AN1501
APPLICATION NOTE

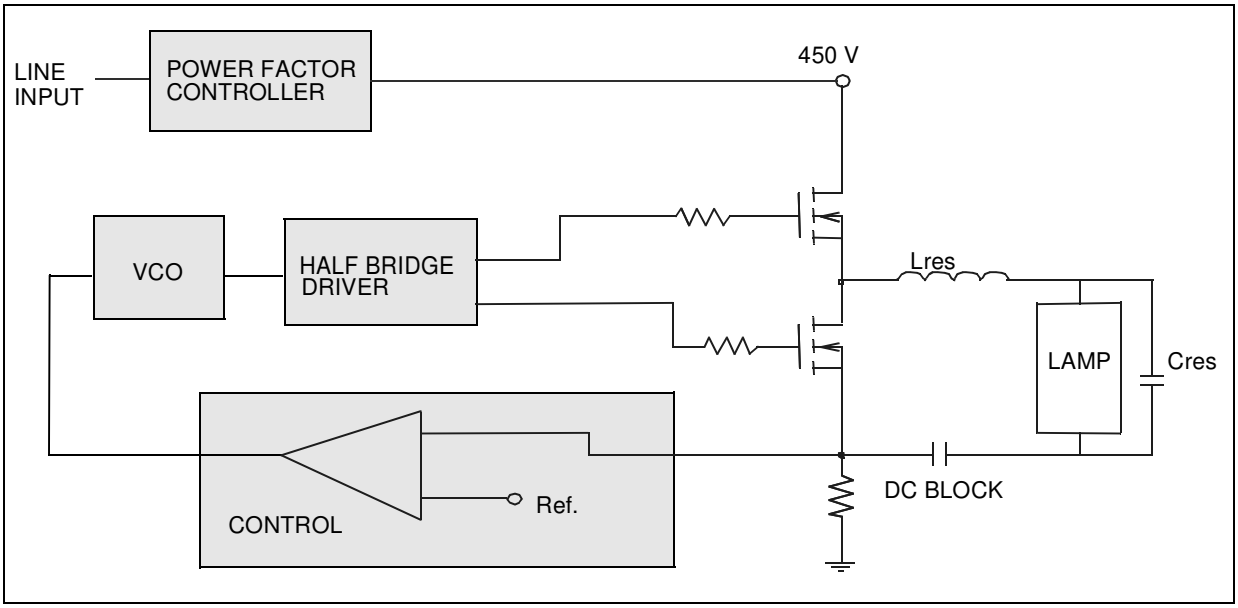
SIMPLE MICROCONTROLLED BALLAST

by Clifford Ortmeyer and Albert Kunickis Jr

INTRODUCTION

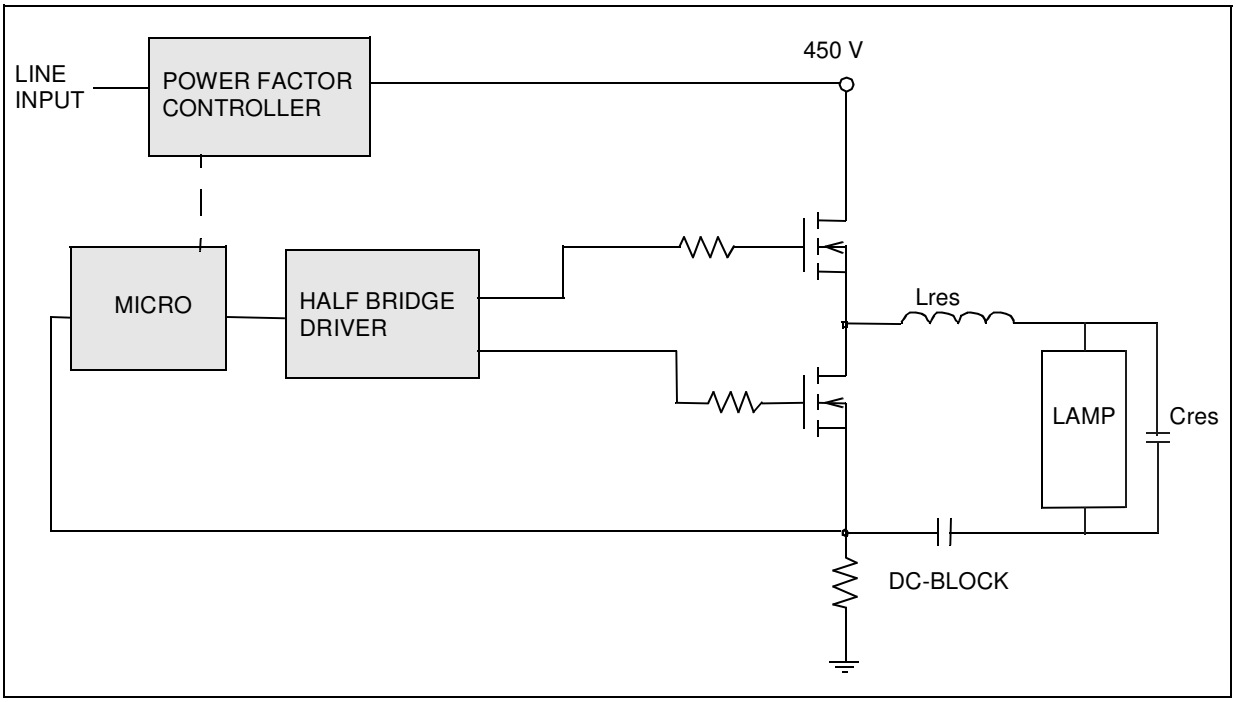
The purpose of this paper is to give a basic understanding of a microcontroller and its potential usage in an electronic ballast. A brief summary of how the microcontroller operates and the most common types of functions it can perform will be shown as they relate to being used in an electronic ballast. Next, ideas of how to implement the most common functions and their associated advantages/weaknesses will be examined. Finally, a brief summary of things that should be examined closely will be presented to help assure a good start to a basic microcontrolled ballast design.

First, lets look at a basic diagram of an existing electronic ballast.



This is a very simple diagram of an existing electronic ballast. Today, the voltage controlled oscillator (VCO) and the half bridge driver are usually combined in a single package. The Control portion, which may be comprised of the fault detection circuitry and an op-amp to close the loop, may also be included in the same package (for example, the L6574 Ballast Controller IC). This is a good solution for having a basic platform from which new designs can easily be made. In some cases, it may be necessary to include a more flexible solution that allows for parameters that are usually fixed in an analog solution – such as ignition profiles and restart methods. It is in these and many other cases that a microcontroller can be used to define a more user specific operating profile.

A simplified diagram of a microcontrolled ballast is shown below.



In this diagram, the microcontroller takes the place of the VCO and the control logic. The microcontroller has an output(s) that emulates the VCO output which in turn controls the turn-on and turn-off the upper and lower portion of the half bridge. In this manner, the dead time, frequency, and duty cycle of the half bridge output can all be independently controlled.

