



**HACETTEPE UNIVERSITY
DEPARTMENT OF ELECTRICAL AND ELECTRONICS
ENGINEERING**

INTERNSHIP REPORT

Name Surname : ÖMER SAMİ ŞAHİN

Student Number: 21329554

Class : 4th Grade

Name of Company: ASELSAN A.Ş.

Internship Start and End Date : 12.06.2017-28.07.2017

Internship Duration : 30 Workdays

**Address of Company: Mehmet Akif Ersoy Mahallesi 296. Cadde No:16,
06370 Yenimahalle-Ankara,Türkiye**

TABLE OF CONTENTS

1)INTRODUCTION TO THE COMPANY.....	1
2)ORGANIZATIONAL STRUCTURE.....	2
3)INTERNSHIP PERIOD.....	3
3.1)WEEK-1	
➤ ASELSAN PRESENTATION.....	3
➤ OCCUPATIONAL HEALTH AND SAFETY.....	3
➤ ELECTROSTATIC DISCHARGE(ESD).....	4
3.2)WEEK-2	
➤ INTRODUCTION TO C#.....	4
• Data Types	4
• Classes.....	5
• Methods.....	5
• Arrays.....	6
• Threads.....	6
• Timer.....	7
• Access Specifier.....	7
• File Handling(I/O)	7
➤ CAN BUS COMMUNICATION PROTOCOL.....	8
• CAN History.....	8
• Object Dictionary.....	8
• CAN Message Format.....	9
• Service Data Objects(SDOs)	10
• Network Management(NMT) Services.....	10
• CAN Messages.....	10
3.3)WEEK-3	
➤ FIRST GUI OF THE PROJECT.....	11
3.4)WEEK-4	
➤ THE PROJECT(Continue)	13
3.5)WEEK-5	
➤ THE PROJECT(Continue)	16
3.6)WEEK-6	
➤ END OF THE PROJECT.....	18
4)CONCLUSION.....	24
5) REFERENCES.....	25

1) INTRODUCTION TO THE COMPANY

ASELSAN is a company of Turkish Armed Forces Foundation, established in 1975 in order to meet the communication needs of the Turkish Armed Forces by national means.

Vision:

Being a national technology company that maintains its sustainable growth by creating value in the global market; preferred due to its competitiveness, trusted as a strategic partner, and caring for the environment and people.

Mission:

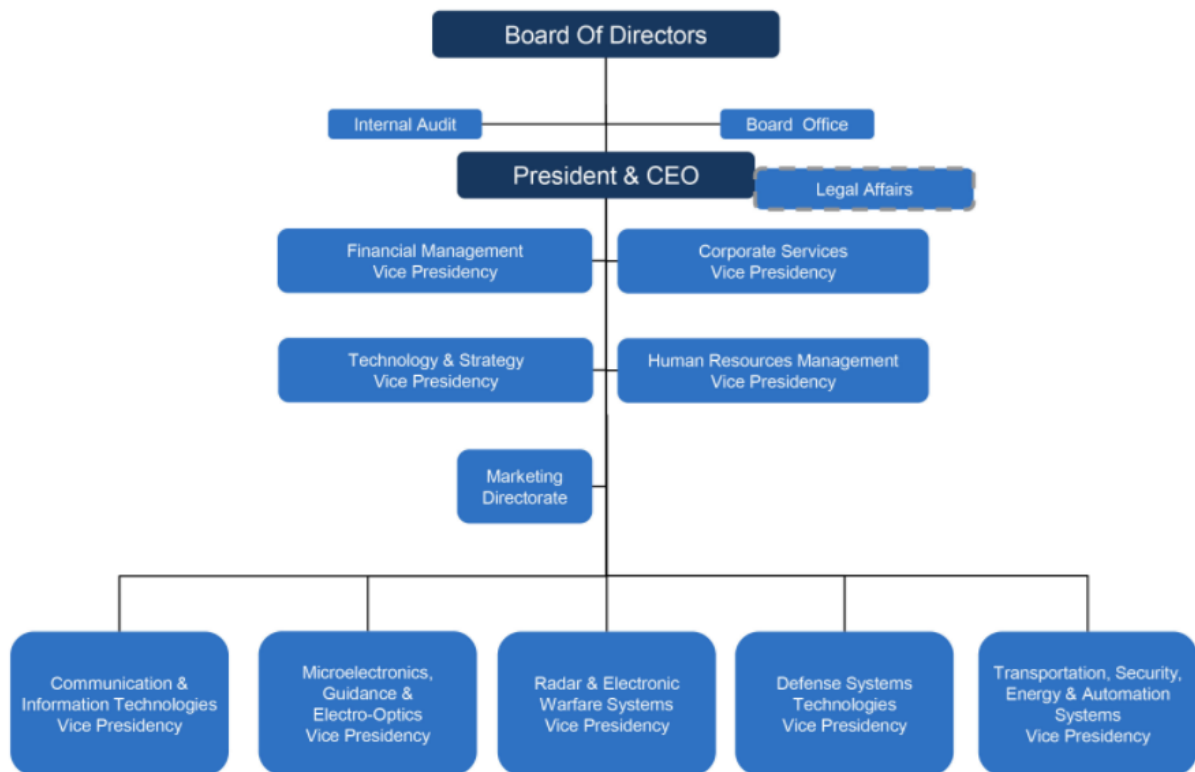
By focusing primarily on the needs of the Turkish Armed Forces; to provide high-value-added, innovative and reliable products and solutions to both local and foreign customers in the fields of electronic technologies and system integration; continuing activities in line with global targets as well as increasing brand awareness and contributing to the technological independence of Turkey.

