Icon

Description automatically generated

Feedback Driven Development

Stephan Bourgeois

President and CTO

MicroMaster LLC

11/17/2021

Contents

[Purpose 3](#_Toc85442151)

[Introduction 3](#_Toc85442152)

[Objective 4](#_Toc85442153)

[Overview 5](#_Toc85442154)

[Development Life Cycle 6](#_Toc85442155)

[Example 6](#_Toc85442156)

[Conclusion 9](#_Toc85442157)

# Purpose

The purpose of this document is to present a novel approach to embedded firmware development called Feedback Driven Development, which has the potential to revolutionize the way that complex embedded firmware is developed.

# Introduction

Over the years since the inception of computers, many different software development strategies have been theorized and put into practice. Although many different names for these strategies have been coined, they basically boil down into three camps. [Agile](https://docs.microsoft.com/en-us/devops/plan/what-is-agile) development is a term used to describe iterative software development. Iterative development breaks the up the development cycle into sprints that usually last 1-4 weeks. After each sprint, the development strategy is revaluated and reformed, making the development process agile in that the plan is subject to change. The opposing camp uses a strategy called the waterfall model, which involves carefully planning out the software before it is written and then executed. Recently though, a new strategy has emerged which is becoming more and more popular, called Test Driven Development. Test Driven Development involves writing tests before the actual code is written, then the code is written to pass the tests. Although this strategy is cumbersome, it is very effective in reducing bugs, which can decrease development time. All of these strategies are proven effective for software engineering on computers with operating systems, however, embedded firmware development presents a number of challenges which make these strategies difficult to implement and developers often revert back to the classic “build and fix” model, whether they intended to or not.

The biggest challenge one faces when developing embedded systems is that there is no graphical screen or console. When developing an application for a computer, a software developer gets instant feedback on whether or not their program is working correctly every time they execute the program, because the program manifests itself on the screen. A software developer can test their program simply by interacting with it and seeing if it works. However, with an embedded system, an embedded firmware developer only usually has a serial terminal. When designing a serial debug terminal to “see” into the microcontroller, firmware developers have had two options. They can simply spew all the debug information out of a serial terminal in a disorganized manner, or they can create a serial command line interface. The more complex an embedded system becomes, the more difficult it is to implement either of these strategies to gain visibility into the microcontroller. Command line interfaces are notoriously difficult to program and spewing all the information out to the screen can get extremely disorganized and confusing.

Test Driven Development has taken hold of the software development world and is generally considered the best development strategy, because every function and discrete operation of a program is thoroughly tested as it is being developed, greatly reducing the time it takes to test a device and increasing the reliability of the program. While that sounds great in theory, it is often extremely difficult to implement in practice on embedded systems for many reasons. The first and foremost is that microcontrollers have complex peripherals and hardware that are very difficult to model in a test driven environment. Emulating a hardware device and writing a test for it can often be more difficult than writing the program itself. Furthermore, as embedded systems become complex and these peripherals and hardware are interconnected, it becomes nearly impossible to write a test that considers all of the permutations of inputs and outputs. In addition, Test Driven Development does not lend itself to good architecture that is easily updateable and repeatable.

The development of embedded systems adds a layer of complexity that does not apply to typical software development. Software development is usually done on a pre-existing system, with hardware that is tested and proven. However, embedded systems usually require custom, brand new hardware that is unproven and often times, developed in parallel with the firmware. An embedded systems developer not only has to consider whether or not their program is operating correctly, but whether or not the hardware is working correctly as well. This is where test driven development falls short and developers have to revert back to the build and fix model as they start integrating the parts they have developed through test driven development. At some point in the design process, development strategies go straight out the window and developers have to rely on experience and the art of engineering to get a complex embedded system to crystalize into a working program.

Well organized architecture is essential for complex embedded firmware, especially when multiple programmers are working on the same program. The program must be divided up in an organized manner into block diagrams with the interfaces of every module well defined, so each individual programmer can work on a discrete part of a program. When trying to implement a serial debug terminal, this can get messy. Integrating a spewing style debug terminal or a command line interface is very difficult in this environment, because each programmer is adding to the complexity of the serial debug terminal and either not considering the cumulative effect on it or spending an inordinate amount of time trying to get it to work correctly. At the end of a complex embedded system development cycle, the serial interface is usually an absolute mess and unusable in the post development testing phase. However, if it was well done and well organized, it could be used throughout the entire lifecycle of the device, from development, to testing, to diagnostics in the field.

# Objective

This document will demonstrate the feasibility of a new development strategy for embedded systems coined “Feedback Driven Development”, which addresses all of the problems that embedded developers face when architecting and programing complex embedded systems. The implementation of Feedback Driven Development through a serial terminal drastically reduces development time and vastly improves testing compared to other development strategies for embedded systems.

# Overview

In order for Feedback Driven Development to be implemented, the serial terminal must be implemented in a way that is well organized, navigable, and functional. This is done by creating a system of “pages” in the serial terminal that can be updated in real time as well as take input and output. This system of pages mimics a computer application or web browser, allowing for instant and coherent feedback all throughout the design process, the same way that a software developer can get feedback from a GUI application on a processor with a large operating system. Every time a firmware developer runs their program on a microcontroller, they can quickly inspect the serial terminal to see if it is operating correctly. Instead of writing static tests for individual functions, the serial terminal serves as a testing platform embedded into the code of the device.

