GALWAY - MAYO
INSTITUTE OF TECHNOLOGY

FINAL YEAR PROJECT

Easysleep

Zdenek Krousky

supervised by
Paul LENNON

April 12, 2020

Declaration

This project is presented in partial fulfilment of the requirements for the Degree of Bachelor of Engineering (Hons.) in Software and Electronic Engineering at Galway-Mayo Institute of Technology. This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

Acknowledgement

I would like to extend my thanks to my supervisor Paul Lennon who made sure I stayed on track with my project, Niall O'Keeffe for his support in embedded part of the project, Michael Murray for his advice on the Android application development and to Michael Keaveney for going the extra mile to support me on the hardware side of things during the pandemic crisis. I would also like to thank my wife Caroline for her ongoing support through my studies.

Table of Content

1 Project background and motivation

Aim of the project, Why?
The aim of this project was to create a device that would help to resolve nocturnal enurism, also known as bedwetting, which is common in children above the age of 5. I am the parent of a child with this issue and can relate to the stress this can cause to the child and the parents/guardians. My wife and I have tried various methods of resolving this - such as encouriging our daughter to go the toilet prior to going to the bed, restricting fluid intake before bed-time and waking her up during the night, yet none of these worked for us.

According to my research there are a number of different causes for bedwetting and medical reasons account only for 3% of these. 15% of children above the age of 5 still wet the bed at night and up to 5% above the age of 10 continue to do so. The selection of "night-time training pants" available in supermarkets go up to the age of 14, indicating that this issue can persist into the early teenage years in some cases.

Having observed my daughter for several years, I have come to the conclusion that the main cause of her betwetting is deep sleep, i.e. her urge to go to the toilet simply isn't strong enough to wake her up at night. In addition, she is also very sensitive to loud noises and therefore my wife and I felt that introducing a bed-wetting alarm sheet might cause her unneccessary anxiety.

This is where my project comes in - it aims to resolve nocturnal enurism in children where deep sleep is the root cause in a somewhat gentler and smarter way than a bed-wetting alarm.

2 Overview

What is Easysleep?, Research, Architecture diagram

The project consists of two devices that are able to communicate via Bluetooth and a mobile phone application. These are 2 FRDM-K64F microcontrollers. One assumes the role of a master and monitors the moisture detection sensor, records the time and date of the event, sends a notification to the secondary device to wake the sleeping person up and if requested, transfers time and date of last 10 events to the mobile phone application. It also notifies the parent/guardian's phone if an event happens and activates the buzzer on the sleeping person's bracelet.

The secondary device (slave) uses a Parallax vibration motor to wake the sleeping person up. If this is not acknowledged by pushing a button (indicating that he/she is awake) a second stage is entered and a buzzer is sounded.

The mobile phone application allows the user to silence an ongoing alarm remotely, request event data from the master device and save those into an SQLite database and check/change the system time and date.

3 Hardware

Hardware used, Connections, Specifications

My project consists of the following hardware:

- 2x FRDM-K64F development board used as the brains of the Easysleep master device and Easysleep bracelet device
- 3x HC-05 Bluetooth module used for communication between master and slave devices as well as between the master and the mobile phone
- Parallax vibration motor with interfacing circuit used in first stage to wake the sleeping person up by vibrations
- Buzzer used in third stage as the final, most disruptive method of waking the sleeping person up
- Relay used in second stage, to bring a light up in the room of the sleeping person

3.1 FRDM-K64F

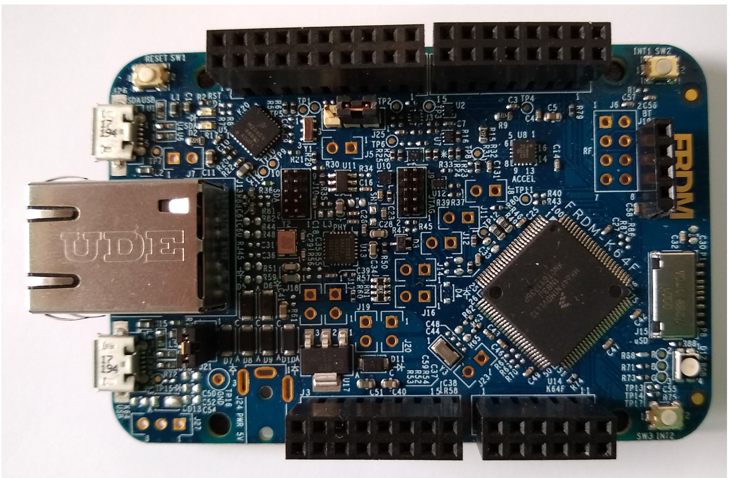

Figure 1: FRDM-K64F development board

FRDM-K64F is a very capable development board manufactured by NXP Semiconductors with headquarters in Europe in Eindhoven, Netherlands and in North America in Austin, Texas. I chose this board due of my familiarity with it and of its abilities. This board and its cousin KL25Z were used throughout the course as part of the embedded systems modules.

