A Fresh Look at Embedded System's Design and Programming

C and to some extent C++ is "The Language of Embedded". However, as Jack Ganssle mentions in how embedded projects run into trouble, undisciplined use of C and C++ can be very costly. Large projects comprised of programmers with varying skills routinely run into delays and project cost increases. Many of the delays are related to C and C++ problems such as buffer overflows, memory leaks, and other memory corruption errors.

Modeling Embedded Systems

For most embedded systems, the software may be modeled as follows:

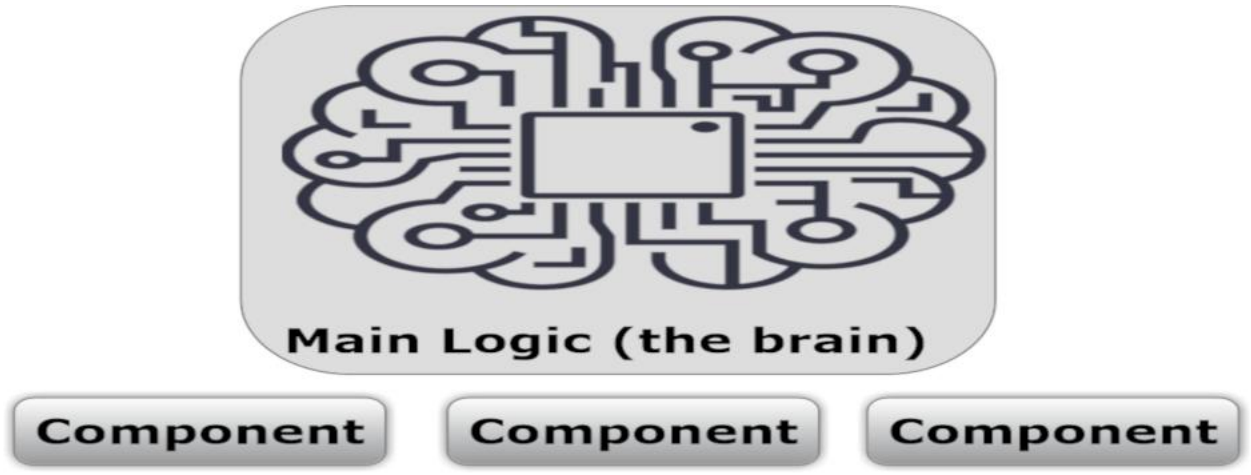


Figure 1: Abstract embedded system's component diagram (hardware not included)

The components in Figure 1 can be anything from I/O units to logic with real time requirements. In most cases, it is possible to design the main logic without strict real time requirements if the embedded system is carefully modeled. The main logic has a tendency to be large and complex, and, in this tutorial, we will look into methods that shorten the development time of the main logic, thus reducing the overall cost.

C is a fantastic high level language. However it is nowhere near as high level as Python, JavaScript, or Lua. The syntax and semantics of C is amazingly powerful and expressive, but with power comes great responsibilities. As we mentioned above, undisciplined use of C and C++ can be very costly. For many projects, a much higher level language than C/C++ can greatly reduce the project cost when many developers are involved.

When selecting a higher level language for the design of the main logic, one must consider how well the higher level language integrates with the C code, the size of the higher level language's virtual machine, and the speed of the executed code. Python, JavaScript, and Lua are all used in embedded systems; however, no language can compete with Lua when it comes to small size and speed. Lua is also designed to be an embedded component and is compiled into a C library that can easily be linked with the embedded system's software/firmware. When we add Lua to the embedded system's component diagram in Figure 1, we get the following:

