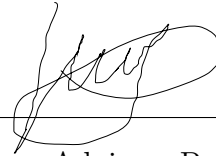


Development of a Tunable Microwave Source for EIT Magnetometer

A thesis submitted in partial fulfillment of the requirement
for the degree of Bachelor of Science in
Physics from the College of William and Mary in Virginia,

by

Jonathan Diaz-Ramos



Advisor: Prof. Irina Novikova



Prof. Todd D. Averett

Williamsburg, Virginia

May 2021

Abstract

A cardio-magnetic field produces magnetic field gradients of 10 - 100 pT . As ^{87}Rb atoms get exposed to magnetic field gradients they experience the Zeeman effect. Two ground state sub-energy levels are shifted. Controlling the frequency of a laser which matches the shifted sub-energy levels leads to a measurement of the magnetic field. By implementing an algorithm to an optic phase modulator, which controls the voltage – thereby affecting laser frequencies, can lead us to the measurement of a magnetic field. In order for the algorithm to operate it takes a given frequency, divides the frequency itself, then redefines three values to represent three integers needed to operate the optic phase modulator. The task completed adjusts the frequencies in a series of steps given a specified amount of time. The algorithm also changes the frequency of the laser as the magnetic field shifts; which leads to the measurement of every magnetic field produced.

I. INTRODUCTION

The goal of the project is to measure the magnetic field of the heart. The heart muscle has electrical activity caused by natural ion currents within its cells as it contracts and repulses; thus, creating its own electromagnetic field that changes in magnitude with every beat. Measuring magnetic field gradients will give us more details of what's happening within the heart. There are two issues with measuring the magnetic field gradients of a heart. The fields will be experiencing interferences such as the Earth – which holds a magnetic field of about 20 - 50 μT . Since the magnetic field of the heart is only 10 - 100 pT, this can be easily disturbed [1].

The solution to the problem was incorporating a gradiometer. The gradiometer will measure two magnetic field gradients in two separate locations. The further the distance, of about 1 cm, the greater the difference in magnetic fields become. Subtracting the two magnetic fields will let us know how the magnetic field of a heart changes depending on its location, noises will be cancelled out.

A cell containing ^{87}Rb atoms will sit within the magnetic field. As these atoms are exposed to the magnetic field they will experience the Zeeman Effect. This allows the atoms' sub-energy levels to shift, giving one the opportunity to measure the magnetic field as two optical fields enter the vapor cell. The two optical fields will need to be tuned to a certain frequency which follows a method known as Electromagnetic Induced Transparency (EIT) helping one measure the magnetic field.

EIT is when two optical fields experience a quantum phenomena, known as the dark state, where photons begin to have no interactions with the atomic system, they propagate through a medium making it transparent. No interactions with the atomic system means it is not absorbing as many photons leading to non-existent spontaneous emissions; this quantum phenomena is known as the dark state. To enter the dark state, the difference in frequency of the two optical fields entering the atomic system needs to match the difference between the energy sub-level of the two ground states. This changes with every new magnetic field. The frequency used in order to enter the dark state is 6.83468635 GHz.

To control a given frequency we use a voltage-controlled oscillator, VCO – the oscillator can manipulate a frequency by tuning its voltage. The following task for the year was to program a D1 Wemos Mini Pro LMX2487 chip that can stabilize the voltage-oscillator

thereby stabilizing a given frequency. We program the VCO to stabilize a light source to a frequency of 6.83468635 GHz. We compare it to a reference source of the same frequency – if it matches, we have proven that we have the correct program for the chip that can control the VCO.

The output frequency is defined by three specific values called registers R0, R1, and R5 that operate the VCO. Setting different frequencies changes the values of the registers R0, R1, and R5. The registers are defined by three numbers, two of these numbers change due to a different frequency input. The three values are called the Numerator, the Function Numerator, and the Function Denominator; the Function Denominator is held constant. Throughout the project we will be working with a frequency of 6.83468635 GHz. The Numerator and the Function Numerator will be defined by this given frequency, leading to the definition of R0, R1, and R5. This then leads us to the next task. A new program changes frequencies in a series of steps given a specified amount of time, this becomes useful when the magnetic field shifts; changing the frequency required to enter the dark state.