ASELSAN designs, develops and manufactures modern electronic systems for military and industrial customers, in Turkey and abroad. According to the field of activities, ASELSAN has been organized in five business sectors:

- Communication and Information Technologies Business Sector (HBT)
- Microelectronics, Guidance and Electro-Optics Business Sector (MGEO)
- Radar and Electronic Warfare Systems Business Sector (REHIS)
- Defense Systems Technologies Business Sector (SST)
- Transportation, Security, Energy and Automation Systems Business Sector (UGES)

These five business sectors have high-technology and automated infrastructure in engineering and production. Electronic production includes surface mount technology, multilayer and flexible printed circuit boards, mechanical and mould productions, system integration and test fields. While Communication and Information Technologies Business Sector's main product spectrum covers military and professional communications systems, Radar and Electronic Warfare Systems Business Sector's main operations are focused on radar and electronic warfare, Defense Systems Technologies Business Sector's main operations are focused on command-control systems. Microelectronics, Guidance and Electro-Optics Business Sector manufactures hybrid microelectronic circuits, night vision devices, thermal cameras, laser ranger/designators and inertial navigation systems.

2) ORGANIZATIONAL STRUCTURE



3) INTERNSHIP PERIOD

3.1) WEEK-1

ASELSAN PRESENTATION

This is the first week of my internship. It was actually an orientation week and to give us opportunity to know environment and also we got some trainings. The week started with a brief presentation of ASELSAN. It was talked about some informations about the company. These are the history, vision mission, organizational structure, facilities, number of employees, presentation of departments and information security policy of ASELSAN. Information security is very important for ASELSAN. This is important for both personal security and company security. They warned us about not taking photo, not using flash disk, CD, etc. or not sharing informations about any project during the internship. They also mentioned about conditions how to get a job and how they make an interview in ASELSAN.

OCCUPATIONAL HEALTH AND SAFETY

Occupational health and safety trainings were presented by an expert. I learned very important topics. I will briefly explain some of these topics. To promote the safety of employees, the laws and regulations are enforced by the government. A company have to provide training to employees for occupational health and safety trainings. Also, the employees are in charge to learn these trainings and to apply any precaution. Security and health signs are very important. In the case of an accident or emergency, it is important to know where the emergency equipment is and what to do. Therefore, we have to know the meanings of warning signs. An example is as shown below:



Hazards might be everywhere in a company. Firstly, we should try to annihilate these hazards and to take security precautions. We have to use safety clothing and equipments. It was shown some videos about occupational safety and health and one of them is about safety clothing and equipments. In the video, there was a person who wears ear-plugs when he works near loud machinery and other person who doesn't wear it. I observed that, the person who doesn't wear ear-plug cannot hear good anymore in the video and his health is a risk for a company. Other important topic is cleanliness and organization of workplace. To work productive and safety in a company, we have to carry about this topic. The last thing that I will mention is first aid. This is also very important topic. If we didn't get a training about first aid, we musn't try to treat a patient. Because if emergency action are not done properly, the patient can even die. There are so many examples of this situation.

ELECTROSTATIC DISCHARGE(ESD)

There was also a presentation about electrostatic discharge(ESD).I learned what electrostatic discharge is ,effects of it and damage protection in electronic circuits. ESD is the sudden flow of electricity between two electrically charged objects caused by contact,an electrical short,or dielectric breakdown.The ESD occurs when differently-charged objects are brought close together or when the dielectric between them breaks down,often creating a visible spark.ESD can cause a range of harmful effects of importance in industry,including gas,fuel vapour and coal dust explosions,as well as failure of solid state electronics components such as integrated circuits.In a company,to prevent electrostatic discharge, it is taken precautions.For example,It is built an area called Electrostatic Discharge Protected Area(EPA).The main principle of an EPA is that there are no highly-charging materials in the vicinity of ESD sensitive electronics,all conductive materials are grounded,workers are grounded and charge build-up on ESD sensitive electronics is prevented.

