ELEC 334 - Homework #4

Berat KIZILARMUT - 171024086

A. Problem 1 – Board Support Package

Board support package is a software that acts guideline that has desired information about an embedded system. BSP's include hardware specific information, about the inner protocols, modules and routines of the microcontroller and can include the drivers for the said microcontroller. BSP's that may or may not be provided by the manufacturer might include basic tests to assure the board is working as intended. BSP's can be customised as desired.

B. Problem 2 – Software Faults in the wild

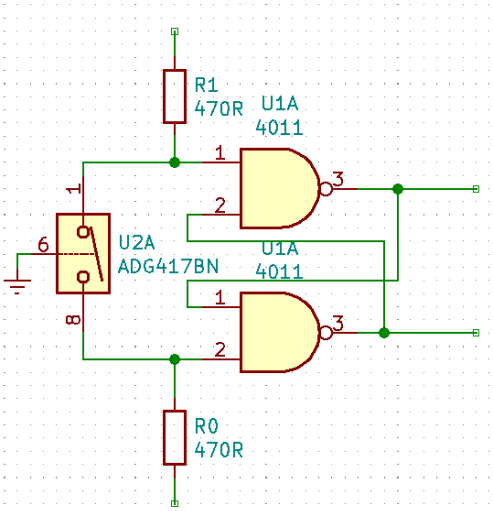
Nest thermostat freeze, a software update for the Nest 'smart' thermostat (owned by Google) went wrong and literally left their users in the cold. When the software update went wrong, it forced the device's batteries to drain out, which led to drop in the temperature. Consequently, the customers were unable to heat their homes or use any amenities. Nest claimed that the fault was due to a December 4.0 firmware update, with related issues such as old air filters or incompatible boilers. Later it released a 4.0.1 software update that solved the issue for 99.5% of customers who were affected.

HSBC Payments glitch, In August 2015, HSBC failed to process about 275,000 individual payments that left many people without pay before a long Bank Holiday weekend. This occurred due to a major failure with the bank's electronic payment system for its business banking users, affecting the individual payments. Bacs, a payment system used for payment processes across the UK, later picked up on this issue, labelling it as an 'isolated issue'.

Bloomberg cancels debt issue, In April 2016, Bloomberg's London office faced a software glitch, where its trading terminals went down for two hours. This came up at an unfortunate time when UK's Debt Management Office (DMO) was about to auction a series of short-term Treasury bills. Later in a statement Bloomberg declared that the services were restored and the glitch was a result of both hardware and software failures in the network, resulting in excessive network traffic.

Airbus software bug alert, In May 2015, Airbus issued an alert for urgently checking its A400M aircraft when a report detected a software bug that had caused a fatal crash earlier in Spain. Prior to this alert, a test flight in Seville has caused the death of four air force crew members and two were left injured.

C. Problem 3 - Bouncing



Bouncing phenomena happens because of the physical attributes of a switch. In actual world a button press is not perfectly instant. When a button is pressed, internal connecting mechanisms can and probably will literally bounce and break/reconnect the connection. Although these bounces only happen for a few milli or even micro seconds, since the microprocessors are extremely fast, these events can be registered by the microcontroller and can be registered as multiple button presses. However, preventing bouncing is possible, both with hardware and software. Starting from the hardware side, most basic method to overcome bouncing is to use a SR Flip Flop Debouncer circuit or a RC Debouncer circuit. I chose a SR FF Debouncer circuit. On the software side it's much easier to deal with by utilizing interrupt routines, if we count those as software side. A interrupt routine set up on push button connected to PA0 utilizing EXTI0_1 interrupt handler by overriding it with desired functions. If we do not count these as software methods, we can implement button ignoring loops within the code or we can insert delays after a button press checking routine. Hardware methods have the disadvantage of added cost. Interrupts do not have many downsides. Delaying or ignoring the button may or may not work in rare cases and is not the most reliable method.

```
    void EXTI0_1_IRQHandler (void) { /* Button interrupt */
        /* Desired Functions Here */
    EXTI->RPR1 |= (1U << 0); /* Clear pending status */ }
    . /* Abstraction */
    /* Setting up the interrupt operation for PA0 */
    EXTI->RTSR1 |= (1U); /* Setting the trigger as rising edge */
    EXTI->EXTICR[0] &= ~(1U << 8*0); /* EXTICR 0 for 0_1 */
    EXTI->IMR1 |= (1U << 0); /* Interrupt mask register */
    EXTI-
>RPR1 |= (1U << 0); /* Rising edge pending register, Clearing pending PA0 */
    NVIC_SetPriority(EXTI0_1_IRQn, 0); /* Setting highest priority */
    NVIC_EnableIRQ(EXTI0_1_IRQn); /* Enabling the interrupt function */
    . /* Abstraction */
```

D. Problem 4 – State Machines

E. Problem 5 – Reading ("Trends in Embedded Software Engineering")

There has been a rapid increase of innovation in embedded systems because of software technologies. These innovations make better and cheaper embedded systems possible but as the demand for increased capabilities in embedded systems increase, it puts a lot of importance on the shoulders of Software Engineers.

Standard order of business in today's modern IT is abstraction of several layers, standardizing operations etc. do not work on embedded system development. Diversity in the embedded market makes it impossible to have a standardized platform across the industry. In the embedded system industry, margins are very tight in computational resources and cost limits. Embedded systems are expected to cost the least for given constraints of a product. These requirements create specific hardware development platforms, safety protocols and requirements for a specific industry, and a specific software environment that comes with it.