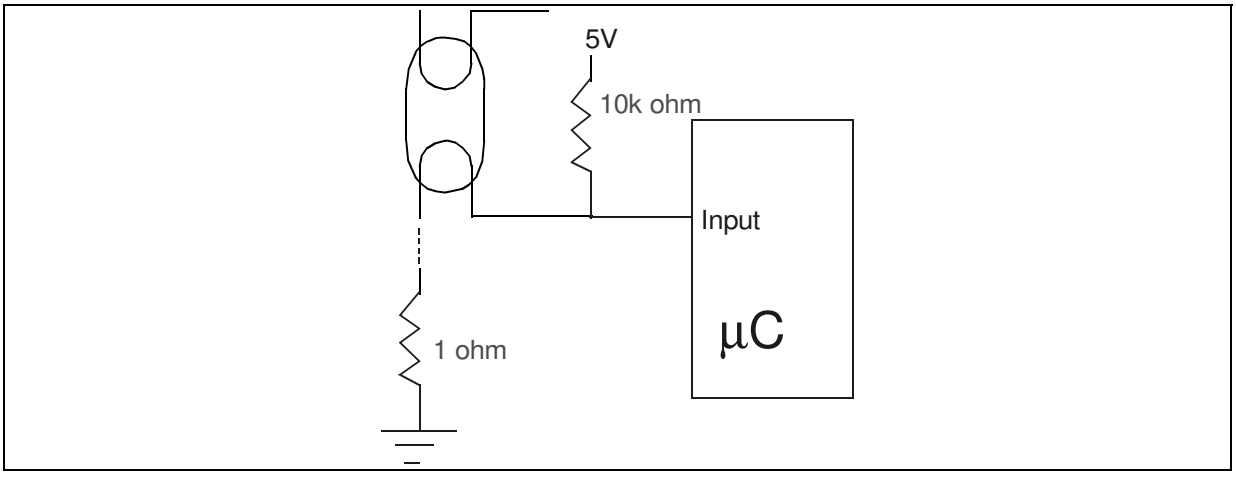
The control logic that has been replaced by the micro is essentially the brains of the control logic. External analog components will still be needed to scale down and, if needed, filter the fault signals. The micro can then control the response to each fault condition as determined by the users programming code. An example of when this might be useful is when a lamp fails to ignite, the micro could detect this and restart the preheat and ignition sequence but with a longer preheat time.

1 MICROCONTROLLER FUNDAMENTALS

To understand how the micro can be used in a ballast, the basic components that will generally be used need to first be understood. One of the few basic components that will be used are “inputs” and “outputs”. By this we mean that when the user programs the micro, the programming code tells it which pins are to be an input or an output. Not all pins can be configured in this manner, but for now we will concentrate on the pins whose functions we can modify.

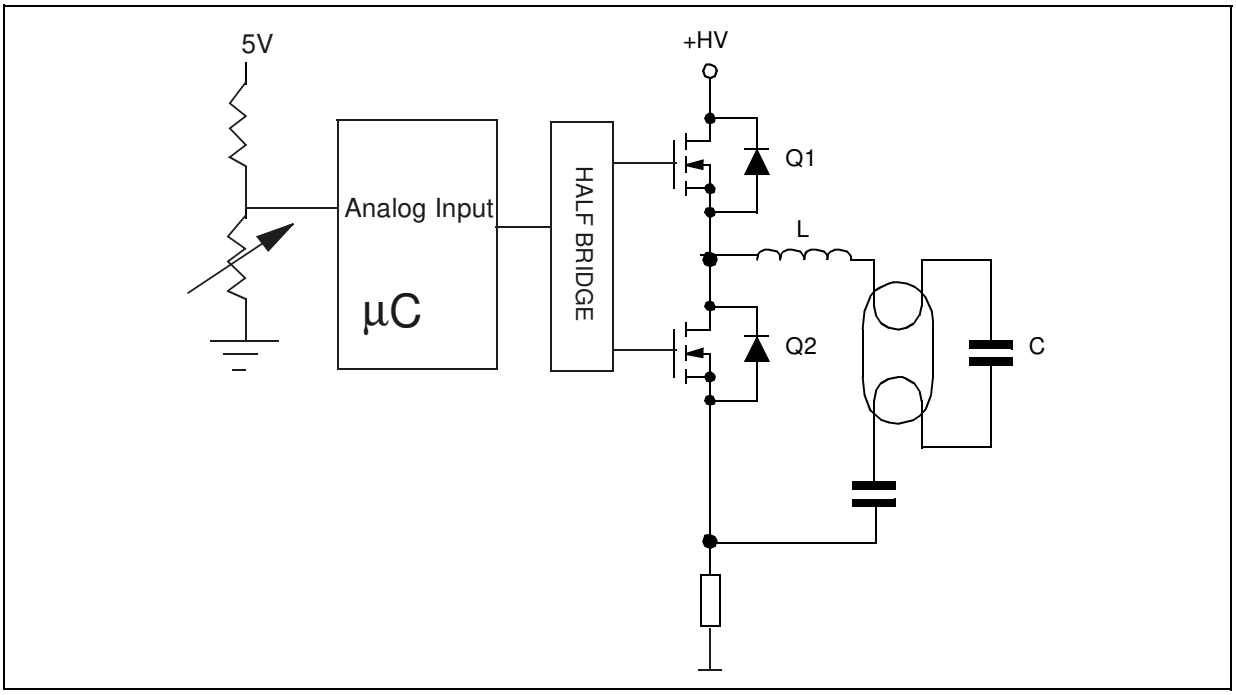
First lets look at a pin that we have configured as an input pin. An input pin essentially looks at the voltage on the pin and passes this information to the main processor (the “brain” of the microcontroller). It can tell the processor when it sees a rising or falling voltage level, or it can read the exact voltage level on that pin. Which type of voltage it is to look for is something that the user configures when the user programs the microcontroller. Generally, a pin that is configured as an “input” looks for either a high or low voltage level. An “analog input” is an input that reads the exact voltage level on the pin as opposed to looking only for a high or low level. A typical example using an “input” pin is shown below.