- After these trainings,we were sent to our departmants.Engineers introduced my department and laboratories to me.I did my internship in Defense Systems Technologies Business Sector(SST).My department was Power and Control Systems-Electronic Design.I only worked in this department during my internship and they gave me a project.

3.2) WEEK-2

In this week, I did some researches to start my project and wrote them in two main topics which are 'INTRODUCTION TO C#' and 'CAN BUS COMMUNICATION PROTOCOL'. In general,my project is to create a GUI and using this interface ,I will try to communicate with a device by using CAN protocol.Therefore,I needed to learn C# to create the GUI and to learn some parts of CAN communication protocol.I didn't explain and research all informations about CAN and C#.I just explained the basic and very important informations that are only related to my project and that mostly used in the project.

INTRODUCTION TO C#

Data Types

Data types in a programing language describes that what type of data a variable can hold.C# is a strongly typed language,therefore every variable and object must have a declared type.In C#,it is possible to convert a value of one type into a value of another type.

The following table lists the data types available in C# along with the range of values possible for each data type:

Type	Size in Bytes	Description	Minimum	Maximum	Example
bool	1	Named literal	false	true	
sbyte	1	Signed byte	-128	127	
byte	1	Unsigned byte	0	255	
short	2	Signed short integer	-32768	32767	
ushort	2	Unsigned short	0	65535	
int	4	Signed integer	-2147483648	2147483647	
uint	4	Unsigned integer	0	4294967295	
long	8	Signed long int	-9.2233E+18	9.2233E+18	
ulong	8	Unsigned long int	0	18446E+19	
char	2	Unicode character, contained within single quotes.	0	128	a,b,4
float	4	floating point	-3.402823E+38	3.402823E+38	3.14159
double	8	Floating point	-1.7976931E+308	1.7976931E+308	3.14159
decimal	16	Floating point, accurate to the 28th decimal place.	-7.9228E+24	7.9228E+24	
object	8+	Base type for all other types	n/a	n/a	n/a
string	20+	Immutable character array	n/a	n/a	"Hello World"
DateTime	8	Represents an instant in time, typically expressed as a date and time of day.	00:00:00 01/01/0001	23:59:59 31/12/9999	14:289:35 08/05/2010

Classes

Class is a template or declaration that is used for classifying object. It encapsulates variable members, functions, structure, properties and many more components. It is the basic building of object oriented programming. We can categorize our code and debug easily by using classes. In order to create class, the class keyword is used. Let's have a look at an example shown below:

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            ogrenci omer = new ogrenci();
            ogrenci ahmet = new ogrenci();
            omer.adsoyad = "ÖMER SAMİ ŞAHİN";
            omer.sinif = "4";
        }
    }
}

namespace ConsoleApplication1
{
    class ogrenci
    {
        public string adsoyad;
        public string sinif;
        public string ders;
        public string dersnotu;
    }
}
```

In this code, I created a class called 'ogrenci' and it has some variables. To use the members of class, we need to create object of this class. An object is created using new keyword. There are two objects which are 'omer' and 'ahmet'. I assigned the values in string type to variables in the 'ogrenci' class by using the objects.

Methods

Method is the building block of object-oriented programming. It combines related code together and makes program easier. After creating method, you need to call it in Main()

method to execute. In order to call method, you need to create object of containing class, then followed by dot(.) operator you can call the method. If method is static, then there is no need to create an object and you can directly call it followed by class name. Let's have a look at an example shown below:

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            ekran e = new ekran();
            e.yaz();
        }
    }
}

namespace ConsoleApplication1
{
    class ekran
    {
        public void yaz()
        {
            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine("Ekran Çıktısı");
            }
        }
    }
}
```

In this code, I created a method called 'yaz()' in the class called 'ekran'. Whenever I want to print 'Ekran Çıktısı' five times, I just need to call the method in one line. We don't need to write the same code everytime. This is the main purpose of creating methods. To call the method, I created an object of the class 'ekran' and then call the method in the main class.

Arrays

Array is a collection of variable of same data type. If you have declare 1000 integer variable, then you can declare an integer type array of 1000 size. The value of array can be accessed using index position of array. The first index position of array is zero. Let's have a look at an example shown below:

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] sayilar = new int[5];
            string[] harfler = {"A", "B", "C" };

            Console.WriteLine(harfler[0]);
            Console.WriteLine("{0}", sayilar[3]);
        }
    }
}
```

In this code, instead of creating 5 different variables for 'sayilar', I created one array and indicated its size. Also, without initializing the size of the array, we can assign the values directly and it indicates the size of the array automatically.

Threads

A thread is an independent execution path with its own resources allocated by the CPU. Threads run in parallel within a process just as processes run in parallel on a computer. A process can have multiple threads and they share memory among them. Each thread has its own context of execution.