II. CONTROL FREQUENCY WITH LMX2487 CHIP

Fig. 1 shows the LMX2487 chip which contains the program that tunes the voltage of the Voltage-Controlled Oscillator.



Figure 1: Image for the LMX2487 chip.

The chip is incased in a 3D model printed box with lids as seen below:

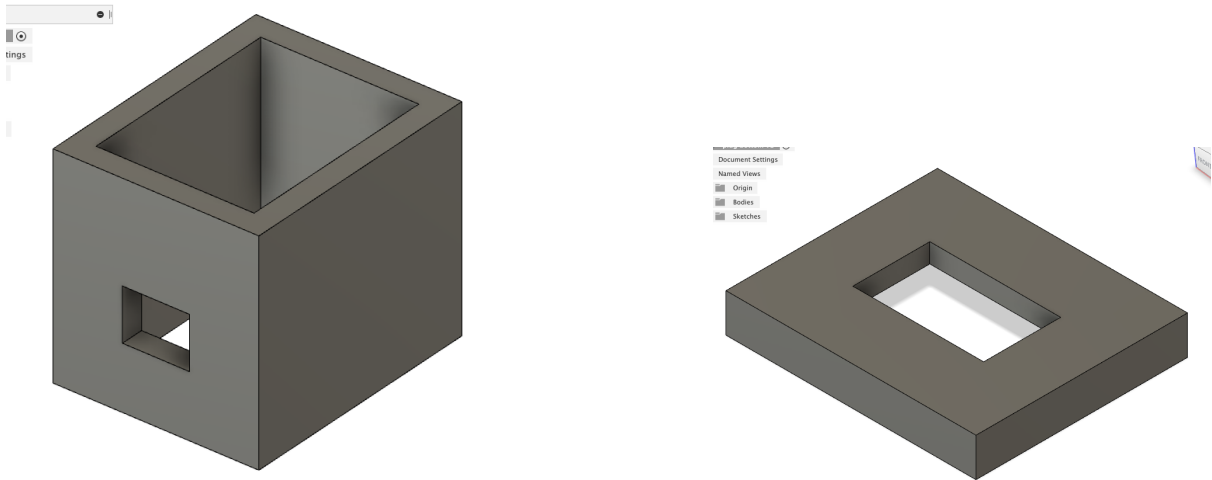


Figure 2: Image of 3D model printed box.

Placing the LMX2487 chip with the case we have:

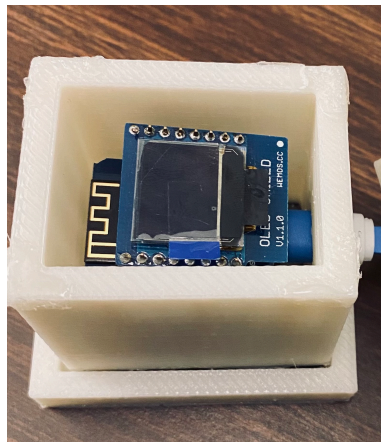


Figure 3: Image fo the LMX2487 chip with case.

A. Control Frequency

To calculate the registers one needs the binary representations of the Numerator, Function Numerator, and the Function Denominator. The Numerator and the Function Numerator come from dividing the frequency, the Function Denominator is held constant. If our frequency is 6.83468635 GHz, our integers are:

$$N = 1366 \tag{2.1}$$

$$F_N = 3749080 \quad (2.2)$$

$$F_D = 4000000 \quad (2.3)$$

N , F_N , and F_D are respectively the Numerator, the Function Numerator, and the Function Denominator.

Below we have a diagram representation of the process, from receiving the frequency, dividing the frequency itself, then redefining the three values to represent the needed registers.