Board specifications:

- 120MHz ARM Cortex-M4 microcontroller
- 1MB Flash memory
- 256kB RAM
- Ethernet
- SDHC
- low-power
- FXOS8700CQ accelerometer and magnetometer
- Add-on Bluetooth module: JY-MCU BT board V1.05
- RGB LED
- 2x user push buttons
- form-factor compatible with Arduino Uno Rev.3 pin layout

3.2 Parallax 28821 Vibration motor

The vibration motor is used as a peripheral of the secondary device (bracelet) to gently and quietly wake the sleeping person up in the first stage of the overall wake-up process. The Parallax vibration motor seemed appropriate device for this task as it requires only 3V of power. However, the current it requires is quite high and cannot be supplied by the K64F thus an external supply has to be used.

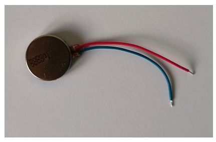

Figure 2: Parallax vibration motor

Motor specifications:

- Rate voltage: 3.0V
- Rate current: 150mA
- Rate speed: 9,000r/min Min
- Starting voltage: 2.3V

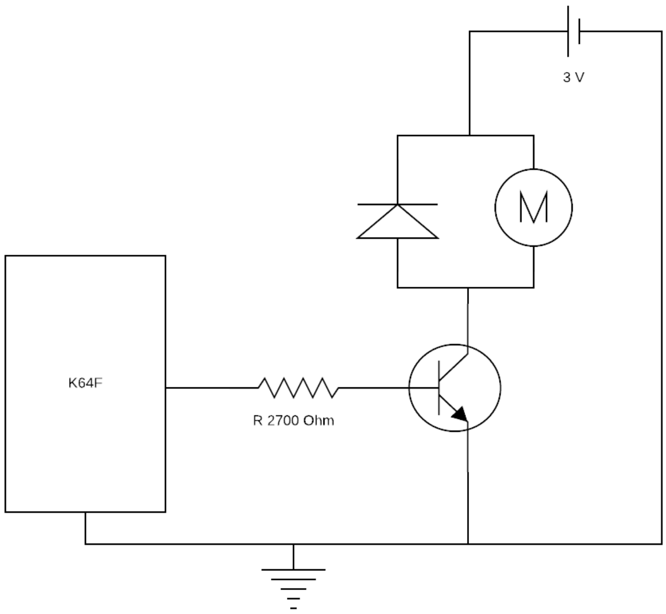

Figure 3: Vibration motor connection diagram

3.3 Relay

The relay is used in the second stage of the wake-up process. It is connected to a desklamp and is triggered when the first stage fails and more disruptive method is needed.

3.4 Buzzer

The buzzer is used for the third stage of the wake-up process if the vibration motor or relay fail to wake the sleeping person. This is a passive device and for the sound to be produced the Flex Timer Module of K64F is used to generate a PWM signal on the connected GPIO pin.

3.5 HC-05 Bluetooth module

Figure 4: HC-05

HC-05 is a commonly used Bluetooth module often used with Arduino projects. The K64F development board supports this module also. I have installed a small section of header pins to house this module. For my project I am using two of these. One to communicate with the secondary device (bracelet) and the second one to communicate with the mobile phone.

This module can operate as a master or as a slave. A second module needed to be added as it only supports one-to-one communication.

3.5.1 Bluetooth module configuration

In order for the Bluetooth modules to communicate, both the master and the slave have to be configured in the same way. Additionally they also have to share the same password and have to be made aware of each other by whitelisting each others address through AT commands.

To setup this configuration I used an FTDI cable and start the individual modules in AT mode. This is done by applying power to KEY pin prior to the VCC pin. To communicate with the Bluetooth modules and configure it in the way I needed, I have used **Minicom** - a Linux command line utility. I started this by issuing the following command upon starting the bluetooth module in AT mode.

minicom -b 38400 /dev/ttyUSB0

The following commands were used to setup the master device:

- AT+UART=38400,0,0 (Baudrate, 1 stop-bit, no parity, default 8-bit mode)
- AT+ROLE=1 (master)
- AT+INQM=1,1,48
- AT+PSWD=7777

Similar setup was used for the second device with the exception of the ROLE, where it needs to act as a slave device therefore the ROLE was defined as 0.

