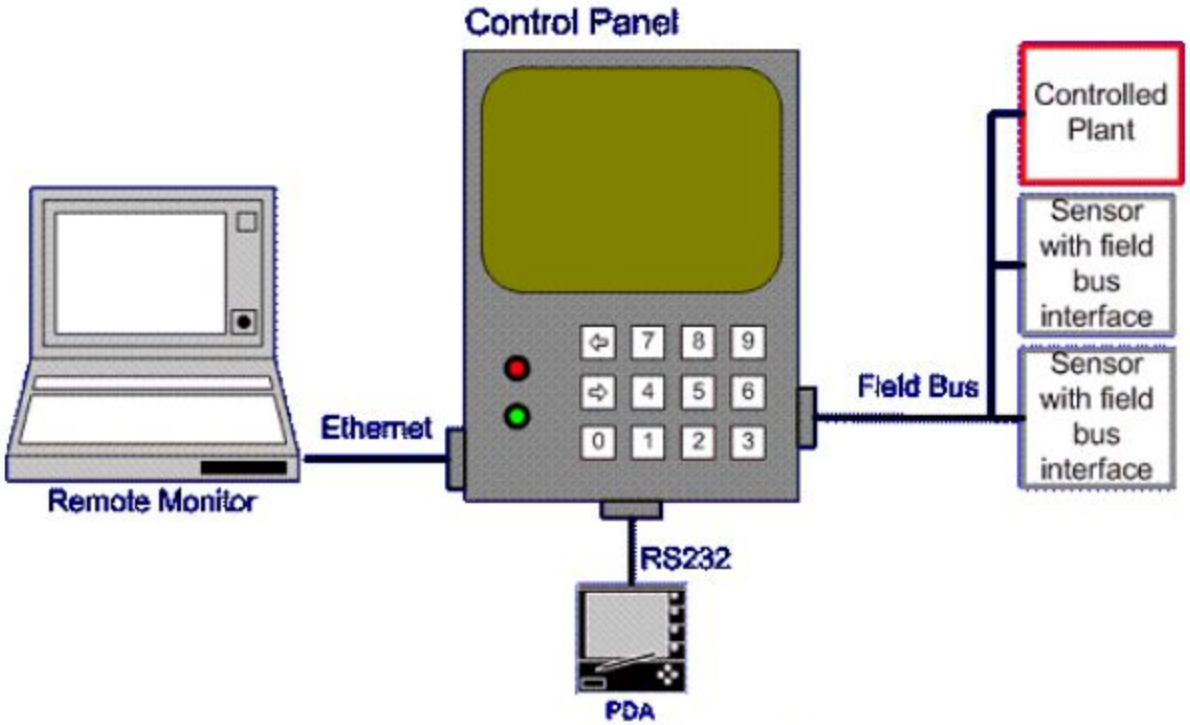
Ejemplo de sistema embebido real time

(sin utilizar RTOS - diseño tradicional)

The [hypothetical] Application



System context [not to scale].

The application will execute on an embedded single board computer that must control a plant while maintaining both local and remote user interfaces.

Depicted above, the system consists of:

1. An embedded computer within a control terminal.
2. Two fieldbus networked sensors.
3. The plant being controlled (could be anything, motor, heater, etc.). This is connected on the same fieldbus network.
4. A matrix keypad that is scanned using general purpose IO.
5. Two LED indicators.
6. An LCD display.
7. An embedded web server to which a remote monitoring computer can attach.
8. An RS232 interface to a configuration utility that runs on a PDA.

Plant Control

Each control cycle shall perform the following sequence:

1. Transmit a frame on the fieldbus to request data from the networked sensors.
2. Wait to receive data from both sensors.
3. Execute the control algorithm.
4. Transmit a command to the plant.

The control function of the embedded computer shall transmit a request every 10ms exactly, and the resultant command shall be transmitted within 5ms of this request. The control algorithm is reliant on accurate timing, it is therefore paramount that these timing requirements are met.