1.1 INPUT EXAMPLE



In this example we configure the pin to be an input pin and to look for a high voltage. For instance, when the lower lamp filament is connected to the 1 ohm resistor, the 10k ohm and 1 ohm resistor form a voltage divider where the midpoint is brought to the input pin. When the filament is connected to the 1ohm resistor, the voltage divider applies approximately 0 volts to the input pin. When the filament is disconnected from the 1 ohm resistor, for instance in the case of lamp removal (as shown in the picture), the voltage on the input pin rises to 5v. The microcontroller sees this voltage level shift, and can then take the appropriate action. The action taken is determined by what the user tells the micro to do when the microcontroller is programmed. In this case the user may tell the ballast to turn off since the lamp has been removed.

1.2 ANALOG INPUT EXAMPLE



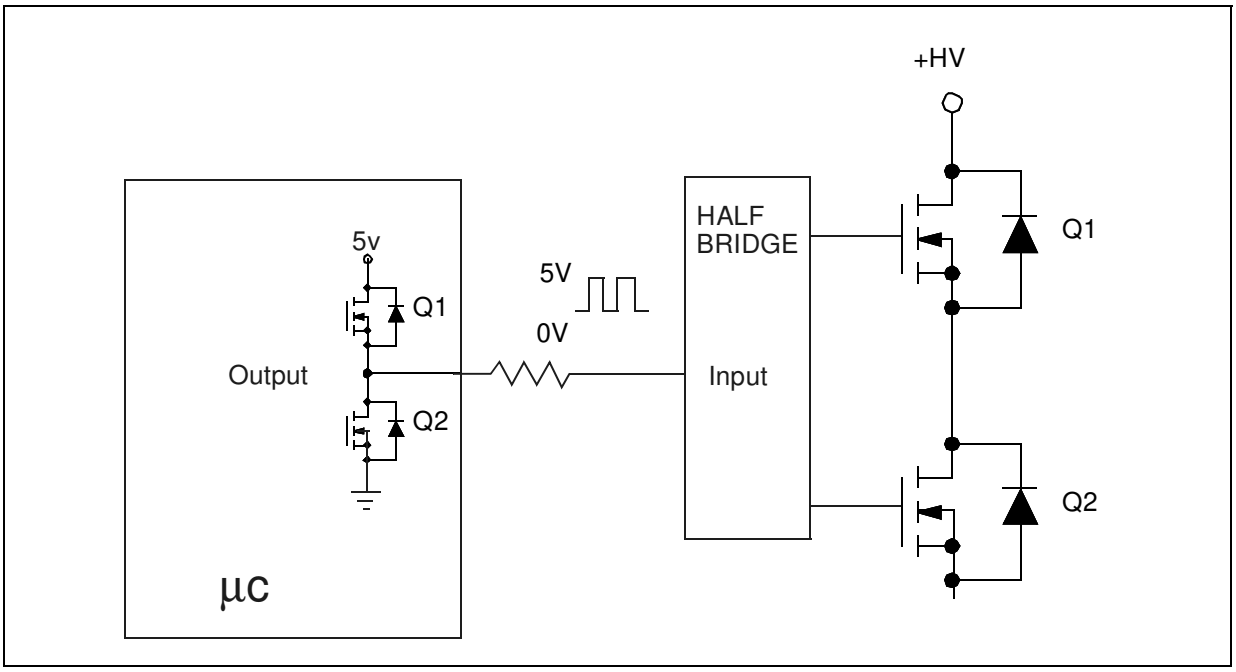
When a pin is configured as an “analog input” it acts as an A/D converter and looks at the voltage on the pin and transforms that voltage of 0V to 5V into a corresponding number between 0 and 255. For example, if the voltage on the Analog Input pin is 2.5 V, then the A/D will convert the 2.5V to a value of 128. The program that has been stored in the micro may then in turn tell an output pin to change the frequency of the half bridge to that of a 50% dimming level.

