A close up of a logo

Description automatically generated

Mazen Amr

900161021

Heart Disease Detection

report

Embedded Systems project

https://github.com/djzenma/ECG

Table of Contents

[Motivation 2](#_Toc41436110)

[Components Needed 3](#_Toc41436111)

[Architecture 4](#_Toc41436112)

[Firm Deadline 4](#_Toc41436113)

[Internal Clock 4](#_Toc41436114)

[ADC & Timer 4](#_Toc41436115)

[GPIO 4](#_Toc41436116)

[UART 4](#_Toc41436117)

[Connections 4](#_Toc41436118)

[Hardware Architecture 5](#_Toc41436119)

[Software Architecture 5](#_Toc41436120)

[ECG Preprocessing 6](#_Toc41436121)

[Machine Learning Model 7](#_Toc41436122)

[TensorFlow Lite 7](#_Toc41436123)

[Limitations 7](#_Toc41436124)

[References 9](#_Toc41436125)

# Motivation

Embedded Systems are found everywhere, and you might even be interacting with some of them without even noticing that it is one. This is due to their large field of applications such as cars (ABS systems and others), Point-Of-Sale systems, anti-focus cameras, avionic systems, etc.… and this is just to name a few.

Embedded Systems are usually cheap as they are often part of a bigger system that should be cheap. This implies that embedded systems are accessible for anyone.

Recently, embedded systems begun to support the implementation of Artificial Intelligence (we will discuss this later). Some of these microcontrollers are, of course, the STM32 family.

Artificial Intelligence (AI) is called “The New Electricity” by some scientists, such as Andrew Ng. This is because AI has found its way in every domain of our lives, starting from our alarms to medicine and diseases detection. Indeed, AI has proven its excellence in medicine and helped in many developed countries to speed up diagnosis and increase the number of possible examinations per day.

If we focused in heart diseases, AI is able to diagnose heart problems as good as a human doctor but in just a few seconds. If we let a human do this same task, it will require him about 13 minutes per case, which is obviously a huge speed up.

If we combined the power of both worlds, i.e. the diagnosis and speed power of AI and the computation power and accessibility of Embedded Systems, we will obtain our beautiful project: Heart Disease Detection in STM32 microcontroller.

In this proposal, we will discuss how to implement Heart Disease Detection using an STM32 microcontroller. We will explain first what is an ECG signal, we will then show what are the hardware and software components needed for this project, then have an overview of the system’s architecture, and finally conclude with the project plan.

# Components Needed

Hardware components:

* A Microcontroller, I will use the Nucleo STM32L432. The Nucleo has built-in ADC and UART modules, if your MCU doesn’t contain any of these modules you will have to acquire them and connect them to your MCU.

A circuit board

Description automatically generated

Fig1.

* A Heart Monitor, I will use the Spark Fun AD8232

A circuit board

Description automatically generatedA picture containing white

Description automatically generated

Fig2.

* Breadboard
* Some jumpers

Software Components:

* Tensorflow Lite Support
* An IDE, I will use Keil uVision 5
* STM32CubeMX, this is not required but it facilitates the implementation a lot

# Architecture

## Firm Deadline

Our application’s deadlines are categorized as Firm deadlines. In other words, it is okay if a couple of deadlines are missed as this will not halt the performance of our system. The deadline in our case is to successfully collect 125 samples of ECG signal and classify the disease state of the patient.

## Internal Clock

SYSCLK = 80 MHz

## ADC & Timer

I used the built-in ADC in my Nucleo along with the Timer to accomplish a sampling rate = 125KHz:

**ADC**:

* Port: PA0
* ADC1 channel IN5 Single-Ended mode
* Resolution = 12 bits
* Its clock = 1KHz
  + Original Clock = 10MHz
  + Prescaler = 9,999
  + Resultant clock = 10MHz / 9,999+1 = 1KHz

**Timer**:

* TIM15
* Its clock = 1KHz
  + Original Clock = 10MHz
  + Prescaler = 9,999
  + Resultant clock = 10MHz / 9,999+1 = 1KHz
* Auto-Reload Register = 8
* Sampling rate = 1KHz/8 = 125Hz

## GPIO

* 2 input Pins: PA9, PA10
* 1 output Pin: PB3

## UART

* Ports used: PA2 of Transmission and PA3 for Receiving
* Baud Rate = 9600 bits/s

## Connections

The Heart monitor has 9 pins, we will use only 5. The following table details what pins of the heart monitor should be connected to which pins of the MCU:

|  |  |
| --- | --- |
| Heart Monitor | Nucleo |
| GND | GND |
| 3.3V | 3.3V |
| OUTPUT | PA0 (ADC1 CH1) |
| Lo- | PA9 (GPIO Input) |
| Lo+ | PA10 (GPIO Input) |

## Hardware Architecture

A screenshot of a social media post

Description automatically generated

Fig3.

The Heart Monitor sends the ECG signal as an Analog signal through its OUTPUT pin. The ADC intercepts it and converts it to a digital signal. The ADC samples and digitizes the signal once every 8ms. This is done using a Timer that raises an interrupt every time the timer reaches the value set in the AR register, which is 8. This allows us to achieve 1000KHz/8 = 125Samples/Sec. The digital signal is then preprocessed and saved in the memory until 125 signals are formed then fed to the ML model to predict the patient’s disease status. This patient’s status is then sent through the UART to the PC Tera Term as a string specifying to which class does the patient belong. Finally, the 125 signals are flushed and this cycle repeats.

## Software Architecture

I followed the **Round Robin with Interrupts architecture**.

