Unlocking the Potential of Your Microcontroller

Ethan Wu

Storming Robots, Branchburg NJ, USA

Abstract. Many useful hardware features of advanced microcontrollers are often not utilized to their fullest potential in RoboCup Junior (RCJ), requiring workarounds such as multiple microcontrollers in a single robot. By allowing microcontroller hardware to handle most robot components, including sampling sensors, driving motors, and sending or receiving other data, the microcontroller's core is able to focus on other tasks such as compute-intensive processing and algorithms. This design principle is demonstrated by an RCJ Rescue Maze robot with over a dozen sensors controlled by a single microcontroller.

1 Introduction

Any advanced robot likely has a multitude of sensors and actuators. The microcontrollers on these robots must be able to quickly and accurately acquire, process, and act on incoming data. However, the most commonly used microcontrollers, members of the ATmega series (found in most Arduinos), typically struggle with meeting these performance requirements. Many turn to more powerful microcontrollers with higher clock speeds, however the additional hardware features of these controllers are often underutilized. Typically kludges—such as adding additional microcontrollers—are employed to satisfy feature requirements, resulting in much larger and more complex robots. To help the RCJ community solve this common problem, this paper examines how to offload data acquisition and actuator control from the core to peripheral hardware. Then, an example is presented demonstrating how these techniques can be successfully implemented with a robot for the Rescue Maze competition. This robot demonstrates that a large quantity of data can be captured using peripheral hardware while leaving most of the core available for other processing. The same strategies can easily be applied to other robotics projects, including but not limited to other divisions of RCJ.

2 Basic hardware

2.1 GPIOs

The most basic peripheral of any microcontroller is the GPIOs, or General Purpose Input/Outputs; these are the digital pins. They can be directly controlled by the core, but can also be assigned to the inputs and outputs of other peripherals on the microcontroller. For example, on the STM32 series of microcontrollers, all non-special pins (i.e. everything except power and clock source) are GPIOs that can be assigned to be used by other peripherals available on that pin, such as ADCs, I^{2}C, and even the debug port. Thus, one physical pin on the microcontroller package can serve many different roles, making pins with many peripherals attached valuable for future expansion and alternative uses or applications.

2.2 Interrupts

Interrupts are one of the most basic and important concepts in an embedded microcontroller; they have many applications. Different types of interrupts can be triggered by various components, but they all serve to “interrupt” other hardware in the microcontroller, essentially informing it that some event happened. Arduino users are likely familiar with external interrupts—those triggered by a digital state change on a GPIO pin. These external interrupts can trigger on a variety of conditions, including at minimum rising and falling edge triggers [4]. They can trigger an interrupt service routine in the microcontroller’s core, which is what is typically seen with Arduinos, but can also trigger other hardware and peripherals inside the microcontroller.

Besides physical pins and external events, other internal peripherals inside the microcontroller can also trigger interrupts, typically signaling an event that occurred in the peripheral. This enables complex hardware-handled behavior by allowing various peripherals to work together asynchronously, responding to tasks as they need to be done.

3 Timers

One of the most basic and versatile features on a microcontroller is its timers. They perform a very simple function—count. However, the various triggers, counting modes, and output options are what makes them so powerful.

3.1 CCR: Input

Timers typically contain one or more capture/compare registers (CCRs). With the CCRs in input capture mode, each CCR channel can be configured to save the current value of the counter when an edge is detected on a GPIO pin. This is particularly useful for capturing precise timing information without any intervention from the core. This functionality can be used to capture information about a pulse or PWM signal. Two CCRs capture each edge of a signal, giving the pulse’s width and its period (in the case of PWM), with the counter resetting on one of the edges.

3.2 CCR: Output

Each CCR can also generate an output by comparing its value with the current value of the counter. This is commonly used for generating a PWM signal (and is how an Arudino's hardware PWM works), with the value of the CCR specifying the on-time and the configurable maximum value of the counter specifying the period. It is often used for motor control.

3.3 Slave mode

The timer's counter can also be controlled by other external factors in what is termed slave mode. A timer can be set to reset, count, or switch counting direction depending on other events (from outside the microcontroller or from another timer) [3]. One important function this enables is encoder mode, where the timer's slave mode controller handles counting up or down depending on the quadrature encoder input. Because this is controlled by hardware, it is virtually impossible for the timer to skip encoder counts, and also does not require interrupting the core (with the possibility of miscounting) like an encoder implementation with external interrupts [1].

