Intelligent Power Supply Controller

R.S. Rumrill, Alpha Scientific Electronics
1868 National Ave. Hayward, California 94545

D.J. Reinagel, JCIL Engineering
293 MacArthur Ave. San Leandro, California 94577

ABSTRACT

We have developed a new power supply controller which would combine 20-bit precision, simple interfacing, and versatile software control. It performs many tasks internal to the power supply and also communicates with an external host computer. Parameters can be entered and/or read over a serial link using one of the 82 command words [1]. In addition, an optional remote control panel can be located up to thousands of feet away. This new controller will reduce the software development time normally spent by the user, while increasing the reliability of the system. The cost is less than buying the equivalent separate CAMAC system. Nonvolatile memory remembers all configuration data; one generic controller can thus be programmed to use anywhere from the smallest power supply to the largest.

HARDWARE

A. General

The controller generates the necessary drive signals for both the transistor regulator and the preregulator. It has inputs for up to 24 interlocks. It controls the main contactor as well as an optional load reverse switch.

The controller is built into a standard 5.25 inch high chassis. An aluminum divider runs vertically down the center axis. The resulting two halves form separate analog and digital sections. Each section has its own motherboard. The motherboards are passive, containing only the 150 pin connectors for the individual boards and several other smaller connectors.

For each major function a new board was developed. These include boards for:

1. CPU Circuits
2. Interlocks Circuits
3. RS-422 Serial Ports
4. DAC's and ADC's
5. Amplifiers
6. Configuration PROM
7. RS-422 Control Panel

Each board is designed to be either a digital or an analog board and to mount into its half of the chassis. The DAC/ADC board is able to span the two halves, picking up the digital half, opto-isolating it, converting it to analog signals, then dropping it down to the analog half of the chassis. CMOS circuits were used wherever possible to reduce power consumption, and therefore, the dissipated heat.

We are presently building 22 power supplies using these controllers. They will be used at the Clinton P. Anderson Meson Facility at Los Alamos (LAMPF) [2].

B. CPU

An NEC V30 microprocessor is used. This is similar to an Intel 8086, however the V30 has a better arithmetic logic unit. 128k bytes of EPROM memory is used, with expansion capability to 512k bytes possible. 64k of battery backed RAM is used, expandable to 256k. A Maxim MAX695 watchdog circuit is used in the reset circuit. Three PAL's are used to provide the various logic decoding, etc.

C. Interlock Circuits

The interlock circuits work on 24 volts DC. Each circuit connects to the LED half of an optical isolator. The output half then connects into the microprocessor circuits. Each interlock one can be individually configured for the following parameters:

1. Interlock name
2. Normal state (open or closed)
3. Fast off or ramp down to off
4. Monitor or ignore

The interlock name is displayed on the front panel during a fault condition and is available to read over the serial link.

The choice was given to allow certain interlocks to ramp down before turning off, thereby, reducing stresses in both the magnets and the input power system. Events such as a transformer overheating may have required many minutes to occur, so a slow ramp down to off is appropriate.

D. RS-422 Serial Circuits

Four serial channels we desired for the following:

1. Host Computer Link
2. Local Control Panel
3. Optional Remote Control Panel
4. Private Diagnostic port

The control panels communicate at 57.6 kbaud. National NS16550 UART's are used, these contain internal FIFO's, and are able to minimize the CPU overhead required for communication. The host computer link can operate at baud rates between 300 and 38.4k. Via this port, one can control virtually any internal function using simple 3 letter commands. The diagnostic port was added for programing and debugging and is not for public use. The two ports which connect to outside equipment are opto-isolated from the microprocessor circuits. A 5 volt DC-DC converter is used to generate to required voltages on the other side of the isolation.

E. DAC/ADC Circuits

The output from the current transductor is measured by an Analog Devices AD624 instrumentation amplifier. The resulting signal applied to a Crystal Semiconductor CS5503 Analog-to Digital Converter. This ADC is a 20 bit delta-sigma device with several features which make it ideally suited for our application. Likewise, the level of the output voltage, transistor voltage, and ground fault current are each digitized by a 16 bit CS5501. The input stage of these ADC's have a 10 Hz lowpass filter with about 50dB of rejection at 60 Hz. The serial output stage is perfect for optical isolating and multiplexing. Because the input range of the device is ±2.5 volts, the 10 volt signals used in the controller are divided 4:1 right at the ADC using precision Vishay resistor networks.

The Current Reference voltage is generated by an 18 bit Sipex SP9380 Digital-to-Analog Converter. The resulting ±10 volt signal is used by the Linear Technology LT1007 error amplifier. Several Burr-Brown DAC's are used to produce the voltage reference and the preregulator reference. A 12 bit DAC is used to provide a fine, or vernier adjustment into the Sipex's summing junction. This accomplishes two things. It produces apparent resolution of greater than 20 bits and allows for a fine correction feedback loop.

This "drift correction loop" was implemented because the Crystal ADC actually has better specs than the best DAC we could find. Because almost all DAC's use a R-2R ladder structure, the ultimate drift is always limited the specs of the resistors used inside the DAC. The ADC, on the other hand, uses no resistors to digitize the signal. We simply try to keep the current measuring ADC reading constant by adjusting the fine vernier DAC as needed. Only the fine DAC is controlled by this loop, the 18 bit Sipex DAC is used "open loop". The current control DAC's are therefore "inside the loop" and their errors reduced to virtually nil. The algorithm used was selected to be dead slow, the desire being to take out drift without causing any loop instabilities, even on the most inductive magnets imaginable.