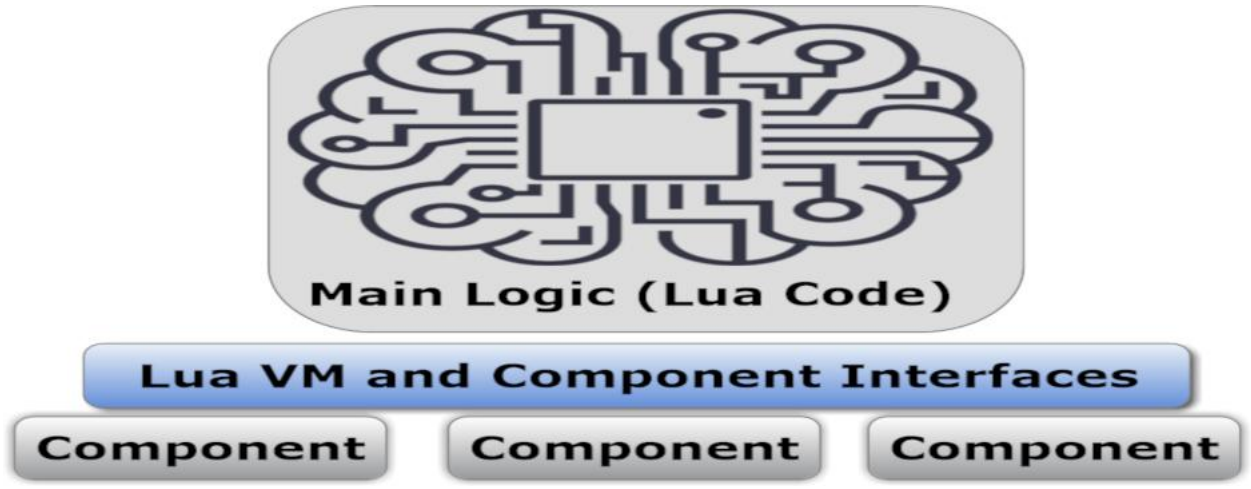


Figure 2: Embedded system's component diagram using mixed programming languages

In Figure 1, all code would have been designed in C and/or C++. In Figure 2, the main Logic is implemented in Lua, the Lua VM is implemented in C, and the component interfaces and all components are implemented in C and/or C++.

The component interfaces in Lua terminology are referred to as Lua bindings, which is the glue that makes it possible for Lua code to call functions in C code and vice versa. Lua bindings can be designed manually or automatically from an interface specification. For more information on Lua bindings, see our online interactive Lua binding tutorial and our online automatic Lua binding generator.

Embedded Systems vs. Games

Lua is "the language of games" and advanced multiplayer games, such as World of Warcraft, are designed similar to the concept proposed in Figure 2. Game developers benefit from rapid game logic development by implementing the core logic using Lua. The real-time game components are implemented in C code and controlled from Lua. Wikipedia lists close to 150 games that use Lua in addition to many other products embedding Lua.

Stock Lua

Although Lua is used in embedded systems and is linked with other code as a C library, the stock Lua at lua.org is not ready for most embedded systems. A detailed understanding of the Lua code base is required for embedded system porting purposes. Also, stock Lua is limited in functionality. In essence, stock Lua provides the Lua VM (C library) and I/O that must be ported to most embedded systems.

Lua with Additional Features for Embedded Systems

Real Time Logic provides an embedded system ready Lua version with extended functionality. As an integral component of the Barracuda App Server (BAS), Lua is referred to as Lua Server Pages and is not limited to web development. BAS provides compact and efficient non-blocking asynchronous sockets and WebSockets, SMTP, HTTP client/server, IoT protocols such as MQTT, industrial protocols such as Modbus, security modules including TLS, and much more, and all available to Lua.

Customer Example: Lua Powered Poka-yoke aka Mistake-Proofing System

Although most of Real Time Logic's customers using BAS design some kind of web interface and/or IoT service, there are use cases where BAS is used in products without a web interface. For example, Mitsubishi Electric designed a human error prevention system for one of their industrial customers, where all system logic was designed in the Lua language. The following video shows the human error prevention system in action.