How does the microcontroller change the frequency of the half bridge? This is done by controlling a pin that has been configured as an “Output” pin.

Just as we configured a pin to be either an “input” or an “analog input”, we can also configure a pin to be used as an “output”. An output pin can configured in two different states – either a “push-pull” output or an “open drain” output.

In the push-pull configuration, a “high” can be applied to the pin. This puts a voltage on the pin that is equivalent to the Vcc of the microcontroller with a limited current sourcing capability (few mA). The second mode in the push-pull configuration is a “low”. In the low state, the pin is shorted to ground and again has a limited capability to sink current up to 30mA (high current pins only). An example of a “push-pull” configuration is given next.

1.3 PUSH-PULL OUTPUT EXAMPLE



In this example, the pin is configured as a push-pull output. The output pin goes from a “low” to a “high” state, which in turn applies 0V on the output pin and then 5V correspondingly. A “low” on the pin is accomplished by turning on the lower FET in the µC, thus pulling the output pin to ground. A “high” is accomplished by turning off the lower FET and turning on the upper FET in the microcontroller.

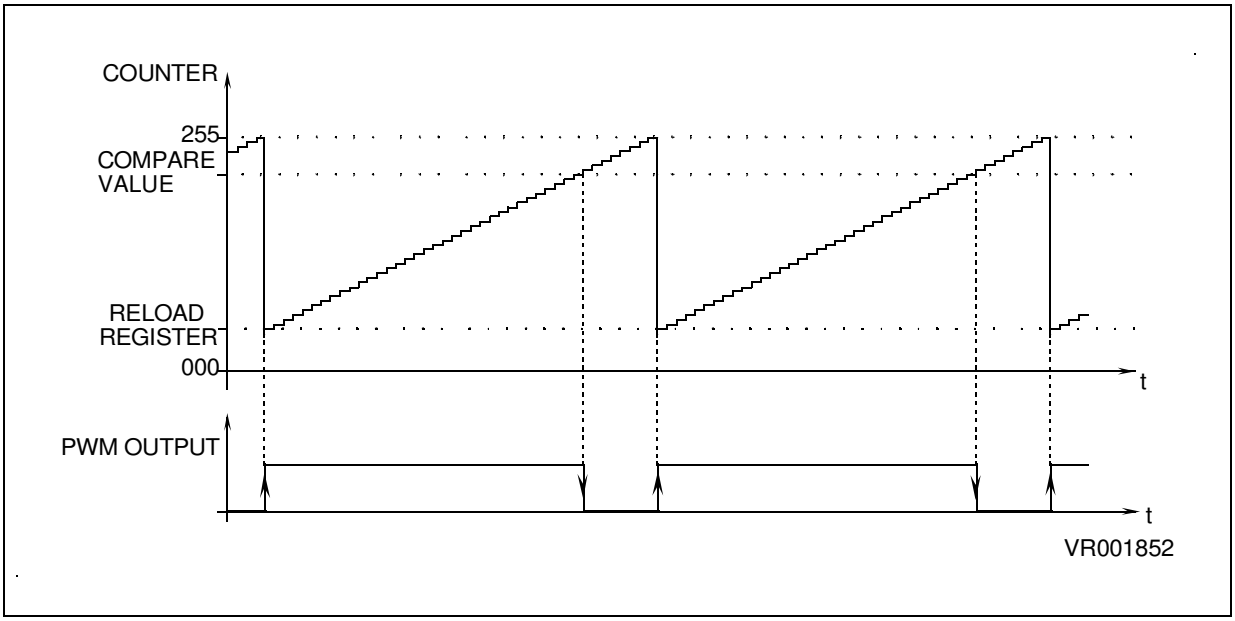
In the example above, a single output from the micro goes into a single input on a half bridge driver. In cases where the user wants to control the upper and lower FET independently, a half bridge driver with two inputs can be used – one for turning on/off the upper FET and another for turning on/off the lower FET. In that case, the user would configure two pins on the micro-controller as outputs – one for each half bridge input. An example of when this would be useful is when the user wants to dim the ballast by varying the PWM duty cycle of the half bridge output instead of only varying the frequency as shown above.