The key to Feedback Driven Development for embedded systems is that the entire program is architected around the serial interface. The program is broken down into classes containing tasks that constitute a module which encompasses a peripheral such as an I2C or an SPI driver. Each module has its own “page” in the serial terminal. The page can be customized to display information that can update in real time, giving the developer visibility into what is happening in the microcontroller. While it is possible to gain visibility into a microcontroller with a debugger by observing the dynamic memory, this process is cumbersome and takes a toll on the processor itself. However, as long as the first page of the serial terminal is static, no toll is taken on the processor until the user selects the appropriate page. The architecture of feedback driven development is conducive to easily splitting the program up into discrete and testable parts that can be easily integrated into an entire program.

In addition to modules having their own pages, additional modules can be created to help debug the device and aid in the design process, such as an error logger, a task manager, gpio control, and self-tests. A well designed error logger can be used not only in the design process, but throughout the entire lifecycle of the device, allowing embedded systems to be easily diagnosed when they are in the field. A task manager allows the developer to track all of the memory usage throughout the device, eliminating the need for complex debugging applications and tools. The ability to manually control GPIOs and monitor ADCs allows a custom board to be easily tested for functionality when it gets back from the factory. A developer can quickly confirm and test the hardware through the serial interface. Having self tests which can be initiated through the serial interface allows the developer to easily test their device in different scenarios.

Feedback Driven Development can easily be integrated with other development strategies like test driven development, agile development, and the waterfall module. In practice, it combines all these methods into a coherent architecture that has all of the benefits of each method, with none of the drawbacks. Feedback Driven Development is more than just a strategy, it is a concrete architecture that is versatile enough to be incorporated into any complex embedded system. Feedback Driven Development drastically reduces development time, improves testability, allows visibility into the microcontroller.

# Development Life Cycle

1. Bring up the microcontroller and install the serial console module
2. Create an error logger module, task manager, GPIO and ADC control
3. Divide up all the task objects and define the interfaces
4. Write tasks object
   1. Write the code
   2. Observe the serial interface for correct operation
   3. Refactor and repeat
5. Embed tests into the code which throw errors in the error logger
6. Write self-test modules
7. Install program on custom board
   1. Manually Test GPIOs through IO control page
   2. Check each page and module for correct functionality
   3. Field test device, observing the error logger

# Example

This section will demonstrate the serial terminal of the MicroMaster Mini, which has an extremely complex serial terminal interface, because it has been designed to be a human machine interface.

Text

Description automatically generated

**Figure 1. MicroMaster Mini Home Page**

The program starts out with the user pressing any key then this homepage is displayed (Figure 1). There are two ways to navigate the pages. The user can press tab to toggle through the pages or they can press a letter to jump to a page. Pressing ESC will return the user to the homepage.

A screenshot of a computer

Description automatically generated with low confidence

**Figure 2. MicroMaster Mini IO control page**

Figure 2 shows the IO control page for the MicroMaster Mini. The General Purpose Outputs (GPO) can be manually controlled with the arrow keys. The user can select the GPO with the up and down arrow keys and turn it on an off with the left and right arrow keys. The GPIs and ADCs can also be observed on this page.

Text

Description automatically generated

Figure 3. MicroMaster Mini Error Logger

Figure 3 displays the MicroMaster Mini Error Logger. In this case, the 1 wire module was attempted to be used without a pullup resistor attached. The error logger gives the name, information and number of occurrences. Additional debug information could be added to this error logger such as the last time the error occurred.

Text

Description automatically generated

**Figure 4. MicroMaster Mini Task Manager**

Figure 4. shows the MicroMaster Mini Task Manager. At the top of the page, the heap size and heap used are listed. After that, it lists off the tasks, giving their status, priority and the size of the unused stack.

# Conclusion

Feedback Driven Development is a revolutionary way to program microcontrollers that allows for ultra-rapid development by providing the programmer with instant feedback every time they run their program, allowing a single programmer to swiftly architect complex embedded programs for medium to large sized microcontrollers (>12MB of RAM). Feedback driven development incorporates all of the features of popular development strategies with all of the benefits and none of the drawbacks. Feedback Driven Development is the most efficient and organized way to architect complex embedded firmware. In addition to being used to develop the device, the serial interface in FDD is instrumental in the testing and life cycle of the device. Feedback Driven Development is a holistic approach to firmware development that is based upon a well-defined architecture which improves the experience not just for the programmer, but the entire engineering team.

In addition to being a rapid development method that incorporates principles of Test Driven Development, the serial interface used for Feedback Driven Development has many applications as an actual human machine interface to an embedded device, like the MicroMaster Mini. Debuggers, flash tools, bootloaders, BIOS systems, and test bench devices are all possible ways in which the serial interface can be used as an HMI. Complex embedded systems can greatly benefit from Feedback Driven Development by allowing the testing to continue throughout the entire life cycle of the device, not just the development phase.