MDD, Model-Driven Engineering, is a newly emerging approach to embedded software development. This approach engineers design a system not with coding, instead with expressive graphical notations. Requirement-engineering tools make it possible to design a functional system by specific requirements. Developers use the requirements and requested functions to develop the software. Developing a platform independent model requires extensive refining work but iteratively these problems and be defeated. Looking at the future, taking a one step further from MDD, we arrive at domain-specific development. This approach utilizes domain specific concepts. In this method development expertise required to create a system is pushed to the libraries and generators. This approach requires intensive initial effort but pays off in the long run. This solution is not expected to replace high level coding or MDD but is expected to be another tool in the toolbox.

In a safety-related system, trust in the system's functionality is the upmost priority. To ensure the safety of a system, developers apply verifications to the software before the code generation. There are different methods to test the safety protocols of a system, dynamic testing being a widespread one at that. No safety test system can assure complete security, a safety analysis is always the best way to ensure security, especially for safety-related systems. Two of safety measure approaches, model-based and measurement-based safety measures have each their own advantages and disadvantages. Model-based safety analysis excels at giving information at the early stages compared to other approaches because system is analysed before initialization but requires more effort. Measurement-based safety analysis on the other hand, is initialized after the system is set-up but because this test depends on measurements, statistical approach is required.

In the real-world dynamic testing method is used more often than not. Dynamic testing utilizes executing subject specific cases to test. This method is easier to implement on complex systems. This testing method can be utilized as soon as first code generation is completed. These tests might not prove safety standards, but they can increase the quality of the software. Developers can test the software in the actual working environment with this method and detect possible real-world failures.

There are no safety and quality standards in embedded systems, so safety analysis is the upmost importance in this sector. A failure works like a tree, from roots to its leaves, every failure affects the structure in line below its hierarchy. This abstraction works within embedded systems too, every part of a tree represents a module within a system. Safety of a system should be tested through its every development phase to provide most amount of security. Safety and reliability should be taken as pair of legs of a desk, these depend on each other to stand and be dependable.

F. Problem 6 – Reading ("Applying test driven development to embedded software")

Test Driven Development, as you might guess from the name, is a development philosophy that utilizes extensive use of tests to develop a working system. This philosophy does not utilize highly detailed models, instead iterates more simple models and responsibilities many times. TDD provides much appreciated predictability and repeatability to the development process. With continuous tests and feedback, the system evolves step by step in this development process. New failures caused by new codes can be detected immediately and error can be reversed without causing further trouble. These continuous test can even be automated to save time end effort and let the development teams pool their resources where it is more required. There might be many doubts about the TDD process, however article jokes about these doubts and assure confidence about TDD process by sharing their own experiences with TDD process in embedded system development. It is stated that TDD allowed them to parallel the work and reduced the debugging times to an astronomically low amount, just a week over a six month period.

UnitTest is a test method that gives constant feedback about the newly written codes abilities, whether it does or does not accomplish what it aims to do. UnitTests aim to test and fully try out the modules within a code, for example a function in the header file of a C project, rather than trying out the whole project at once, UnitTests test individual modules.

Constant development process is a harsh reality for a embedded system. There may be many problems along the path, for example not having access to target to complete tests or system and target compiler problems etc. First order of business in a TDD process is to write a test, make the code pass that test and refactor it. Secondly, compatibility of the compiler and linker by compiling on the target. Third and fourth is mostly automated tasks on evaluation and target hardware. At last, complete manual tests are completed to ensure no problems. Frequency of these cycles happening decreases from first to last. This process ensures a time efficient development process. Priority and frequency of these cycles are not set in stone through the whole project development stages, as the project goes on these can change depending on the situation and should be tailored perfectly to the current circumstances. In a perfect situation we would want to run every test on the target hardware, however this is seldom true. TDD aims to overcome hardware scarcity by implementing automated tests that both run on dev system and hardware system. Utilizing tests through the whole project ensures no complete surprises happen towards the end of a project.

TDD requires a different approach to coding. Software should be tailored to describe the inner process' of an embedded system to utilize this development philosophy. Abstraction levels should be implemented ease the process. Unified Modeling Language works with isolated and abstracted interfaces within the modules of the system. Implementing abstraction levels make the software more manageable and modifiable. For example, talking about a home security system, a homeguard function is getting tested, mock alarms and artificial inputs are sent to the module to test its inner workings and develop the module further without requiring extensive test through the whole embedded system. Utilizing these isolated interfaces within the system allows us to test behaviour of inner modules without requiring the target hardware. Automated internal tests are set to randomly test the behaviour of the system with artificial inputs and parameters. However not all test can be done within internal tests, some tasks require actual inputs and outputs, these test mostly occur less often than the artificial automated tests. Article states that sometimes embedded engineers think the development process requires the hardware being present too much and software tools are more than enough to further develop the project. Just a simple led connection process do not actually require the led to be connected on the evaluation board, you can just set the led and the functions and control the memory for the desired outputs. Tests within a limited memory environment has to be planned accordingly to get around the limitations of the testing or target hardware. Tests have to be set up to work as quick as possible and be tested with small batches.

G. References

1. Top Software failures in recent history, cigniti, https://www.cigniti.com/blog/37-software-failures-inadequate-software-testing/
2. Trends in Embedded Software Engineering, Peter Liggesmeyer & Mario Trapp, IEEE
3. Applying Test Driven Development to Embedded Software, James Grenning, IEEE