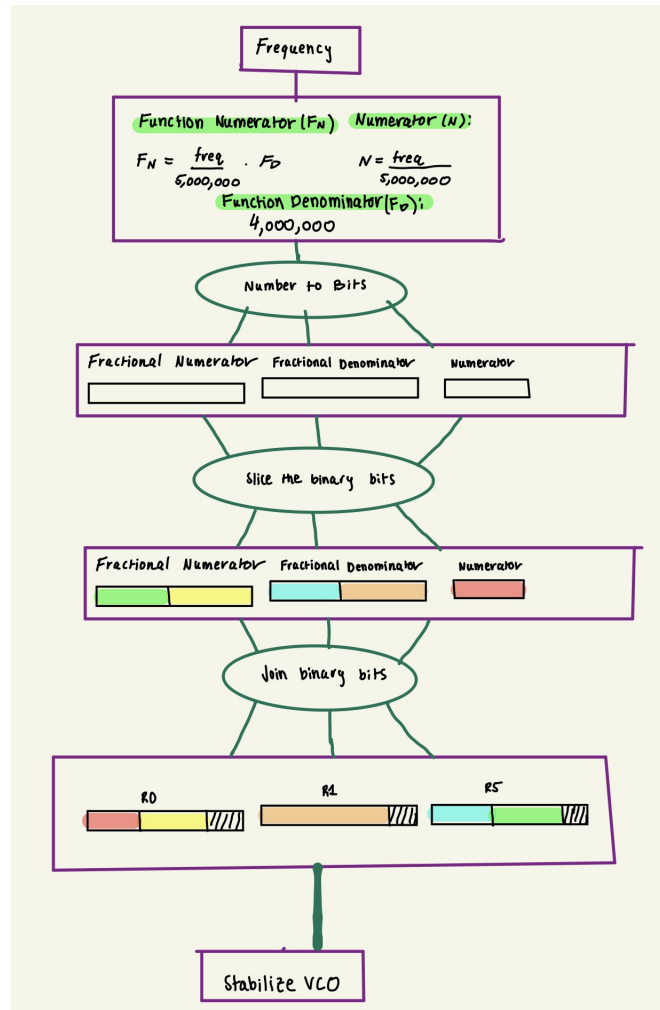


Figure 4: Flow diagram of the program used to define the needed registers.

As we receive the frequency, the following mathematical procedures are used to setup the Numerator and Function Numerator.

Given our specific Frequency, $freq$, divide the given frequency taking the integer of the value found:

$$N = \frac{freq}{5000000} \quad (2.4)$$

N is called Numerator. Our Function Denominator, F_D , is given, 4000000. By estimating the following:

$$F_N = \lceil (N - \lfloor N \rfloor) F_D \rceil \quad (2.5)$$

We find our Function Numerator F_N . Now that we have the three values, we perform three functions to produce our registers: R0, R1, and R5.

Our given register outputs should consist of the following:

$$R0 = N[10, 0] + F_N[11, 0] + [0] \quad (2.6)$$

$$R1 = F_D[11, 0] + [0, 0, 1, 1] \quad (2.7)$$

$$R5 = F_D[21, 12] + F_N[21, 12] + [1, 0, 1, 1] \quad (2.8)$$

Where one views $N[10,0]$ as the bits in the binary string N located at positions 0 to 10. Same goes for the rest of the binary strings.

The binary representations of registers R0, R1, R5 before we start the algorithm are respectively $[0]$, $[0, 1, 1, 1]$, and $[1, 0, 1, 1]$.

B. Converting Numbers to Binary

Binary representations consist of major bits and least bits. The major bit is defined as the first bit generated, this bit is the right end of a binary representation. The least bit, or the last bit, generated sits on the left end. The positions are based on the number of bits a binary representation contains. For example, we have register R1 which is initially given as:

$$R5 = [0, 1, 1, 1] \quad (2.9)$$

This representation consists of four bits. The positions are from 4 to 1; though, for our algorithm the positions are interpreted as 3 to 0. Another notation for register R5 is $[3, 0]$. The major bit sits at position 0 while the least bit is at position 3. For this project we read from right to left when looking at a binary string.

In order to produce our needed registers, one needs to convert the Numerator, Function Numerator, and the Function Denominator into binary form.

