Micro-Controllers: An Overview

Anil Aswani

December 1, 2005

1 Introduction

With the advent of low-cost processors and accessories, the ability to do control digitally has become increasingly feasible; this has led to a dramatic change in the way controllers are designed, tested, and implemented. Moreover, these digital controllers are usually more robust and easier to design than their analog counterparts.

A micro-controller (MC) is the processor that is used to do the digital control. Specially designed MC's often include built-in peripherals (such as memory, analog $\leftrightarrow$ digital convertors, and timers) to deal with the practicalities of doing MC design. A controller is a program written on the MC that does the control. The controller must be able to read information from the outside world, process the information, and output a control signal based on the information.

Digital controller design has a unique set of constraints which must be met. The controller must be:

1. Functional: The MC must be able to do the control. This means that it must be fast enough to sample the reference and feedback signals, process the signals, and output a control signal that stabilizes the system and meets the design constraints. The requirement of a low system time-delay due to the controller is inherent in these constraints.

2. Cheap: Many times, cheaper MC's have limited computation abilities - they are unable to do floating-point computations. This means that computations must be done with fixd-point computations. Also, the amount of memory and the speed of the MC are minimal.

3. Fast: Many systems use a single MC to do multiple tasks. Thus, a single controller on a MC must behave nicely. First, the controller should not use significant computation power. Second, the controller should be modular in terms of memory usage (and other MC peripherals usage).

These constraints and the overall flow of MC design will be explained further in this overview.

2 MC Paradigm

A MC must be able to take signals from the outside world, process the signals, and output control signals. The transition between the "outside world" (outside the MC) and the "inside world" (inside the MC) is an important one to understand. Conceptually, we start with analog signals, pass the analog filter through an anti-aliasing filter, sample and quantize the signal, process the digital signal in the MC, reconstruct the signal with a zero-order hold. The reconstructed analog signal can be passed through an optional anti-imaging filter. This process is seen in Figure 1.

The sampling must be fast enough to be able to do the control. The difficulties arise through having to satisfy the Nyquist rate keep the system time-delay low. The theoretical minimum sampling rate is given by the Nyquist rate, but in practice we sample several times faster than the Nyquist rate. Unfortunately, we are limited in how fast we sample by how much data the MC can process and by the sampling abilities of the MC. Moreover, we need to keep the system time-delay low. System time-delays add a phase-lag, which can lead to instability if the phase-lag (time-delay) is large.

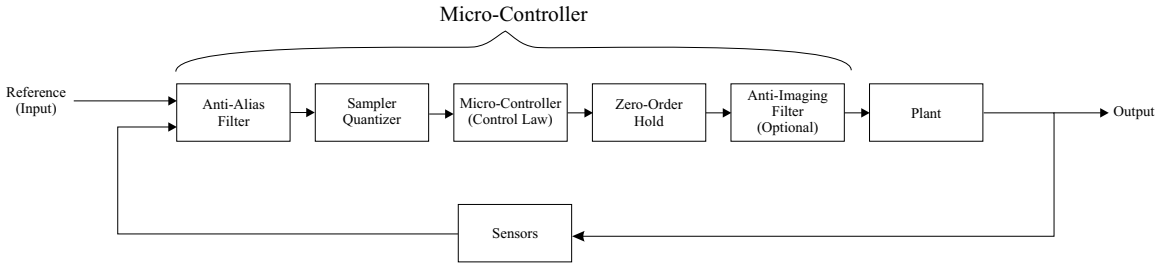

Figure 1: The Micro-Controller Paradigm

3 Controller Flow

The program flow of the controller is non-linear. The difficulties arise with the need to sample data and process the data at the same time. There are two approches to this problem; one approach uses interrupts and the other uses high-level MC functions written by the MC manufacturer.

Interrupts can be thought of as event-driven programming. In other words, when a special event occurs (e.g., the timer counts down to zero) the interrupt stops the program and calls a special function. After this special function finishes, the processor goes back to the last position in the original program and continues running.