The second configuration of the output pin is as an “open drain” output. This configuration is generally characterized as an open drain FET. In this case a “high” corresponds to the pin being in a “floating” state and a “low” again acting as a short to ground. An example of this state being used would be when the user wants to short a point to ground, but at all other times, the pin is to have no affect on the circuit.

In addition to the aforementioned parts of a micro, there are many other components that can be used in a ballast. We will discuss the most common components briefly and leave it to the reader to further investigate each components workings in more detail.

2 AUTORELOAD TIMER

The first component (or peripheral) is called the “autoreload timer“. The autoreload timer is just what its name implies. It is a timer that the user sets up in his/her program that automatically restarts after it has timed out. The user will specify how long the timer is to count for and in addition, can also have actions taken at the beginning and at any point within the count. If for instance the user had an operation that was to operate continuously with respect to time, there would be little sense in having the program repeat lines of code continuously. This is because the microprocessor core (brain) has to operate on each line of code as it runs through the program. Thus in an operation such as the push-pull output example above, each time the output is to switch from high to low and visa versa, the processor would have to process the instructions for each change. However, with the autoreload timer, we can tell the timer to make the output go high at the beginning of each count and somewhere before the count runs out, we tell the timer to make the output go low. An example diagram of this function is shown below.



Notice in the diagram above there is a starting point called the reload register. This is where the count begins and the output goes high. The second point is the compare value. Once the counter reaches this value output goes low. So, by adjusting the reload register value the frequency of the PWM output is set. By adjusting the compare value, the duty cycle is then set. In most cases if the autoreload timer is being used to drive the PWM for the half bridge, the duty cycle will most likely be 50%. So in that case the compare value would be set half way between the reload register value and the value 255.

In summary, the autoreload timer can be set up once to operate by itself so that the processor core can perform other functions. If the frequency needs to be changed, the program can again load new values into the reload register and compare value, and start the counter again

running on its own. The autoreload timer also has other functions, but this is probably one of the most common usages in ballast design. A brief look at a datasheet of one of ST's microcontrollers with the autoreload feature will go into more detail on the capabilities of this peripheral.

3 TIMER

In addition to the autoreload timer, there may also be more than one generic timer available. The generic timer counts down to zero from a number that is input from the user program. The duration of each count can also be modified by loading a different number into it's corresponding register. An example of when the timer could be used would be the following. In order to preheat the lamp filaments of a microcontrolled ballast, the autoreload timer could be set to output a PWM (as discussed above) at 100 kHz. Once the PWM was started, the timer could be set up and turned on to count for 1 second. After the timer finished counting, the counter would tell the processor it was finished and the user program could then tell the autoreload timer to operate at a different frequency or ramp down to an ignition frequency. This process will be detailed once we look at the example program (later in the paper).

4 PROGRAMMING AND OPERATION

Now that we have some of the basic components of the microcontroller covered, lets discuss a few more details and then show how this is all implemented in an example program.

First, lets discuss how the microcontroller operates. It is assumed that the user has had some programming experience. When the user writes a program, that program is stored in the memory of the microcontroller. The processor then starts from the beginning of the program and works its way down performing operations that the program tells it to do (such as configure pin one as an output, then make it a "high" etc.) from the beginning to the end of the program. Programs must never end and should always run in some type of continuous loop.