To convert this number into a binary representation we first need to know if this specific number is divisible by 2. Binary code works with 2 since a binary is represented as either a 1 or 0. The algorithm takes a number and gives the remainder of the number when it's divided by 2.

$$F_D : \frac{4000000}{2} = 2000000 \quad (2.10)$$

If a remainder exists, we insert a 1 into our current empty binary representation. If there is no remainder, we insert a 0 into our empty binary representation. By using a while loop we take the number and check if it's divisible by 2. The number is then subtracted by the binary representation given:

$$F_D : 2000000 - 0 \quad (2.11)$$

We keep repeating the process while the number is greater than zero. The binary representation for the Function Denominator is now:

$$F_D = [1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0] \quad (2.12)$$

Here we have a binary representation consisting of 22 bits; with position 21 to 0. This binary form is represented as $F_D[21, 0]$. We go through the same process for each of the numbers needed; therefore, we have the following:

$$N = [1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0] \quad (2.13)$$

$$F_N = [1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0] \quad (2.14)$$

Which are the binary representations of integers found in Eqs. (2.1 - 2.3). These binary representations are given from the frequency 6.83468635 GHz.

C. Dividing Binary Representations

Our binary representations of the registers R0, R1, R5 consist of bits produced by the Numerator, Function Numerator, and the Function Denominator. As an example, to produce

register R1 we would need:

$$R1 = F_D[11,0] + [0,0,1,1] \tag{2.15}$$

The Function Denominator, from positions 11 to 0, is added to the left of the existing register R1. To execute this we need an algorithm that takes a binary representation, identifies the following positions needed, which then leads to it being sliced from positions y to x ; where $y > x$. When a user inputs a specific position, we need to make sure position y is less than the length of a given binary representation. If one were to call position 7 in a binary representation that only consists of 4 bits, the algorithm would lead to an error. An if statement takes note of this by making sure position y is less than the length of the binary representation, when the statement holds True, we proceed to the next step.

Keep in mind the major bit of a representation sits at the right end, to keep this arrangement we slice bits right to left. Using a while loop, we start slicing from position y , while $y > x$ proceed to slice. After every slice, we subtract position y by one. Looking at an example, slicing the Function Numerator is seen in Fig.3.

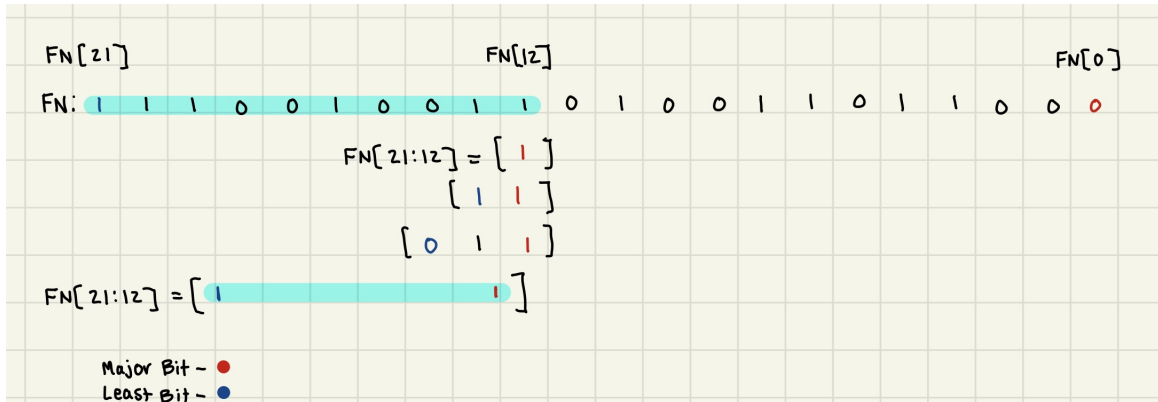


Figure 5: F_N being sliced from positions 0 to 11.

We do this for each of the registers in Eqs. (2.6 - 2.8).

D. Defining Registers

To define the registers, we need to join binary representations. After slicing the Function Denominator from positions 11 to 0, we are ready to insert it into register R1. An algorithm

would need to take two existing binary representations and join them together. One needs to be aware of the arrangements the bits are being placed. The steps of this procedure is seen in Fig. 4.