4 Software

Software used, Programming languages, IDEs, Software tools

Througout my project I used various software tools to develop C and Java code, as well as software to monitor the behaviour of the system. All the code for the embedded devices was developed on a Linux system, Android Studio was however too resource heavy for my laptop so I borrowed a lab computer with the permission of Michael Keaveney, the GMIT technician.

4.1 MCUXpresso

I used MCUXpresso 11.1 to develop code for K64F. This is an Eclipse based IDE tailored to suit NXP devices. There are lots of alternatives from other manufacturers such as Atollic Studio for STM32. I became familiar with this IDE during my time in GMIT as it was used for code development in embedded systems classes. I also used this IDE during my work placement in Jaguar LandRover in 2019.

The project uses Amazon FreeRTOS to efficiently manage various tasks of the system. This is an open-source real-time operating system and its usage greatly reduces the complexity of C code used for functionality of the system. A programmer can divide the code into smaller, easier to manage blocks known as tasks. A communication between those is facilitated through the usage of semaphores, notifications and queues.

Prior to the commencement of work on the project I had to obtain the appropriate Software Development Kit. I did so, through the SDK Builder present on NXP website. This site allowed me to select specific processor and middleware and generated an SDK package that I then imported into the MCUXpresso. This package included all necessary drivers for various peripherals present on the development board such as GPIO or FlexTimer driver.

I also created a repository on Github to have a proven record of my work as required by my supervisor and other lecturers (and also to have a safety net in case things went wrong). This is an extremely useful tool to know and use. I was grateful to have the ability to revert to previous revisions of my code throughout the development as on a small number of occassions, the path I chose to steer the development proved unsuitable, and it would have been impossible to revert the changes from memory.

4.2 Android Studio

This integrated development environment was the best choice for the development of my mobile application. There are other options out there such as Eclipse IDE but the Android Studio is the most supported one and was actually used during my last semester in Mobile application development module - so I was already familiar with it.

This IDE is developed by BrainJet, the studio behind the well known IntelliJ and one can really see the similarities of both environments. This is an advanced IDE with plenty of features fully supported by Google Inc.

The idea behind this application was to allow the user to wirelessly control the Easysleep module. This communication allows the user to check Easysleep's time and date or configure those as well as request data of recent incidents and silence an ongoing alarm.

During my project development I used a third party application that allowed me to send characters via Bluetooth to Easysleep and receive data back. I wanted to recreate this application in a way that would better suit the project.

4.3 Other

Additional software tools used

4.3.1 SystemView

SystemView is a great tool developed by SEGGER for monitoring the behaviour of Real-time operating systems as well as interrupts. It is free to use. It requires J-Link to be either installed as a bootloader on the development board or a harware J-Link probe needs to be used. In my case I had access to the hardware version. In order for it to work fully, I had to cut two traces on the development board otherwise it would clash with the OpenSDA debugger and the code would not get uploaded to the memory.

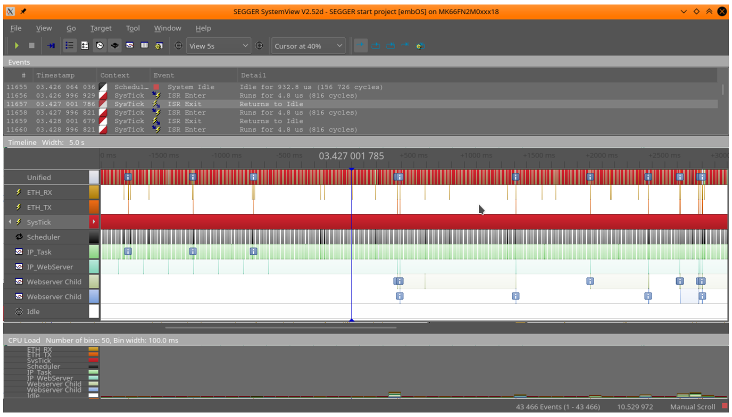

Figure 5: SystemView

4.3.2 FreeRTOS

FreeRTOS is one of many real-time operating systems available. Other options include QNX, ThreadX, embOS or Zephyr. Using a real-time operating system allows the programmer to divide the functionality of the project into individual blocks that are easier to manage, can communicate amongst themselves and maintain responsiveness. Real-time operating systems are often used in places where the functionality of a system is somewhat complex and a certain level of responsiveness is required (time-deterministic).

4.3.3 Pulseview

Pulseview is a Qt based logic analyzer. It is licensed under GNU GPLv.3. This software allows the user to monitor the output of GPIO pins as well as decoding of various protocols, such as SPI or UART. This package is in the repositories of my Linux distribution so installing it was as simple as typing a **sudo apt install pulseview** into the command line and let the system to do the rest. This tool proved to be invaluable when I was working with the buzzer and FlexTimer Module as it allowed me to monitor the output square wave.