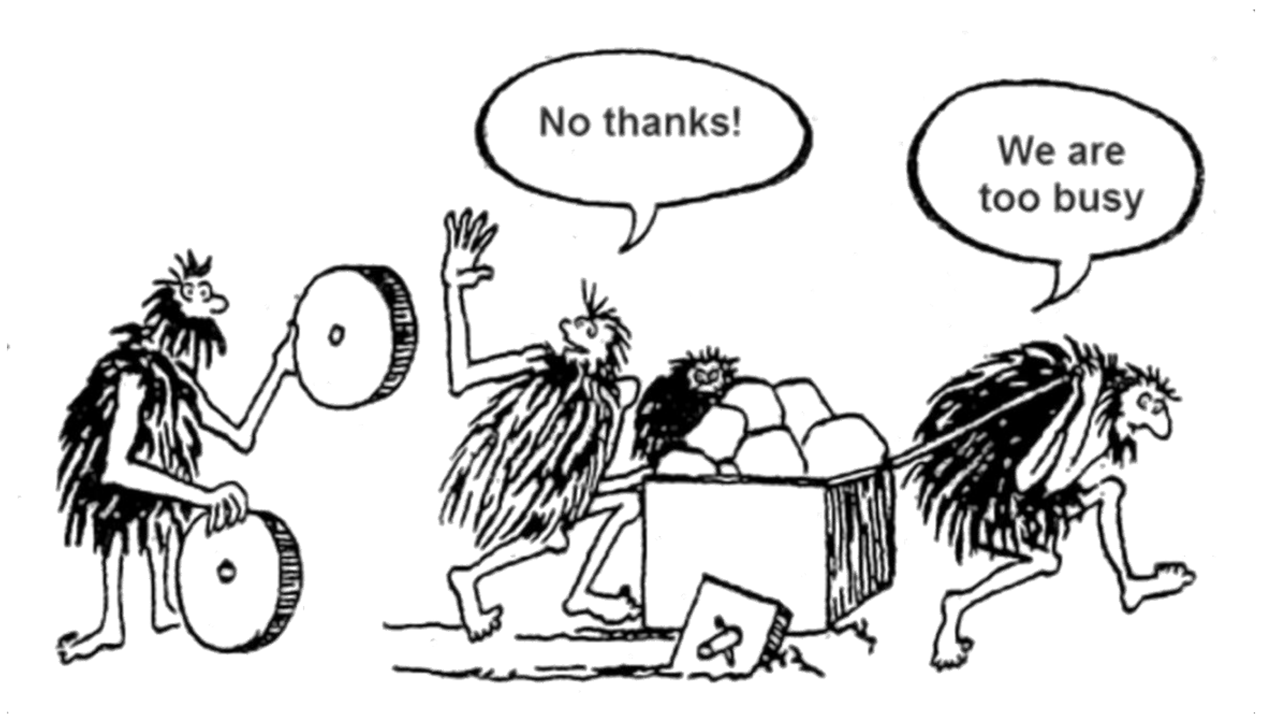
VIDEO URL: https://youtu.be/fozEH1sO-zQ

Video 1: Lua powered human error prevention system (Japanese: Poka-yoke)

Poka-yoke is a Japanese term that means "mistake-proofing" or "error prevention". Poka-yoke prevents human assembly errors during manufacturing. Picking is a key area where production efficiency can be improved and where product quality can be enhanced. The Lua powered guided operator solution significantly reduces overall man-hours, eliminates mistakes in parts picking, and also reduces operator fatigue. Just as game developers benefit from rapid game logic development by implementing the core logic using Lua, so do the engineers designing and customizing the Poka-yoke system in Lua.

Additional Lua Material:

- Lua Fast-Tracks Embedded Web Application Development
- Online Interactive Lua Tutorial
- Online Barracuda App Server Lua Documentation
- Designing Socket Protocols in Lua
- Lua Server Pages Product Page



C programmers may sometimes be too busy to see the opportunity.