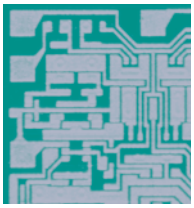
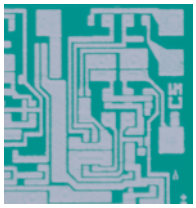
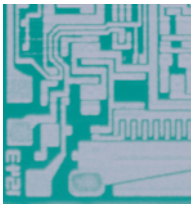
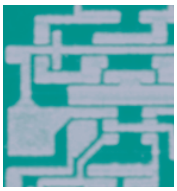




The hard and soft option



Bus your way around LabMap to obtain the ultimate auto test platform

by D. Kazakov, A. Fechner, & C. Bruce-Boye, cbb software GmbH

Modern automation structures require new approaches to smoothly integrate a great variety of hardware and software standards. Such integration can be achieved only if the application level software is abstracted from the specifics of the underlying protocols. An instance of such an approach is the IDA initiative [1].

A ready-to-use approach is offered by the LabMap software bus system. LabMap responds to the problem by abstraction of the application level from the hardware specific, and decoupling the hardware interface modules from the application level. In other words, there are two levels of abstraction supported by the bus system (an example in the automotive application area is Figure 1).

The application interface allows the developer to concentrate on its domain issues. This interface hides not only the hardware specific, but also the configuration issues. The goal is to allow writing automation application with minimum effort. It is the hardware driver interface that allows the developer to integrate new hardware or software protocols into the system, while the software bus is viewed by an application as a set of variables. Each variable has a type, the current value and the current timestamp – not requiring any configuration. An application may access a value immediately after the system start. The following basic I/O requests are provided for each variable:

Get: The value of a variable can be read in a safe way. It is guaranteed that the read value parts and the timestamp are consistent. The application is relieved of the burden of locking the value in the presence of concurrent tasks accessing it. The application is unaware of the source of the value and the policy used to actualize or obtain the value.

Set: The value of a variable can be set in a safe way.

Request: A new value of a variable can be requested from the underlying hardware. The application does not

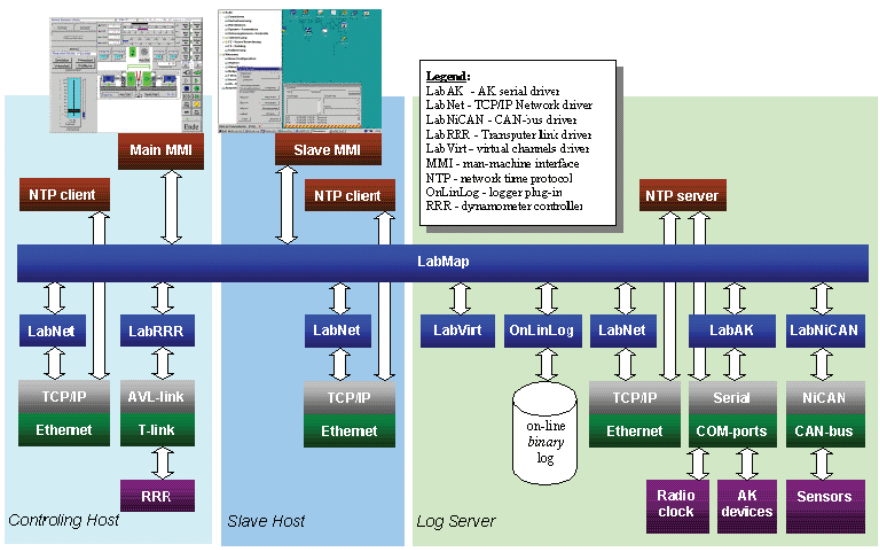


Figure 1: A working example of a roller dynamometer automation project

know which actions are necessary to request the new value. It is therefore the responsibility of the driver. The request is asynchronous and the application is not blocked until I/O completion. However, it may enter a non-busy waiting for the I/O completion.

Send: The value of a variable can be sent to the underlying hardware. In the case of a request, the application is unaware of the actions that the hardware undertakes when sending. The application may again enter a non-busy waiting for the I/O completion.

The software bus offers a variety of synchronization mechanisms to the application. For example, one needs to wait for I/O completion: an application may enter time-limited waiting for completion of I/O involving a variable.

Then there's the wait for value change. An application may enter time-limited waiting for the moment that the value of a variable is changed. This method is very often used for monitoring system state variables. The main advantage of this approach is that the application developer is relieved from the burden of polling the variable value and may concentrate on domain objectives.

Next, we move on to the blackboard. Sometimes it is necessary to trace all variable changes. For instance, an application visualizing signal waveforms would like to trace the signal. The usual approach for catching value changes involves a notification mechanism between the source of the value and the application. Such point-to-point bias is hard to implement without overstraining the system resources and there is a danger of resource leaks when an application ends abnormally. The software bus uses an alternative approach. All value changes are written on to the blackboard and remain for a certain time. Any application may inspect the blackboard contents in order to trace the changes of desired variables.

The hardware abstraction interface uses the messaging mechanism for decoupling the application from the communication software. A hardware driver's responsibility is to provide values of the variables. Any variable has a hardware driver attached to it. The driver receives notification messages upon interface start/stop and I/O start/stop. For the values requested from the hardware, the following I/O strategies are supported: on request – a new value is requested from the hardware only on an explicit request of an application; polling – a new value is requested periodically; and on change – a new value is requested when the corresponding physical variable value is due to change. The key features of the software bus include:

Decoupling the application level from the hardware: The underlying hardware can be exchanged without modifying the application software. Furthermore, there are integrated measurement units to support where variables of floating-types have measurement units attached. An application may get the value in SI units or in the desired units compatible with the units used by the hardware driver.

Distributed systems support: The system architecture allows the systems distributed to be easily built on the LAN or WAN. For example, there is a hardware interface to the TCP/IP based networks called LabNet.

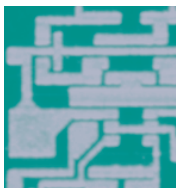
Mixed networking: The system hides the networking details behind the hardware interface drivers. Therefore, a distributed system could be built on any hardware basis. It could be a field bus, Ethernet or a mixture of different transport protocols.

Stress on safety: An application may ensure that no other application accesses a variable to write. The software bus distinguishes local and global variables. Local variables are seen by the applications running on the same network host where the variable is declared. Global variables are accessible from any host. Access operations on global variables are also global.

Data consistency: Any value is provided with the time stamp. A hardware driver is responsible for supplying correct time stamps. An application may figure out that several values are consistent by inspecting their time stamps.

Ready for real-time: The software bus does not proclaim itself being real-time, though a soft real-time can be easily achieved if the corresponding hardware meets real-time requirements.

Integrated on-line logging support: All the data broadcast via the software bus can be automatically logged. Log writing occurs on demand, no polling is used. Each record is provided with the time stamp. In other words, no data or information is lost. The off-line post processing has an ability to restore the system state to any past moment of time.



Integrated on-line monitoring: Any variable can be monitored on fly. The monitoring software allows graphic and digital inspection and visualization of the variables. It is also possible to view the hardware I/O.

Virtual channels: Variable values can be obtained using the powerful set of formulae.

The software bus LabMap is being successfully applied the automotive area[2]. It currently supports a variety of software and hardware protocols, such as OPC, AK, Modbus, CAN, etc.

References

1. *M. Schneider, J. Pisarz & S. Thiebeaut. Ethernet-Newsletter – To whom it concerns. IEE automatisierung+ datentechnik, No. 10, Oct. 2000.*
2. *Müller et al. A smooth ride. Testing Technology International, Nov. 2000.*