One main feature that a program performs is to look for signals that require some type of action to be taken. This was shown in the first example. When the input went from 0 to 5V, the processor is to shut down the ballast. They way in which the processor knows that the voltage has changed can be determined in two ways – either by polling or by an interrupt. When we poll an input, we essentially write in code into our program to go and look at the input pin to see if its voltage level has changed. If this voltage change is a critical parameter, code must be entered quite often to go and look at this pin. One problem with this method of detection is that it the processor must stop what it is doing, go look at the pin, and go back to what it was doing. However, if a critical fault occurs, damage may be done before the code gets back to the point where it looks for that fault. In that case it may be best to use the second method of detection – an interrupt.

When time is of the essence to capture a fault signal, an interrupt may be used to tell the micro that a problem has occurred. An interrupt is a signal that is sent to the processor to essentially “interrupt” what it is currently doing and take care of the issue that caused the interrupt. Many of the pins that are configured as inputs can also be configured to generate an interrupt. With an interrupt enabled pin, the program code does not have to be written to constantly go back and look at the voltage on the pin. The microcontroller is able to watch the pin for changes, while at the same time running the program code that is performing other operations. Relating again to our first example, if the lamp is removed and the ballast is not shut down immediately, hard switching could occur and damage to the circuit could result. In this case the user could program the input pin to trigger an interrupt when the fault occurs. The interrupt would then tell the micro where to go in the code to take care of the problem.

So what method should be chosen to detect changes in a pins voltage level or state? This is dependant on how critical the timing is. If it is not critical, the user may want to poll the pin since this is always available (simply adding additional code). The main reason that one would not always want to use an interrupt is two-fold. First, depending on the type of interrupt that is being used, a more critical interrupt may happen on another pin. If this happens, the more critical interrupt may not be taken care of until the first interrupt is resolved. Second, not all pins can be enabled with an interrupt that will operate in the same fashion. For instance, some pins will only allow a rising edge interrupt, and if all of those interrupts are taken with only falling edge interrupts available, then the user might have to put an inverting circuit into the project (thus more cost). Other reasons and options will become more apparent as the user begins programming.