With interrupts, two approaches can be used for the controller flow. One approach is to process and implement the control laws as a function called from inside the interrupt. The pseudo-code for this approach is:

```
interrupt on_timer_reaches_zero()
{
   turn_off_interrupts()
   initiate_adc_conversion()
   reset_timer()
   turn_on_interrupts()
}

interrupt on_adc_conversion_completes()
{
   turn_off_interrupts()
   read_sample()
   process_sample()
   implement_control_law() /* includes output */
   turn_on_interrupts()
}

main()
{
   setup_timers()
   setup_peripherals()

   while(1)
   {
        process_data()
   }
}
```

The approach using the high-level MC functions and the other approach using the interrupts are essen-

tially the same. In this approach, the interrupt or high-level function is used to grab data from the outside world and then the processing is done inside the main loop of the code. Using these high-level functions, it is important to set the sampling rate and the rate of the control-law. The pseudo-code for this method is:

```
main()
{
   setup_timers()
   setup_peripherals()

   while(1)
   {
         set_rates()
         read_data_from_outside() /* using interrupts or high-level functions */
         process_data()
   }
}
```

Modular programming is important, because in many systems there are multiple controllers on a single MC. For instance, in a car the MC will have a controller for cruise-control, another controller for the fuel-injection system, and so on. Therefore, the controller has to be written in a way that it is able to run with other controllers. This means, the controller should use as little processing power as possible, should minimize peripheral usage, and use memory carefully (i.e., not overwrite the memory of other controllers).

An increasingly difficult problem to deal with is that different controllers require different input sampling rates and different output rates. This difficulty is compounded by the need to share multiple data streams across different controllers. Dealing with this problem requires careful planning and organization of the controllers on a single MC. Interestingly, this is also an area of active research because of the ever-increasing complexity of these systems.

4 Peripherals

MC's often include several built-in peripherals including timers, memory, and analog $\leftrightarrow$ digital convertors. The timers are usually used to set the sampling rate and the output rate, but they can also be used to get a measurement of the time (in seconds) elapsed between different events. The analog $\leftrightarrow$ digital convertors often require careful use, since the bits these convertors input and output do not always match up between each other and with the integer representation used by the MC. For instance, the MC we use in the lab has 16-bit integers, a 10-bit analog $\rightarrow$ digital convertor, and a 12-bit digital $\rightarrow$ analog convertor.

5 Fixed-Point Arithmetic

Many MC's do not have floating-point capabilites; therefore, we must use different techniques to overcome this limitation. There are two approaches to this, and each has its advantages and disadvantages. One approach is to emulate floating-points using integer arithmetic. The other approach is to used fixed-point calculations. Fixed-point calculations directly use integer arithmetic and interpret these computations as decimal computations. These two approaches might sound the same, but there are important differences which are beyond the scope of our course - we are more interested in their relative merits.

The advantages of using emulated floating-points are because of their ease-of-programming and high numerical precision. The disadvantage of using emulated floating-points is that they are extremely slow and use up a lot of processing power. In practice, only a few calculations which require high precision are done using emulated floating points, and the rest are done with fixed-point calculations.

The main advantage of fixed-point calculations are that they are very fast. The disadvantage is that it is hard to program and has limited numerical precision and range (in fact there is a tradeoff that must be made between precision and range). The basic idea is that we pretend that there is a decimal point in

our integer, and do the computations pretending that the decimal point is there. For most purposes, this is the preferred technique, but most practical programs will use both approaches based upon the needs of the controller.

6 Sampled Data Systems

Despite having to sample our system, we can still control our system exactly. We can consider the sampling effects as part of the plant itself. Even the delay can be incorporated into the system model. These techniques are quite powerful, but still limited. we can only control our plant at the sampled points. This means, that we cannot control the behavior of the plant in between sampled points. Furthermore, if the sampling rate is too low, we will still get aliasing issues and time-delays which will lead to either instability or unacceptable system performance.