**In the Timer Interrupt Service Routine (ISR)**:

Every 8ms, the Timer *TIM15* issues an interrupt where the *ADC1* converts the analog signal, then the sample is preprocessed, then stored in the *ECG* array, finally, a *ready* flag is raised when the array is full (we have a full heartbeat).

**In the Main Loop:**

Once the *ready* flag is raised, the ML model is called to infer using the freshly created array. The result is then sent as a string to the PC (Tera Term) using the UART2 and the *ready* flag is reset to false.

## ECG Preprocessing

The authors of the ECG Heartbeat Classification [1] preprocessed the data as follows:

1) Splitting the continuous ECG signal to 10s windows and select a 10s window from an ECG signal.

2) Normalizing the amplitude values to the range of between zero and one.

3) Finding the set of all local maximums based on zerocrossings of the first derivative.

4) Finding the set of ECG R-peak candidates by applying a threshold of 0.9 on the normalized value of the local maximums.

5) Finding the median of R-R time intervals as the nominal heartbeat period of that window (T).

6) For each R-peak, selecting a signal part with the length equal to 1.2T.

7) Padding each selected part with zeros to make its length equal to a predefined fixed length.

A close up of a logo

Description automatically generated

*Fig. 4: An example of a 10s ECG window and an extracted beat from it.*

To apply these preprocessing steps, a 10s window is required, in other words, they collect a 10s window then extract the heartbeats from it, where every beat window is of variable period T. However, I wanted to do the preprocessing on the fly (preprocess every collected sample). Therefore, I modified the **variable T** to be a constant = **100**, which allowed me to do the processing on the fly without the need wait until a 10s window is collected. Of course, T in my case is a hyperparameter that if tuned properly will improve the performance, however in my case it is performing well.

Furthermore, every beat has a **fixed length** (step 7) which I chose to be **188** because this is the length used in the dataset [5].

## Machine Learning Model

I followed and implemented the ECG Heartbeat Classification [1] paper’s model as it tackles the same ECG problem and classifies the same 5 classes that we are trying to predict. I trained the model normally using Tensorflow Keras then migrated it to our MCU using Tensorflow Lite. However, this model raised a problem that we will discuss in the Limitations section.

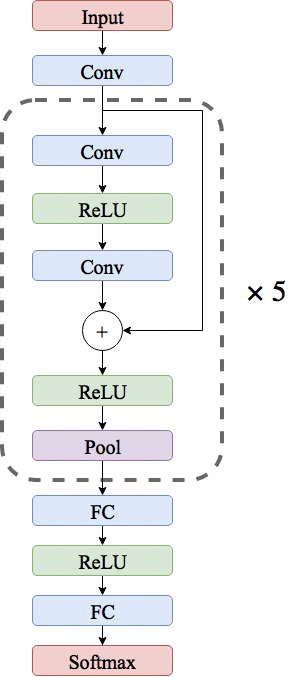


Fig.5

## TensorFlow Lite

After training the model using Keras, I converted it to a TF Lite model using TensorFlow Lite Converter that generates a FlatBuffer file (.tflite). I optimize the flat buffer using a method called Dynamic Range Quantization which reduces the size of the buffer, it can go up to 4x smaller and can have 2-3x speedup in performance while having a very close accuracy to the original model. This is done by reducing the weights to 8-bit precision. Then I included this FlatBuffer in the MCU C project and used it to infer the patient’s class.

# Limitations

My Nucleo board’s FLASH size = 256KB and RAM size = 64KB.

My ML model’s (discussed in the Machine Learning Model section) RAM size = 70KB and, whenever I tried to reduce this size by deleting a layer or a block of layers from the model or by trying other models, that never helped because the RAM size remained constant and the FLASH size was the one that was being reduced every time. In fact, the FLASH size was never a problem as it was always less than 256KB (the most reduced ML model (containing Convolutional Layers) that I got had a 50KB of FLASH only). Convolutional Layers were the layers consuming the most RAM. Therefore, because of the great limitation in the RAM size of my Nucleo, I created a very simple model composed of 4 Dense layers only and got rid of all the Convolutional and Pooling layers in my model to have a RAM size of 400 Bytes only and a FLASH of 7KB. Of course, that means that my model performs very poorly with respect to the original one discussed previously.

# References

[1] **ECG Heartbeat Classification: A Deep Transferable Representation, by** [**Mohammad Kachuee**](https://arxiv.org/search/cs?searchtype=author&query=Kachuee%2C+M)**, [Shayan Fazeli](https://arxiv.org/search/cs?searchtype=author&query=Fazeli%2C+S), [Majid Sarrafzadeh](https://arxiv.org/search/cs?searchtype=author&query=Sarrafzadeh%2C+M)**

<https://arxiv.org/abs/1805.00794>

[2] **TinyML**, by [Pete Warden](https://learning.oreilly.com/search/?query=author%3A%22Pete%20Warden%22&sort=relevance&highlight=true), [Daniel Situnayake](https://learning.oreilly.com/search/?query=author%3A%22Daniel%20Situnayake%22&sort=relevance&highlight=true), ISBN: 9781492052043.

<https://learning.oreilly.com/library/view/tinyml/9781492052036/>

[3] **Tensorflow Lite documentation**.

<https://www.tensorflow.org/lite>

[4] **STM32L432KCU Datasheet**

https://www.st.com/resource/en/datasheet/stm32l432kc.pdf

[5] **ECG Heartbeat Categorization Dataset**

https://www.kaggle.com/shayanfazeli/heartbeat