ELM 334 - Homework #4

Ömer Emre Polat

1801022037

A. Problem 1

BSP is a package that includes functions to provide the user an easy way to write code. These support packages can include most basic things where there is a function that lights an LED to more complex things. The board support package can be specific, or it can have general functions to do certain things on the board. Inline assembly or just assembly can be used while creating a BSP which can help with more complex actions such as changing the value of a register.

B. Problem 2

Software errors can occur everywhere around us and it all varies with the size and cost of the error. While many software errors are not so costly here are some of the costliest software errors that have occurred in the world.

- **Nasa's Mars Climate Orbiter**

On its mission to Mars in 1998 the Climate Orbiter spacecraft was ultimately lost in space. Although the failure bemused engineers for some time it was revealed that a sub-contractor on the engineering team failed to make a simple conversion from English units to metric. An embarrassing lapse that sent the $125 million craft fatally close to Mars' surface after attempting to stabilize its orbit too low. Flight controllers believe the spacecraft plowed into Mars' atmosphere where the associated stresses crippled its communications, leaving it hurtling on through space in an orbit around the sun.

- **EDS Child Support System**

In 2004, EDS introduced a highly complex IT system to the U.K.'s Child Support Agency (CSA). At the exact same time, the Department for Work and Pensions (DWP) decided to restructure the entire agency. The two pieces of software were completely incompatible, and irreversible errors were introduced as a result. The system somehow managed to overpay 1.9 million people, underpay another 700,000, had US$7 billion in uncollected child support payments, a backlog of 239,000 cases, 36,000 new cases "stuck" in the system, and has cost the UK taxpayers over US$1 billion to date.

- **Bitcoin Hack Mt. Gox**

Launched in 2010, Japanese bitcoin exchange, Mt. Gox, was the largest in the world. After being hacked in June 2011, Mt. Gox stated that they'd lost over 850,000 bitcoins (worth around half a billion US dollars at the time of writing).

Although around 200,000 of the bitcoins were recovered, Mark Karpeles admits "We had weaknesses in our system, and our bitcoins vanished."

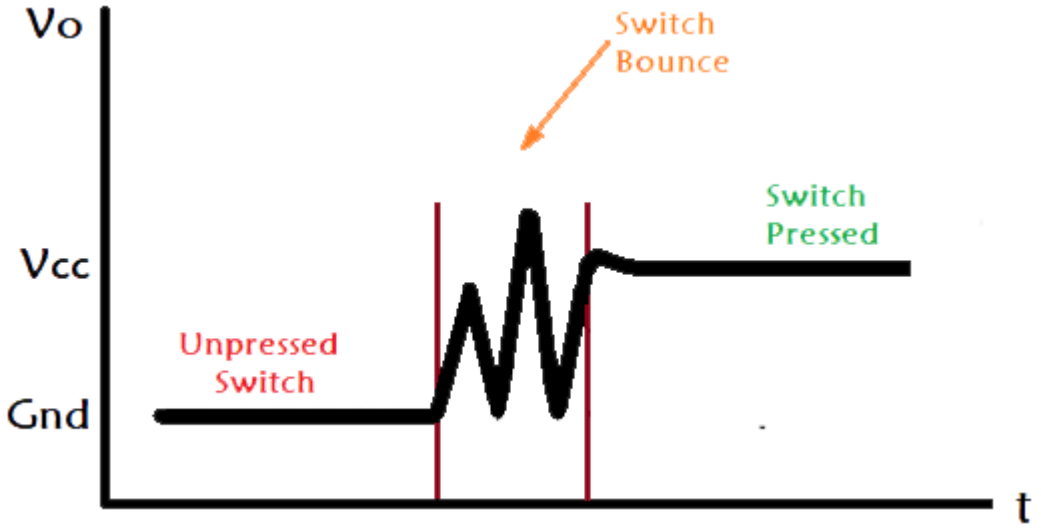
- **Patriot Missile Error**

Sometimes, the cost of a software glitch can't be measured in dollars. In February of 1991, a

U.S. Patriot missile defense system in Saudi Arabia, failed to detect an attack on an Army barracks. A government report found that a software problem led to an inaccurate tracking calculation that became worse the longer the system operated. On the day of the incident, the system had been operating for more than 100 hours, and the inaccuracy was serious enough to cause the system to look in the wrong place for the incoming missile. The attack killed 28 American soldiers. Prior to the incident, Army officials had fixed the software to improve the Patriot systems accuracy. That modified software reached the base the day after the attack.

C. Problem 3

Button bouncing phenomena can happen when the fluctuation on the button creates one or more multiple false triggers.



To fix this we can use debouncing circuits or we can add a capacitor in parallel to the ground to pull down the parasitic signals.

As for software methods of fixing this we can create an input delay that will ignore the other inputs after an input is triggered. This ignore can be slightly longer than the average bouncing period so that we can take inputs immediately after the input. But this solution would limit the speed of the input presses since there is going to be an ignore period for each of them.

We could create a more complex solution which can include using the total on time of the pulses and ignore the small bounce pulses and acknowledge inputs that actually come while the bounce is already happening.

D. Problem 5

Trends in Embedded Software Engineering

Embedded software development has long been a simple task of implementing a functionality or a control algorithm. Nowadays with the increase in complexity and demands the industry is shifting towards a model-driven development methods. In this summary we will talk about the characteristics of software engineering and challenges to ensure the higher quality of products.