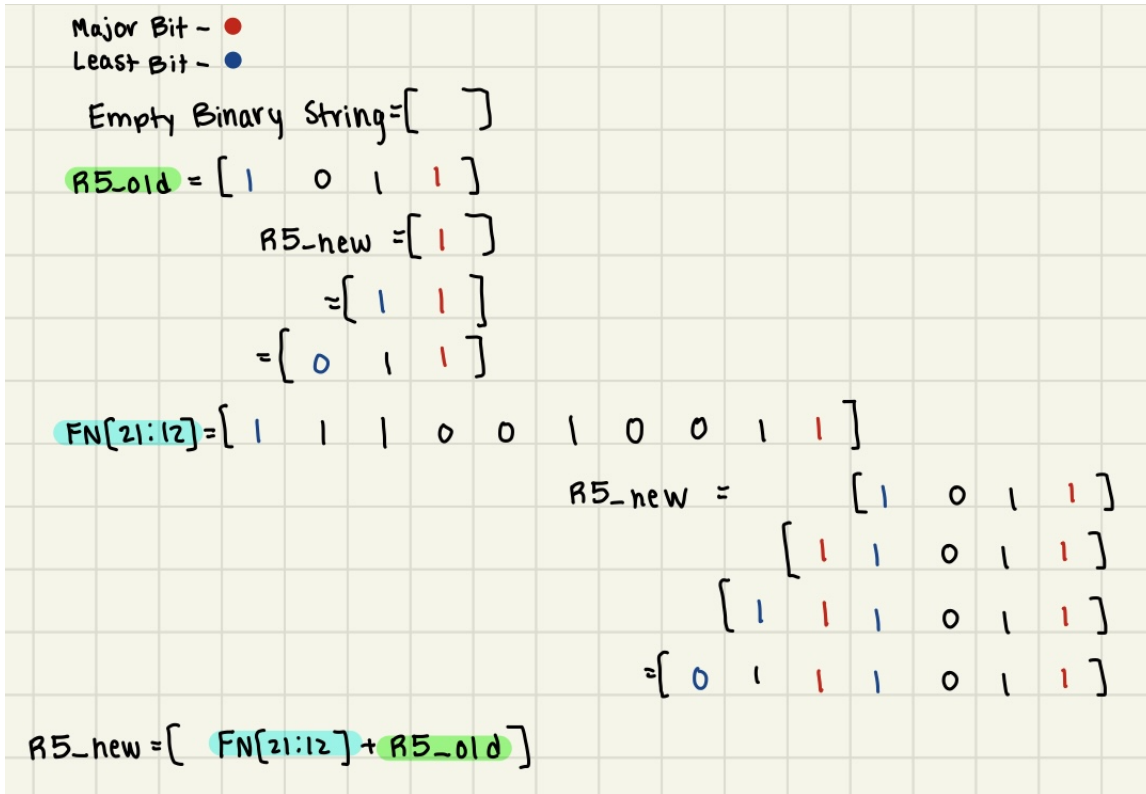


Figure 6: Defining register R5.

Taking the length of the binary representations, we use while loops that take the major bit and implements it at the end of the empty binary representation. After this first step, we subtract the length by one which leads us closer to the least bit of the binary representations. While the length of the binary string is greater than 1, we proceed to implement bits.

With the sliced Numerator, Function Numerator, and the Function Denominator placed in its correct location – we have what we see in Eqs. (2.6 - 2.8).

III. FREQUENCY SETTINGS

As the magnetic field shifts the frequency needed in order to enter the dark state, which is how we measure magnetic fields, has to be adjusted to match. Our next task is to develop a program that can adjust the frequency of a laser modulation in a certain amount of time;

we call this sweeping frequency. The issues when changing frequencies lies on how fast the program can control the LMX2487 chip and how it reads frequency inputs.

The LMX2487 chip works off of eight different registers. Initially, registers R0 to R7 are defined as:

Register	Value
R0	11197502
R1	4352515
R2	48238
R3	15925287
R4	10618633
R5	393579
R6	8126541
R7	10639

Table 1: Initial Registers Definition.

