crystal, a programmable low-frequency interrupt to the MCU at a 1 Hz and slower rate would be useful in many situations. The counters

available on most MCUs are often not long enough to divide the MCU clock down to that rate directly. This would be useful for low-power applications in which the MCU is in Sleep mode most of the time. As I mentioned earlier, the event counters would be more versatile if a little extra logic were added to allow them to handle rotary encoders, which are common user interface devices today. Of course, it's always easier to wish for things than it is to implement them! ▣

Brian Millier (brian.millier@dal.ca) is an instrumentation engineer in the chemistry department at Dalhousie University in Halifax, Canada. He also runs Computer Interface Consultants.

SOURCES

AT25256 EEPROM

Atmel Corp.
www.atmel.com

FM25256 FRAM and IPCs

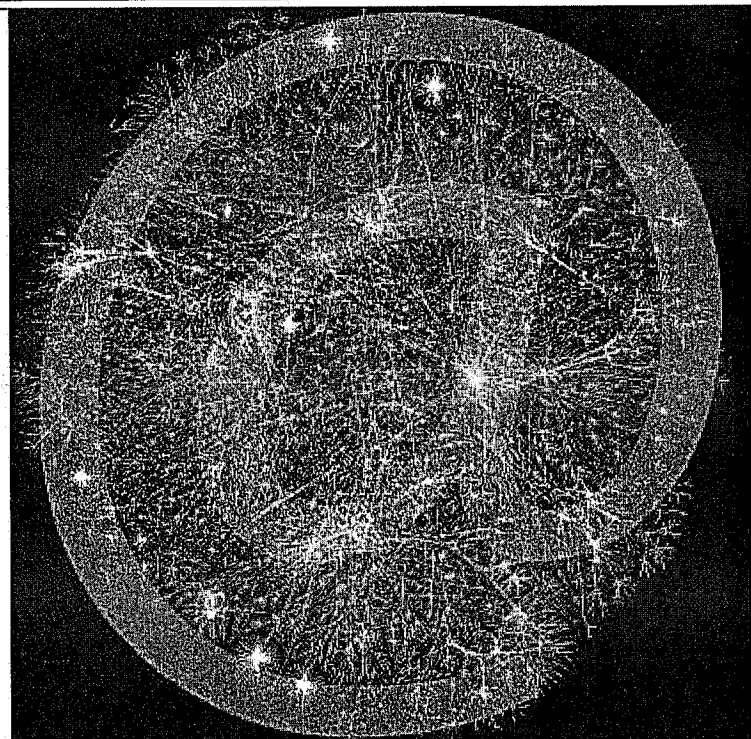
Ramtron International Corp.
www.ramtron.com

CIRCUIT CELLAR

Designer's Notification Network

Circuit Cellar design contest entrants have received thousands of valuable development tools and product samples. Because of their contest participation, these engineers receive advance e-mail notice from Circuit Cellar as soon as new samples become available. Now you too can benefit from this early notification.

Welcome to the Designer's Notification Network. Print subscribers are invited to join the Network for advance notice about our new sample distribution programs.



Find out more at www.circuitcellar.com/network.