In modern IT development standardized platforms are used to provide basic infrastructure for service orientation, distributed systems and error recovery with reliability. Embedded systems are rarely a standalone product. Rather it’s a single piece of the bigger piece that is the product. And

because it is used in many products it must be mass produced with tight cost restrictions. To meet these requirements software developers often create a model-driven product specific design philosophy. This need of specialized hardware and software platforms are the main difference between IT systems and embedded systems. For example, we could the Automotive Open System Architecture development platform is a standardized development platform based on embedded systems in automotive industry. This platform illustrates the difficulties in establishing a common platform for even one domain in embedded system development. With all this in mind embedded systems developers require extensive knowledge on the subject. For example, developing a vehicle stability control system is impossible if you don't understand the physics of vehicle dynamics. Consequently, embedded software development has been restricted mainly to control engineers and mechanical engineers, who have the necessary expertise on the said subject. With the ever-increasing complexity of software systems many countries have ran into software quality problems. Supporting cooperation between experts and software developers. Model driven development is one of the approaches to this increasing complexity problem. In this approach developers model software systems using easy to understand, more expressive, graphical expressions, which provide a higher level of abstraction than native programming languages. This model driven architecture creates an abstraction like how the higher-level languages replaced assembly. Model driven architecture is the primary developments leading to the increase in model driven development in embedded systems. For further simplifying development developers can even use graphical languages such as UML or even system modeling language SysML. Transforming a functional design to an architecture is a mostly manual process. This process can be done using UML and its support of different diagrams and ease of extendibility. UML supports data-flow oriented models to define the structure. These platform independent models are sufficient for generating executable code. Some studies have already successfully applied domain specific development methods to prototypes. Model driven development is based on general purpose languages and their respective code generations. Domain specific models help developers use domain specified concepts and provides them with tools needed for optimizations into code generators. In certain application domains, domain specific development is a more efficient approach. But in general, even if these special cases exist in many fields model driven development is more efficient than domain specific development. In fields where domain specific development is more efficient a more hybrid approach can be used to maximize efficiency and effectiveness. For example, UML lets developers choose specific domains to enrich its semantics so that it can be used more effectively. Although formal verification and testing techniques can fit safety and critical software requirements, many developers use dynamic tests to have a more general and defined test cases. In the model driven development context, using iterative processes, developers can apply tests based on the design models. Although tests can't prove that the code works properly, combining statistics and dynamic testing procedures we can allow quantification of residual risks. There are other techniques which can provide a general safety and reliability control. These techniques include reliability block diagrams, fault tree analysis and Markov models. fault tree analysis depicts causal chains leading to a tree consisting of failures. Safety engineers can use these described models for constructively guide the developers to develop safer products.

Up to now researchers and engineers considered reliability and safety as a separate part from model driven engineering. Even emerging standards such as ISO/CD 26262 for the automotive domain insufficiently address model-driven engineering of embedded software. The real challenge here is to develop model driven developments with ensured safety. There are many techniques used in industries currently. But with the rapidly progressing new techniques, these technologies that come with the techniques will be essential in the future of software development.

E. Problem 6

Test Driven Development in Embedded Software

Test driven development provides benefits to the developer and the development team with improved predictability. Test driven development has a high repeatability, this is a critical property because the software will continue to evolve throughout the product development cycle and potentially in future developing products. Automating test is crucial for saving both time and revealing possible existing unwanted behavior in the software. Code can be rolled back to the previous versions to fix the cause of the problem. This can greatly reduce the debugging time by reducing the

development over bugged code.

Test driven development can be applied as unit tests and as acceptance tests. Unit test exist to provide feedback for the developers about what to fix or what to implement about the software. Most embedded development is done on C or C++ languages that's why there's no single struct defining a unit. For software written in C, there must be extra effort to make it as modular as possible. But there can be modules to access function prototypes via interfaces. These modules provide regression tests and relieve the programmer from the burden that is having to repeatedly test the same code. In short acceptance operate on modules that are bound together. Various tests and scenarios are loaded into the system and checking the system behavior.

People often react to test driven development in a skeptical way. This skepticism is caused by people who say: code can't be tested without the hardware, programming in C is hard, a full build takes around 6h etc. Clearly for something simple test-driven development can be used and deployed. But embedded development is different, and it has challenging needs. Fortunately, some of these challenges can be solved with test driven development.

Concurrent hardware/software development is a reality for many embedded projects. If software can only be run on the target, developers must wait for late development and target. The embedded driven development cycle is an extension around the core of the test-driven development cycle. Cycle consists of 5 stages and each stage has its own traits. Stage 5 is less frequent in the target while stage 1 is very frequent in the development system. These stages consist of writing a test, compiling for target, running tests in the evaluation board, running tests in the target and manual target tests. At different points of the project cycle some stages described can be impossible or not as critical to perform. For example, if there is no hardware early in the project stage 4 and 5 are not practical or useful. The ideal situation is to run all the automated test on the actual hardware unit. But often this cant be the case since the hardware constrained the memory on the software development. Embedded test-driven development cycle helps to remove these problems associated with lack of hardware unit. Build and test times must be short so that the development and testing can be easier to create.

The use of interfaces and isolation allows developers to test significant system behavior without access to the real target hardware. Automated tests are valuable to both hardware and software developers because they perform test independent of the hardware subsystem.

Often engineers assume that something is hardware dependent when it is not. The tests can still be done on the memory and still work like how it would work on the actual hardware. Still embedded systems often have severe memory constraints. Unit tests require space in memory to run while there isn't much space to begin with. An evaluation board with enough memory can be used instead of the target board for better early stage testing.

Test driven development is an important step in the software development practice that can help embedded developers deliver a better product to the market. The embedded test-driven development cycle can help with the hardware accessibility issues enabling steady progress with software development. A commitment to test driven development will result in a lower coupling and higher cohesion withing the development team resulting in a more efficient development for software development products." The biggest schedule killers are unknowns, only testing and running code and hardware will reveal the existence of these unknowns".

Refferences:

[1]. https://raygun.com/blog/costly-software-errors-history/