5 STORING DATA

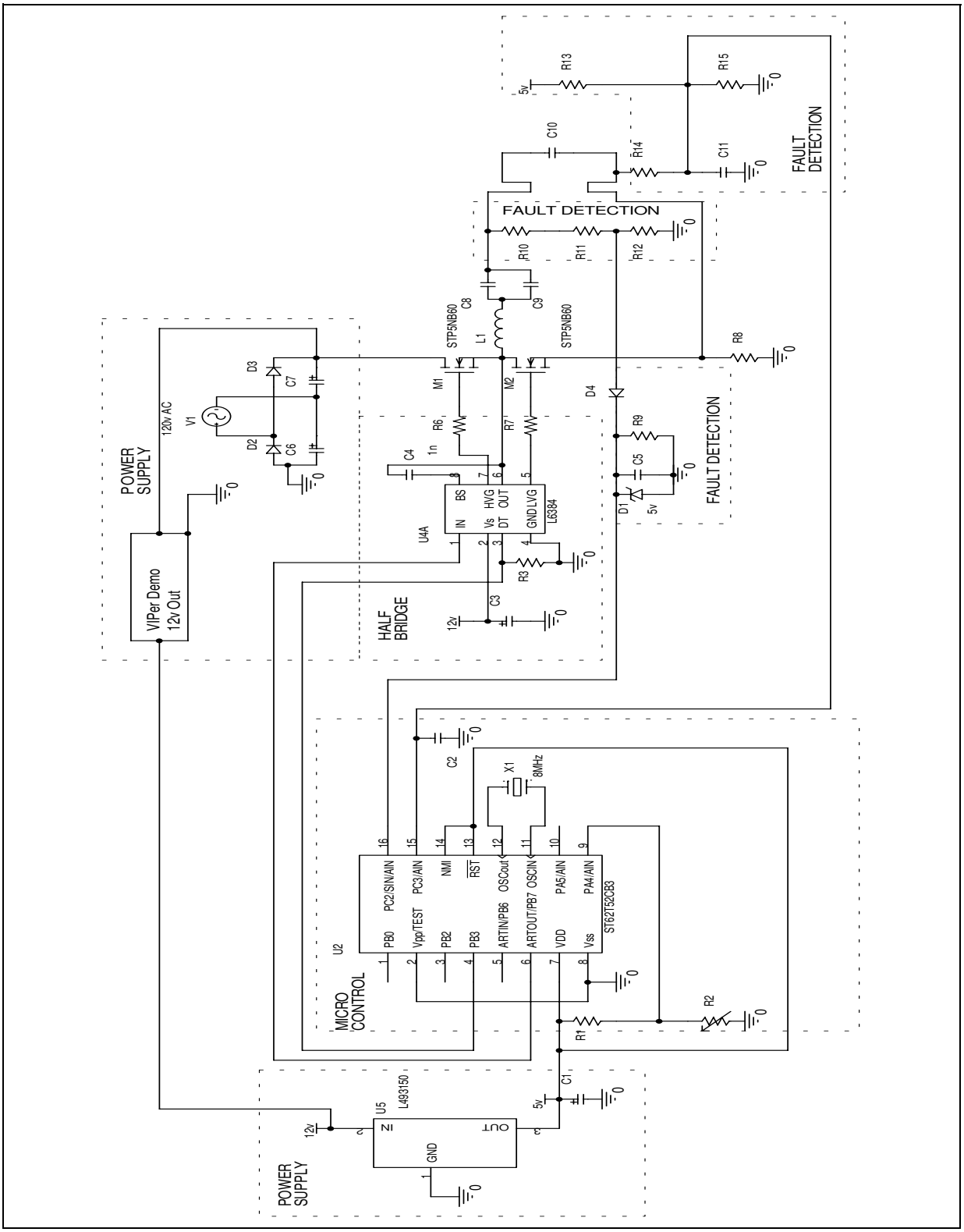
Another advantage of using a micro is the ability to store information. This information can take many forms. One type of information that can be stored is data that may be used depending on what inputs the micro receives. For instance, if the circuit detects that a 18W CFL is inserted into the lamp socket, a different preheat time may be desired than would be used for a 13W CFL. The microcontroller's "analog input" will take the input from the lamp detection circuit, match it with the correct preheat time, and load that time into the "timer". If the 13W CFL was inserted, the same process would occur, but the timer would pick the other stored time variable and load it into the "timer" instead.

If the amount of data that needs to be retrieved is larger than just a few points of data, a "table" may be used. A table is an area of memory where larger amounts of data can be stored. An example of when a table might be used is for the definition of a dimming level. Lets say that one of the microcontrollers inputs monitors a voltage level (0-5V for instance). A voltage of 0V corresponds to a dim level of 1%, 2.5V equals 50%, and a voltage level of 5V corresponds to a dim level of 100%. The PWM frequency that corresponds to each of the dim levels is 90kHz, 65kHz, and 55kHz correspondingly. Thus a representative table may look like this for a typical dimming design:

A/D reading	Corresponding dim level	PWM frequency
0V	1%	90kHz
1V	20%	80kHz
2V	40%	70kHz
3V	60%	60kHz
4V	80%	58kHz
5V	100%	55kHz

The actual data table(s) would store the appropriate variables to make the autoreload timer operate at the frequencies shown in the chart. Also, if a more detailed dimming range is needed, the table would be enlarged to include A/D readings that had fractional voltages. For example, 1.1V may correspond to a dim level of 21% - etc.

Figure 1. Simple microcontrolled ballast schematic



6 EXAMPLE PROGRAM

We will now look at an example program written in assembly language just to give an idea of what the code might look like. We will point out the major sections of the code and what they are doing. It is also written in C language as well and is included after the assembly language.

The associated zip file is the code (written in “C” & assembly) used for the demonstration ballast. Programming comments are included to help users and other programmers determine what is happening in the main parts of the code.

In addition, in the appendix section, you will find the flowchart of the program.

APPENDIX

Figure 2. Flowchart