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

Customizations of the functionality (options) are often requested by the customer and have to be acknowledged in the manufacturing process of fork lifts. When new options have to be created or present option algorithms 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 is very resource consuming. Therefore it is desireable to have a very modular system for the option handler to simplify the development process. Although the market value of this improvement is neglecteble the possible long term savings is the desireble effect. The purpose of this thesis is to explore the posibility of migrating the option handling software to a separate hardware module to help the development process by effectivly increasing the modilarity of the system architecture aswell as exploring new methods for model based development.

(fyll på mer om uppnått resultat efter projektet)

/ DETTA KANSKE SKA HAMNA UNDER METOD? /

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.

"En sammanfattning (abstract) ska kort och koncist beskriva och motivera det studerade *problemet, metoden samt resultat och slutsatser. Arbetets bidrag till huvudområdet ska tydligt framgå. Vad är det rapporten säger om huvudområdet som vi inte visste tidigare? Exempel på bidrag kan vara vilken effekt en specifik algoritm eller programutvecklingsmetod får i en specifik tillämpning. Normalt ska en sammanfattning vara högst 150 ord, och inte innehålla några* _referenser eller radbrytningar."_

Introduction

We have conducted our thesis work at a big fork lift manufacturer located in Östergötland. We have been asked not to call it by name in text. In this report it will be called "The company".

Motivation

A large quantity of the sold fork lifts is equipped with non-standard options requested by the customer. These options are all implemented in the firmware that controlls 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.

Här ska det studerade problemet översiktligt beskrivas och sättas in i ett sammanhang som gör det tydligt att det är intressant och viktigt att studera närmare. Målsättningen är att göra läsaren intresserad av arbetet och skapa en vilja att läsa vidare.

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. Now, 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 work aims to decouple the options implementation from the main firmware and dedicate a separate unit for just option handling in order to speed up development of new features, and decrease the number of potetional bugs in the main firmware. By doing this we achive a more modular system.

Vad är det som examensarbetet ska leda till?

Problem

- How is the options handled currently?
- how are the options stored internaly, 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 will the CAN bus be affected if additional controllers is added as The company runs the bus in the slowest speed according to the CAN bus standard.
- do the CAN-bus communication protocol need modification?

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 graphical 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 may tamper with it.

The long term goal is to incorporate the possibility to customize into the standard software. 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.

Ibland bygger ett examensarbete på ett specifikt uppdrag vilket kan göra det svårt att ge hela sammanhanget i inledningskapitlet utan att det blir för långrandigt (inledningen ska ju väcka läsarens intresse). Då kan ett bakgrundskapitel användas för att ge en mer detaljerad beskrivning av själva uppdraget. Det kan till exempel handla om någon form av kravspecifikation eller dylikt. Detta kapitel ska enbart användas vid behov.

Related work

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

Existing work

Method

Feasibility Study

Currently, The company has an options handling where a trigger[1] or override[2] is used. **FIND OUT HOW IT IS IMPLEMENTED**

**It is implemented through... *

Analys över hur det ser ut nu. inkl kod & teori bakom lösningen

Modell baserat utvecklingsrelaterat arbete

The company desires 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 may be changed 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. (länk till relaterad artickel om riktlinjer för modell based development).

The standard software wich we will be expanding has a well documented system model inkluding TLM (förklara förkorttning) 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.

(förklara möjliga vektyg för att ta fram modellerna)

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

Projektet kommer ha ett antal milstolpar iform av mindre delprototyper för att itterativt validera resultatet. tex:

- milstolpe 1 Huvud loop färdig. Grundläggande funktionalitet klar. inte uppkopplad mot CAN-buss än.Existing work
- milstolpe 2 Individeuell test av CAN-protokoll
- milstople 3 koppla ihop resultat från milstolpe 1 & 2
- milstolpe 4 (slutgilig prototyp) alla detaljer enligt kravspecifikation är inkluderade och fungerande

Complete system model

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

Ideer:

Generell dataobjekt som lagrar alla karaktäristiker för samtliga optioner

```
struct option
{
   void* trigger
   void* parameter
   void* override

   int parameter_max
   int parameter_min
}
```

dataobjekten lagras intärnt i optionshanteraren i lämplig datastruktur. typ stack. Fördelen är att det är lätt att modifiera aspekter på objekten och det är superlätt att lägga till nya optioner (expandera).

Parameterkorrigering i realtid: - max-/min-värden korrigeras beroende på inparametrar (triggers) - booleska värden sätts beroende på inparametrar (override)

Optionshanteraren är kännslig på listade tirgger events som bildar intärnavbrott och triggern kontrolleras kontuernerligt via Can-bussen. vid intärnavbrott agerar optionshanteraren enligt hårdproggrammerad algorithm. Tex. ändra hastighetsparameter vid knapptryckning (trigger)

kontinuerlig kontroll av trigger events:

```
loop()
{
   Bellybutton = can.isBellyButtonPressed();

   if (Bellybutton)
      Gör något;

   if(...)
      ...
}
```

MCU ska ha en funktion för att ta emot ett parameternamn, parameterfält, parametervärde, steg, exempelvis `PAR_TOP_SPEED_FORK_DIR`, `MIN`, `30`, `5`

De parametrar som har ändrats finns i en lista på det externa chipet. Endast värden som har ändrats skickas för att uppdateras på MCU. Detta för att spara bandbredd.

MCU bör också ha metod för att läsa nuvarande inställningar för en parameter.

```
MCU_SETTINGS get_settings(PAR_TOP_SPEED_FORK_DIR);
```

Implementation

Evaluation (Metod för att utvärdera löpande under projektets gång)

```
Möjliga metoder:
MBD (Model based development)
```

Result

Discussion

Result

Method

Conclusions

Citations