4 Peripheral features

4.1 ADC

Almost all microcontroller contain at least one analog-to-digital converter (ADC). More advanced microcontrollers also have additional features on ADCs such as analog watchdogs, which will generate an interrupt when a certain channel's value goes outside of a predefined range. ADCs can also include various sampling modes and sampling controls. The most basic of these is the sample rate; higher rates can be achieved by using fewer bits of precision. Many ADCs can also set up sampling sequences, allowing multiple channels to be sampled with their own sampling times, and the sampling order to be customized. For example, if a channel needs to be sampled more frequently than the others, it could occupy more sampling slots in the ADC's scan sequence; if a channel needed additional accuracy, it could have a longer sampling time.

4.2 Serial protocols

Microcontrollers contain various hardware peripherals for common serial protocols, such as I^2C, SPI, USART (which supports both UART or standard "serial" and more obscure synchronous protocols), and CAN. These peripherals are likely familiar to Arduino users and handle transmitting data over the protocol. The peripheral typically transmits or receives one word (usually byte) at a time, while handling protocol-level logic such as flow-control in UART and clock lines for I^2C and SPI. They will fire interrupts when the transmit register is done transmitting for the next word to be loaded into the register. Similarly, incoming data often also triggers an interrupt for the data to be read out of the receive register before the next word is received. However, both these reading and writing processes can be streamlined with DMA which is discussed next.

5 DMA

Many microcontrollers also possess Direct Memory Access peripherals, which copy memory from one region to another. This is typically used to move data between a peripheral and main memory, and is particularly useful for "queuing" up data to a peripheral that can only accept one word at a time. Each DMA consists of streams which will transfer between specific regions when an interrupt is fired. Most peripherals are able to trigger DMA requests when appropriate.

The DMA channels can be configured to automate many "long" tasks that would typically involve the core repeatedly writing or reading data. An ADC can read all of its input channels into main memory without core involvement by configuring DMA to cycle through memory addresses. Thus any necessary ADC values can be updated independently in the background. Communications over serial ports (UART, I^2C, SPI, etc.) can also be streamlined with DMA, because the DMA can take care of sequentially "feeding" words to the peripheral from a memory buffer. The DMA request is fired after each word is transmitted, then DMA copies the next word from memory to the transmit register. The core simply needs to start this request without further involvement in transmission, making this operation completely asynchronous and handled by hardware.

6 Example robot

6.1 Design

For the RCJ Rescue Maze competition, a compact robot was designed around a single microcontroller (STM32F405) to power all decision-making, data acquisition, and system control other than its vision system (see Fig. 1 for an architectural overview). In the planning phase, microcontroller hardware was carefully allocated between sensors of various types using the features discussed above.

Motor PWM control was driven by a timer, like the hardware PWM output of Arduinos.

Analog sensors, such as a reflectivity sensor and analog Sharp distance sensors, were connected to ADCs in continuous mode, with readings transferred into a buffer in main memory by DMA. Because the entire process is handled by hardware, adding additional sensors has minimal impact on overall microcontroller performance; the robot was designed with six of these distance sensors to facilitate accurate navigation.

Two Melexis temperature sensors were configured to output using PWM, which could then be read by a timer. Thus acquiring temperature readings was done independently for multiple sensors entirely utilizing microcontroller hardware, without needing to poll over I^2C or SPI.

An ultrasonic sensor was used with a timer in pulse capture mode to capture the length of time for which the echo signal was high. This allowed for highly accurate time (i.e. distance) measurement without needing to interrupt the core.