Six of the registers are predefined while the other two [R0, R5] are calculated with every new frequency. In order for the chip to process the given values, our program takes a register, converts it into a binary string, adds bits so it consists a total of 24 bits, then defines the binary string in terms of its register in a binary table labeled “initSettings” or initial settings. The initSettings goes as follows:

$$initSettings = [R0 = [BinaryString], \dots, R7 = [BinaryString]] \quad (3.1)$$

Every frequency will be producing new binary strings. R0 and R5 are the ones that change. Looking at e.q. 2.6 and 2.8, we see the strings include F_N . This value changes given its definition in e.q. 2.5. Once a register is changed, it’s labeled, then inserted into a new initialized settings.

There are three arguments taken when sweeping frequencies. First, we want to get from frequency A to B. Second, getting from A to B must take N total steps. Lastly, the whole process takes a certain amount of time, t. The procedure goes as:

$$N_{sub} = \frac{A - B}{N} \quad (3.2)$$

N_{sub} is the size of the step it must take to get from A to B in N total steps. Next, each of these increments takes an amount of time, t_{sub} , where the sum of every t_{sub} is the total time t :

$$t_{sub} = \frac{t}{N} \quad (3.3)$$

N_{sub} takes t_{sub} , both sum up to the total step N and time t . The program successfully sweeps frequencies. Though, with every new frequency, the program needs to calculate new registers which takes a certain amount of time. This negatively impacts the t_{sub} between every new frequency. The next task is to optimize the program to improve time efficiency.

IV. DEBUGGING

The issues found within the algorithm were mostly surrounded by the arrangements of the binary representations. In the first steps of developing the algorithm, the major bit seemed to sit at the left end of a given binary string, but this is not true. For it to be true the major bit would have to sit to the right end of a given binary representation since it's the first bit being generated. Realizing that binary bits are represented as the least to major bit helped resolve the issue. When slicing the binary representation, bits were missing. The positions of the bits were not being taken note of. The length of the binary representations gave errors because of the missing bits, which led to imprecise slicing. The correct register binary representations have been done.

Sweeping frequencies has its own issues. Every frequency generates new registers. In order to receive the new registers, one needs to go through the steps as seen in Fig. 4. The average time it takes to calculate registers is 0.42 seconds. Therefore, every t_{sub} is actually $t_{sub} + 0.42s$; making our total time more than needed. To decrease the amount of time, the process of converting integers to binary, slicing binary strings, and joining them needs to be more efficient.

V. RESULTS

The process of creating new registers with every new frequency took 0.42s. After the optimization, the delay is now only 0.06s.

Register R1 was taking the most amount of time, approximately 0.2s. Referring back to Eq. 2.15, R1 is defined by F_D , this is held constant with every new frequency. When we initialize our settings we include R1, since R1 does not change with every new frequency. The functions introduced in Fig. 1 were using empty binary tables. Using empty binary tables lags the program 0.1s on average. Instead of the empty binary tables, we took the binary tables given in the argument. For example, if we needed to slice a binary table we use the existing binary table and remove the bits that were not being called upon.

Another lag for the program were the many unnecessary print functions. The only information needed were which registers were defined by which frequencies.

The program can make binary strings successfully seen in Eqs. 2.6 - 2.8. Also, The registers can be defined by their own frequencies. Sweeping frequencies now have a better delay; though, it can still use some improvements.

VI. CONCLUSION

The current program can take integers, convert it to binary, slice's binary strings and join them together; with this, it can create registers needed in order to modulate frequencies. The program can also sweep frequencies. It can run through the process seen in Fig. 4 with only 0.06s of delay. The next step will be to continue the optimization of the program. And whether the optimized program can still perform the tasks seen in Fig. 4. Once this is completed, it will be connected to the LMX2487 chip in order to modulate the frequency. Which will then lead to taking measurements of magnetic field gradients.

VII. REFERENCES

- [1] A. Fey, "Atomic Magnetometry for the Detection of Cardio-magnetic Fields" (2020). Undergraduate Honors Theses.
- [2] Texas Instruments, "LMX2487 1-GHz to 6-GHz High Performance Delta-Sigma Low-Power Dual PLLatinum™ Frequency Synthesizers With 3-GHz Integer PLL," January 2016.