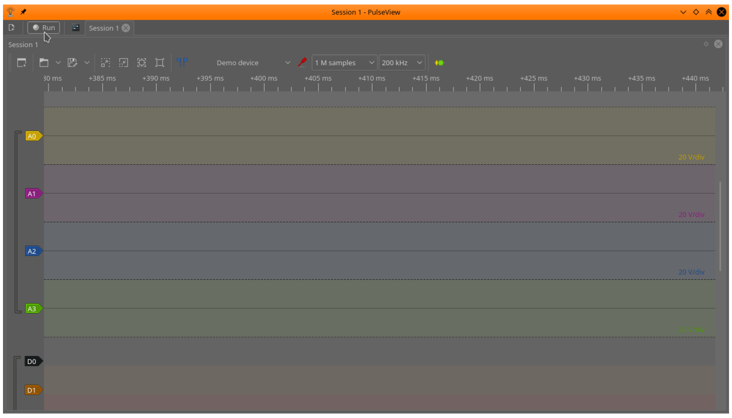

Figure 6: Pulseview

4.3.4 Git/GitHub, gitk

Throughout the development of the C code for the embedded platform, as well as of the Android application and this report, I used a Version Control System known as Git and it's online counterpart GitHub. This allowed me to safely store my code, revert to previous code revisions and work from different computers at school or at home.

4.3.5 Doxygen

Doxygen is a documentation generator widely used in software development companies. It allows user to generate documents describing code functionality by adhering to certain code commenting standards.

Doxygen is available for various programming languages. It comes with a large configuration file that the user can adjust to suit the need of the project.

4.3.6 Project management software

project management software, BT configuration,

5 Code Development

Languages used, code development and examples

The programming languages that were used for the development for the code were C programming language and Java. C was used to program the two K64F embedded devices while Java was used for the Android application development.

5.1 K64F Master code development

K64F Master Device code repository
I started developing the code for this device in early October 2019 and in parallell ordered few items via the technician in GMIT, i was able to proceed without having the ordered items to hand. It was my intention to use FreeRTOS from the very beginning. This component was on the curriculum in the first semester of the final year at GMIT, however, I had already completed an online course in FreeRTOS the previous summer, so I was already familiar with it's concepts and inner workings. The Real-time Operating Systems module covered at GMIT in the final year, helped me to strengthen my understanding of the subject further.

Through my online course, I became aware of the SystemView tool which allows the developer to monitor the behaviour of FreeRTOS tasks as well as of the interrupts. To use the software to it's full potential I had to order a SEGGER J-Link debugging probe so I could use the software's continuous recording capabilities and watch how the system behaves and reacts over time and to different interactions. Later on, I discovered that similar functionality can be achieved by upgrading the on-board debugger to J-Link version (in this case, the original on-board debugger of K64F is OpenSDA). If I didn't use the J-Link, I would be limited to recording of few seconds only, until the allocated SystemView buffer filled up. This would be problematic, because I would not be able to test everything this quickly. As a result, this exposed me to yet another tool that is commonly used in embedded software development and I hope to benefit from it in the future.

In order to successfully use the J-Link Edu debugging probe, adjustments to the project had to be made first. First I had to download and extract SEGGER SystemView target sources and place them into the project folder. I had to setup the include paths in MCUXpresso so the header files and source files would be visible for the compiler. This is done via Projec -¿ Properties

-¿ C/C++ build. Below is the directory structure of the SEGGER include files.