SOFTWARE

In the design of this controller, it was desirable to use a high-level communications protocol for host communication as well as inter-task communication [3]. This allows for supporting multiple command sources and differing physical communication medium. It was also desirable to have each sub-system in the power supply controller, respond to commands in exactly the same way, independent of the source of the command. To achieve these objectives, the software program was written in an object oriented programming language, C++ [4].

Objects are software entities within the processor. For example, the set of all interlocks form a class; the set of all analog-to-digital converters form another class. However, the most significant class in this controller is not one that corresponds to a physical entity, but one which is used for communication between the objects and application programs. This class, named cmdFrame, forms a generalized data packet which was sufficient for all inter-process communication. This class will be detailed below. Since a number of these objects co-exist and are independent of each other, the system behaves as though several programs were running concurrently under the control of a multi-tasking operating system. No such operating system was used.

The communication class, cmdFrame, has a group of public parameters, listed below with their data type:

```
int        command_number;
int        done;
int        application_scratch[
           SIZE_APPLICATION_SCRATCH];
unsigned   next_crc_should_be;
int        error_code;
int        wait_flags;
int        need_current_value;
int        argcount;
float      float_value;
float      pot_scale_factor;
char*      response_units;
int        response_flags;
char*      strings_out[SIZE_STRINGS_OUT];
```

The command_number indicates which process, or application, is to receive and act on this object. Other entries in this object will give specific information as to what is being requested. The Boolean, done, is set by the receiving process when it has completed the requested task. For a number of commands, as with the reading of the interlock conditions, communication between the command source and the application must repeat until all requested information has been sent. To allow for parameter passing between applications and for intra-application data storage, a small array, called application_scratch, is provided; its usage is very application dependent. The next_crc_should_be is used for error checking when a command source is sending a block of data to the controller. When an application detects an error condition, whether due to the passed parameters or due to the state of the power supply, it signals the error through the error_code parameter. When the cmdFrame object needs to wait for some event before being passed along, the wait_flags parameter is set appropriately. This saves valuable processor resources by eliminating the need for busy-waiting. When one application needs to read another application's current setting, then the Boolean need_current_value is set. If the application is sending a floating point response, then the argcount parameter is set and the value is place in float_value. If there is a unit name associated with this floating point response, e.g., Amps/sec, then the response_units is set to point to the appropriate name. For those applications which have their values adjusted at a control panel, through the use of a digital pot, the pot_scale_factor parameter is set to indicate the magnitude of the effect of the digital pot. When the application is returning a response, the parameter response_flags is used to indicate which parts of the data in the cmdFrame is to be sent back to the control source and in which order. If the application is returning a text message, then the array, strings_out, is set to point to entries in the word dictionary which form the message.

To manage the cmdFrame class, there are a number of subroutines, known as member functions, which will coordinate the loading and parsing of the object. These member functions with their return types are:

```
        cmdframe(const int);
int     get_source_ID();
int     active_controller();
int     set_active_controller(int value);
char*   getbuf_text_out();
char*   gets_text_in();
char    putc_text_out(char);
char*   puts_text_out(char*);
char    getc_text_in();
void    ungetc_text_in(char);
int     check_for_no_args();
void    clear();
void    init();
```

When a cmdFrame object is created, then its constructor, cmdframe(), is invoked. This initializes key private parameters in the object, as well as clearing most of the other parameters by calling another member function, init(). Init() is invoked each time a communication transaction has completed as indicated by the setting of the Boolean done. Another member function, clear() is called by any application when a command is not yet done but all the data in the object has been operated upon and the application wants a clear slate to work on. Although it is ideal for all applications to respond in exactly the same way to all command sources, there are some idiosyncrasies associated with the different command sources. For example, to enter the super-user mode, a password is needed; from the serial communication link, an ASCII string is entered, but from a control panel, a combination must be entered using the digital pot. In order for the SUN application to know which parameter to examine, it must call the member function get_source_ID(). When adjusting power supply parameters, it is necessary to have exclusive access to these parameters for a period of time. To determine if the command requester has been granted this exclusive access, the member function active_controller() is called and compared to the source_ID. If they match, then the application makes the requested change; otherwise, an error_code is returned. The member function set_active_controller() is called by the REQ application to inform all the users of the class cmdFrame who is the active controller at that instant.

Often times in object communication, there is need to pass ASCII characters and strings using the cmdFrame as the medium of transfer. To coordinate this, a set of member functions are provided; for reading characters out of a cmdFrame, the functions getc_text_in() and ungetc_text_in() are provided. For fetching the entire remaining stream of characters, the function gets_text_in() is provided. In some cases, the application expects there to be no input at all, and to verify this condition, the member function check_for_no_args() is provided. For writing characters into the cmdFrame, the function putc_text_out() is provided; this function ensures that character array lengths are not violated. When it is necessary to write characters directly into this array, the function getbuf_text_out() is provided to pass a pointer to the location where the next character should be placed. However, care must then be taken by the application that it does not exceed the character array length. When writing a character string into the array, strings_out, the function puts_text_out() places that string in the next available location. There are a number of private parameters within the cmdFrame class to affect these member functions, but these will not be detailed.

The use of this communication object has proven to provide a clean interface for the entire controller program and makes it simple to locate and solve programming problems. The modularity of this approach also makes it possible to add other communication mediums without requiring significant changes to the other parts of the program, as well as adding other application commands.

REFERENCES

[1] For a detailed listing of the command words contact the authors.

[2] S. Cohen and R. Stuewe, "Magnet Power Supply as a Network Object," in this conference proceedings.

[3] L. J. Chapman, "Object-Oriented Communications," in Proceedings of the 1989 IEEE Particle Accelerator Conference, pp. 1631-2.

[4] B. Stroustrup, The C++ Programming Language , Addison Wesley, 1986.