Local Operator Interface [keypad and LCD]

The keypad and LCD can be used by the operator to select, view and modify system data. The operator interface shall function while the plant is being controlled.

To ensure no key presses are missed the keypad shall be scanned at least every 15ms. The LCD shall update within 50ms of a key being pressed.

LED

The LED shall be used to indicate the system status. A flashing green LED shall indicate that the system is running as expected. A flashing red LED shall indicate a fault condition.

The correct LED shall flash on and off once ever second. This flash rate shall be maintained to within 50ms.

RS232 PDA Interface

The PDA RS232 interface shall be capable of viewing and accessing the same data as the local operator interface, and the same timing constraints apply - discounting any data transmission times.

TCP/IP Interface

The embedded web server shall service HTTP requests within one second.

The timing requirements of the hypothetical system can be split into three categories:

1. **Strict timing** - the plant control

   The control function has a very strict timing requirement as it must execute every 10ms.

2. **Flexible timing** - the LED

   While the LED outputs have both maximum and minimum time constraints, there is a large timing band within which they can function.

3. **Deadline only timing** - the human interfaces

   This includes the keypad, LCD, RS232 and TCP/IP Ethernet communications.

   The human interface functions have a different type of timing requirement as only a maximum limit is specified. For example, the keypad must be scanned at least every 10ms, but any rate up to 10ms is acceptable.

Implementation

This solution uses a traditional infinite loop approach, whereby each component of the application is represented by a function that executes to completion.

Ideally a hardware timer would be used to schedule the time critical plant control function. However, having to wait for the arrival of data and the complex calculation performed make the control function unsuitable for execution within an interrupt service routine.

Concept of Operation

The frequency and order in which the components are called within the infinite loop can be modified to introduce some prioritisation. A couple of such sequencing alternatives are provided in the example below.

The Human Interface Functions

This includes the keypad, LCD, RS232 communications and embedded web server.

The following pseudo code represents a simple infinite loop structure for controlling these interfaces.

```
int main( void )
{
    Initialise();

    for( ;; )
    {
        ScanKeypad();
        UpdateLCD();
        ProcessRS232Characters();
        ProcessHTTPRequests();
    }

    // Should never get here.
    return 0;
}
```

This assumes two things: First, The communications IO is buffered by interrupt service routines so peripherals do not require polling. Second, the individual function calls within the loop execute quickly enough for all the maximum timing requirements to be met.

```
int main( void )
{
    Initialise();

    for( ;; )
    {
        // Spin until it is time for the next
        // cycle.
        if( TimerExpired )
        {
            PlantControlCycle();
            TimerExpired = false;

            ScanKeypad();
            UpdateLCD();

            // The LEDs could use a count of
            // the number of interrupts, or a
            // different timer.
            ProcessLEDs();

            // Comms buffers must be large
            // enough to hold 10ms worth of
            // data.
            ProcessRS232Characters();
            ProcessHTTPRequests();
        }

        // The processor can be put to sleep
        // here provided it is woken by any
        // interrupt.
    }
```

En un diseño tradicional
(bucle infinito + interrupciones) Tendríamos:

- Una ISR que cada diez milisegundos activa el flag TimerExpired (necesario para la función de tiempo estricto del control de la planta)
- Una ISR para obtener datos desde HTTP y que los coloca en un buffer (lo procesa luego la función ProcesssHTTPResquests).
- Una ISR para obtener datos desde RS232 y que los coloca en un buffer (los procesa luego la función ProcessRS232Characters).

Este diseño tradicional “podría” funcionar
(cumplir con los requisitos real time). Peero,

- Es dificil de escalar (agregar mas funcionalidad)
- Es dificil verificar si se cumplen los tiempos de respuesta (algunas funciones dependen del tiempo de demora de otras, aún si son funciones independientes)..
- Es dificil de diseñar. El código es dificil de leer.
- Etc.