Quick notes with report headings

Niklas Blomqvist, Robin Gustafsson

Contents

Revisions

Version	Date	Sign off	Change note
0.1	2015-02-23	Robin, Niklas	Quick jots
			ida.liu.se/edu/ugrad/thesis/instructions/Exjobb_anvisning.pdf

Abstract

Unique customizations (options) of a fork lift trucks features are often requested by the customer. When new options have to be created or present options have to be modified in the main software the complexity increases, the firmware revision pool gets large and with the increasing code size the memory limit is threatened. This affects the software development since the frequent modification of the option handler software is very resource consuming. Therefore it is desirable to have a highly modular system for the option handler to reduce the development process. Although the market value of this improvement is negligible the possible long term savings is the desirable effect. The purpose of this thesis is to explore the possibility of migrating the option handling software to a dedicated hardware module. This will help the development process by increasing the modularity of the system architecture and thus reducing the development scope. Methods for model based development will be utilized to explore ways to efficiently speed up the software development process. The terms of inclusion and the tools to accomplish this option handler is analyzed. A system model of the resulting approach will be designed and a prototype will be developed to validate the result.

Introduction

We have conducted our thesis work at a big fork lift manufacturer located in Östergötland. In this report it will be called “The company”. In this section we will present the motivation behind this project as well as the purpose. The initial problems for the project will be specified and the delimitations needed to be able complete the project within the scope.

Motivation

A large quantity of the sold fork lifts are equipped with non-standard options requested by the customer. These options are all implemented in the firmware that controls the truck. The company currently has no way of decouple the option implementation from the main firmware. This means pollution of the source code tree as separate branches has to be created for each customer specific option. This also means that there are multiple variants of the same version of program code that needs to be maintained.

Purpose

In order to satisfy the increasing customer demand of new features (options), The company needs a faster, more reliable and testable way to develop them. Currently, options are added to the main firmware. This creates the problem where many branches of the firmware has to be created and it gets more difficult and time consuming to create patches and updates as the size of the firmware pool expands.

This thesis aims to decouple the options implementation from the main firmware and dedicate a separate unit to handle the options in order to speed up development of new features, and decrease the number of potential bugs in the main firmware. By doing this we achieve a more modular system.

Problem

- How are the options handled currently?
- How are the options stored internally, what data structures and are they applicable on the new module?
- What type of hardware do we need to add, what are the requirements?
- What needs to be taken in consideration when designing the new system model?
- How do we validate the results, what tools do we use?
- How is the CAN bus affected if additional controllers are added as The company runs the bus at 125 kbit per second.
- Do we need to make modifications on the current CAN-bus communication protocol?

Delimitations

The time will not be sufficient to develop a full scale version of the options handling. With respect to that, we have chosen to spend most of the time developing a working architecture, and a prototype. The prototype will be written with flexibility in mind. This meaning it will be written in such way that it should be easy to extend with new features such as a GUI[ˆGraphical User Interface]. From a testing perspective, this is the natural way to go.

The fundamental part of this thesis is the development of an architecture as general as possible. It is therefore not vital that we implement all the existing options, as long as the architecture can be deemed good enough to handle them. Then we might choose a few options, preferably some of the more vital, to implement for validation purposes.

Further, one possible delimitation might be to hand off the MCU side of the development to The company. This option, however, depends on how much time The company can spare. This delimitation is only applicable if we decide to put the options handling in an external chip. In this case, all of the options currently existing shall be implemented.

The communication between the MCU and the intended extra unit will occur over the CAN bus. This means that the protocol must be implemented in the prototype. The results will be validated with a HIL (Hardware In the Loop) system and/or with a truck.

Background (optional)

Today, The company handles a big quantity of customer specific options on their fork lifts. The options are all being built in into the main controller (MCU) of the truck. This leads to problems:

- Developing the options requires a lot of resources, this because it began as a "one-off job".
- Because all of the features exists in the main firmware, the code becomes very complex and hard to follow. Many of the features are also inactivated for most of the customers.
- The available code memory will soon be filled. Adding additional code will require a larger on chip memory.

The company is looking for an options handling solution which allows the functionality to be moved from the MCU to a separate controller.

We will work out a solution where the options handling will allow development of functionality, independent of the main firmware, in a more modular fashion. This will allow parameter based configuration without the need of rewriting code. It is important that all existing options is handled properly. This will imply a reduction in time needed to develop new features.

A graphical interface is desired in order to simplify the administration of options without the need of deep programming knowledge. A *PLC representation* would give the user a good overview of active options and also the possibility to customize parameters. It is also important that the options handling is secure in a way that ensures that no unauthorized person can tamper with it.

The long term goal is to incorporate the possibility to customize into the standard software. Currently there is no significant market value in the options handling it self, but in the long term there will be. Both in time savings as well as in product quality. This will give The company Products a better foundation to decide on the possibility to include this in the control units of the trucks.

Related work

We will discuss work done, mostly by The company, as well as related academic work.

Existing work

In this big project scope we decided to utilize model based development to our advantage in the hope of being able to implement as much of the desired functionality as possible and document the project well. Model based development allows for the project to bee fractioned to smaller tasks and thus aids the prototyping process and simplifies high level software development. It also greatly simplifies the planing and project flow which leads to more efficient resource management, which is the goal of our use of model based development. The overall software quality is strengthen when following a model and this help us reach the high quality standard of the software developed by the company.

The model based development approach also helps motivation within the project since the development from idea or model to finished prototype is the main goal. The model can be done relatively early in the project which help to start the project compared to cold turkey style development.

Another advantage to the model based development is that the model is universal to all platforms and programming languages so in our case when developing sub-prototypes we can develop it in C++ and then later convert it easily to C for the final prototype.

By adding another hardware unit to the existing modular hardware architecture we reuse the present CAN-protocol available within the standard system. The major modification is to the embedded CAN module where the API had to be modified to bee able to execute the new instructions needed when the option handler were migrated to another external hardware-unit on the other side of the CAN-bus. All the core functionality to every system operation are available on the MCU and would bee unnecessary to migrate so the only CAN-bus traffic needed for the model to operate is call to operation instruction algorithms and modification of data parameters. The Option handler is interrupt driven in the sense of a trigger event sparking an internal interrupt immediately followed by the corresponding Instruction. The validity of the instruction is controlled on the MCU side and otherwise a error package is returned on the CAN-bus.

Method

Feasibility Study

Currently, The company has an options handling where a trigger[1] or override[2] is used.

Currently, these triggers and overrides are checked by functions called from the main loop. This is not ideal considering that a firmware update needs to cascade to all of the branches of the firmware that might exist.

FIND OUT HOW IT IS IMPLEMENTED

**It is implemented through. . . *

The company wishes to explore new ways of model based development and relating tools. Designing the system model will require us to think ahead and make the model basic enough to follow at the implementation phase of our prototype. The model might change if we run into a problem during the implementation phase but it is of great importance to break the system down to smaller components in advance.

The standard software which we will be expanding has a well documented system model including TLM[ˆTransaction-level modeling] and a complete transaction processing chart of all the processes and algorithms. It is only fair if we add our system to this in a similar format as an expansion.

Complete system model

To reduce the overhead on the CAN-bus the ideal solution is to let the option handler remotely modify the parameters of the MCU on the defined trigger event.

[1]Button pressed, speed under/above, etc.
[2]Service key is used, reduce drive speed, etc.

Implementation

Evaluation

Result

Discussion

Result

Method

Conclusions

Citations