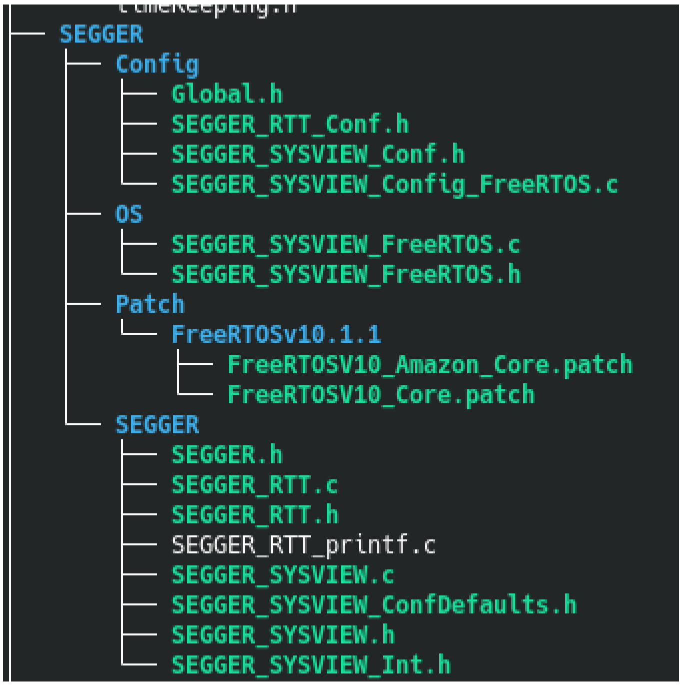

Figure 7: SEGGER directory structure

All these files had to be included for the SystemView to work correctly, but additional adjustments had to be made to the code. First, a patch provided by SEGGER had to be applied to the FreeRTOS files in order to link the FreeRTOS with SEGGER target sources. The patching capability can be found in right-click menu under Team option. This will provide a list of files that would be patched and view of the actual changes for the developer to review. To continue with the integration of the SystemView I had to make changes to a number of files manually. First changes were added to **FreeRTOSConfig.h** file.

```
// FreeRTOSConfig.h
#define INCLUDE_xTaskGetIdleTaskHandle 1
#define INCLUDE_pxTaskGetStackStart 1
    .
    .
    .
#include SEGGER_SYSVIEW_FreeRTOS.h
```

The **SEGGER_SYSVIEW_FreeRTOS.h** has to be included at the end of the FreeRTOSConfig.h or above every include of FreeRTOS.h as it defines the trace macros to create SystemView events.
The processor used has to be specified in **SEGGER_SYSVIEW_Conf.h**. There are four macros to choose from defined in this file and as I used the ARM Cortex-M4 I had to redefine the SEGGER_SYSVIEW_CORE macro to SEGGER_SYSVIEW_CORE_CM3.
I also had to define the size of the SystemView buffer in this file. By default this is set o 1024 Bytes but because the size of the RAM on K64F is 256kB I could afford to assign more to it so I have allocated 8kB of RAM. This would actually affect the amount of time that the data can be recorded by SystemView in Single-Shot mode (that is without the use of J-Link).

```
// SEGGER_SYSVIEW_Conf.h
#define SEGGER_SYSVIEW_CORE_OTHER 0
#define SEGGER_SYSVIEW_CORE_CM0  1 // Cortex-M0/M0+/M1
#define SEGGER_SYSVIEW_CORE_CM3  2 // Cortex-M3/M4/M7
#define SEGGER_SYSVIEW_CORE_RX   4 // Renesas RX

#define SEGGER_SYSVIEW_RTT_BUFFER_SIZE (1024 * 8)
```

I also defined SystemView application name and SystemView device name in **SEGGER_SYSVIEW_Config_FreeRTOS.c** as well as SystemView RAM base address which on this microcontroller is at 0x20000000.

```
// SEGGER_SYSVIEW_Config_FreeRTOS.c
#define SYSVIEW_APP_NAME      "Easysleep -
    Dev1_moisture_detection"
#define SYSVIEW_DEVICE_NAME   "MK64FN1M0XXX12"
#define SYSVIEW_RAM_BASE      (0x20000000)
```

Next step was to enable the ARM Cortex Mx Cycle Counter which is done by enabling the Current PC Sampler Cycle Count Register **DWT_CYCCNT** as it is disabled by default. This register holds the number of core cycles and this can be used to measure elapsed execution time. In SystemView, it provides the time of execution of every instruction since the start.

With the SystemView properly setup, I could begin to develop the functional code for my application. I started by creating a

5.2 K64F Slave code development

K64F Slave (Bracelet) Device code repository

The secondary device, the bracelet, also uses FreeRTOS for task management. Setting this one up was not as difficult as the first one as it has less responsibilities and I already had some experience setting up the first bluetooth module. However, it made sense to take advantage of the FlexTimer peripheral of the K64F to generate a square wave needed for the buzzer and thus eleviate this responsibility from the processor.

To setup this peripheral I have used the MCUXpresso Configuration tool again. This allowed me to specify the output frequency of the square wave thus changing the pitch of the produced sound. This is also possible to do programmatically but if there is

5.3 Android application code development

Android code repository

5.3.1 User Interface Development

GUI

5.3.2 Bluetooth Communication

BT

5.3.3 Database Development

???

The development of this application was challenging and I had to follow a number of tutorials online in order to establish the Bluetooth communication. I have decided that for the purpose of this document I will divide the development process described into three subsections:

- User Interface Development
- Bluetooth Communication
- Database Development

6 Conclusion

what was the development of the project like

7 References

8 Bibliography

List of Figures