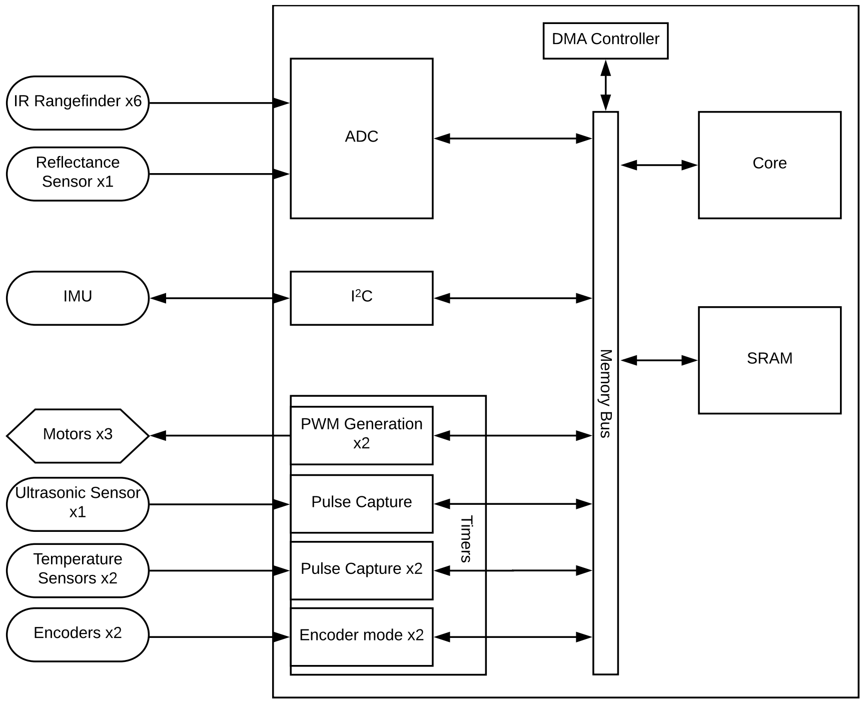

Fig. 1. An RCJ Rescue Maze robot architecture utilizing hardware peripherals of a microcontroller

Quadrature encoders were also managed by timers, providing accuracy at high speeds without constant core interrupts as would be typical on an Arduino. The timer hardware can easily handle a high rate of encoder counts (up to 84 MHz in the STM32F405 [2]) because it is implemented entirely in circuitry. Thus it is particularly useful in high speed, high precision applications.

Since the current values of these sensor inputs could be read by a simple memory access to the register of a timer or the memory buffer copied to by DMA, data acquisition code running in the core can be very simple and efficient.

The only sensor that could not be handled entirely by hardware was the I^{2}C-based IMU, which required periodic polling over the bus and sensor fusion processing by the core. However, this could still be optimized by utilizing DMA in I^{2}C transactions to perform the register reads asynchronously, allowing the core to perform other actions while the I^{2}C bus was busy.

Using this intelligent design, most sensors were handled by hardware peripherals of a single microcontroller. Therefore a majority of core processing time was dedicated to perform calculations relevant to the algorithmic task, navigating the robot through the maze.

6.2 Implementation

To realize all the features previously discussed, a printed circuit board was designed, fabricated, and assembled. The circuit board allowed for creating a very compact robot, as it contained most supporting components such as the robot's power supply (including voltage regulators, power filtering, and battery monitoring), its motor driver, and certain sensors (like Hall-effect encoders and the reflectivity sensor). As most components were located on one board, the only wires necessary were to plug in off-board sensors, allowing the overall robot to be small. Reliability was also greatly enhanced because there is no need for complex communications between multiple microcontrollers.

To program the microcontroller, first a project with basic libraries (utilizing CMSIS, the newlib standard library implementation, and ST's HAL), early initialization code (such as microcontroller core clock and loading memory), and linker configuration was created using GNU MCU Eclipse. Next, to initialize all the peripherals, ST's STM32CubeMX utility was used to graphically configure the device (including peripherals and their pin assignments as well as other clocks) and generate additional initialization code. Then, the linker configuration was modified to allocate a section of flash memory for persistent data storage. For loading and debugging code, a STLink debugger (hardware) was used with OpenOCD as a GDB server to bridge the STLink to debugger frontends.

With this hardware and software system, a robot was built capable of acquiring large quantities data from over a dozen senors in real-time without core interaction while leaving plenty of time for other tasks. The end result was a compact reliable robot utilizing a single microcontroller for most processing.

7 Conclusion

By unlocking more of the hardware capabilities of a microcontroller and choosing sensors to suit these capabilities, one can greatly streamline data acquisition for a robot, leaving much of the core's processing power available for other compute-intensive algorithms. By using this intelligent design, a compact RCJ Rescue Maze robot was able to be powered by a single microcontroller while successfully and reliably accomplishing its missions.

References

1. Anderson, R., Cervo, D.: Pro Arduino. Apress, Berkeley, CA (2013)
2. STMicroelectronics: ARM Cortex-M4 32b MCU+FPU, 210DMIPS, up to 1MB Flash/192+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces & camera, 8 edn. (September 2016)
3. STMicroelectronics: STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced Arm®-based 32-bit MCUs, 16 edn. (April 2018)
4. White, E.: Making embedded systems. O'Reilly, Beijing (2012)