```
using System.Threading;

class Program
{
    public Thread katkas1_kalibrasyon_thread;

    static void Main(string[] args)
    {
        katkas1_kalibrasyon_thread = new Thread(new ThreadStart(this.katkas1_kalibrasyon));
        katkas1_kalibrasyon_thread.Priority=ThreadPriority.Lowest;
        katkas1_kalibrasyon_thread.Start();
    }
}
```

In this code,I show how to create a thread and how to use it.We need to call the library 'System.Threading' to use the thread .We can indicate priority of a thread.

Timer

The timer control allows us to set a time interval to periodically execute an event at a specified interval.It is useful when we want to execute certain applications after a certain interval.The main difference between timer and thread is that timer functions cannot work in parallel.

```
private void timer1_Tick(object sender, EventArgs e)
{
    timer1.Interval = 1000;
    sayac++;
}
```

In this code,timer interval is set to 1000 ms.This means that the code goes into timer1_Tick function once in a second and variable 'sayac' is increased by 1.

Access Specifier

Access specifiers defines the scope of a class member.A class member can be a variable or a function.I will explain two important access specifier that is frequently used.

Public :The class member,that is defined as public can be accessed by other class member that is initialized outside the class.A public member can be accessed from anywhere even outside the namespace.

Private : The private access specifiers restrict the member variable or function to be called outside from the parent class. A private function or variable cannot be called outside from the same class. It hides its member variable and method from other class and methods.

File Handling(I/O)

To save the information permanently on the disk or reading information from the saved file through C# is known as File Handling. 'StreamWriter' is used to write something to a file and 'StreamReader' is used to read something from a file.Let's give an example for writing to a file. In this example,I created a txt file and wrote 'ÖMER SAMİ ŞAHİN' to the file.

```
StreamWriter SW;

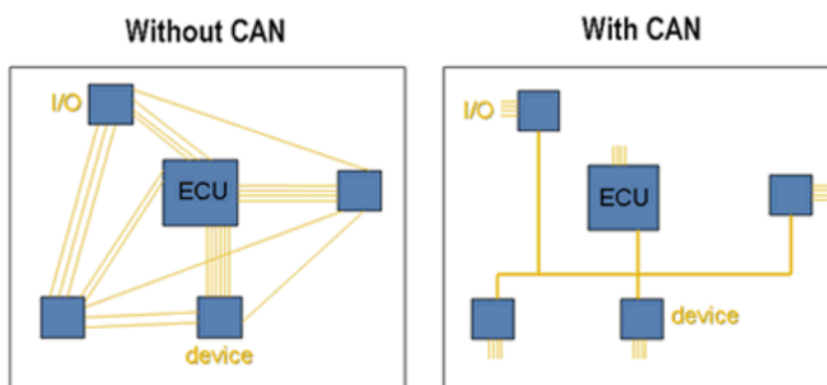
SW = File.CreateText("C:\\Users\\Administrator\\Desktop\\OMER\\TMAB_TKTM_kalibrasyon_log.txt");

SW.WriteLine("ÖMER SAMİ ŞAHİN");
```

CAN BUS COMMUNICATION PROTOCOL

CAN History

Bosch originally developed the Controller Area Network(CAN) in 1985 for in-vehicle networks. In the past, automotive manufacturers connected electronic devices in vehicles using point-to-point wiring systems. Manufacturers began using more and more electronics in vehicles, which resulted in bulky wire harnesses that were heavy and expensive. They then replaced dedicated wiring with in-vehicle networks, which reduced wiring cost, complexity, and weight. CAN, a high-integrity serial bus system for networking intelligent devices, emerged as the standard in-vehicle network. The automotive industry quickly adopted CAN and, in 1993, it became the international standard known as ISO 11898. Since 1994, several higher-level protocols have been standardized on CAN, such as CANopen and DeviceNet.



CAN Properties:

- Multimaster concept
- Low-Cost, lightweight network
- Number of nodes not limited by protocol
- No node addressing. Message identifier specifies contents and priority.
- Message based protocol
- Easy connection/disconnection of nodes
- Broadcast communication
- Sophisticated error-detection and error handling mechanism
- High data transfer rate of 1000 kbps at a maximum bus length of 40 meters

Object Dictionary

One of the central themes of CANBus is the object dictionary(OD), which is essentially a table that stores configuration and process data. The object dictionary is the method by which a CAN device can be communicated with. It is a requirement for all CAN devices to implement an object dictionary. The CAN standard defines a 16-bit index and an 8-bit sub-index. That is, it is permissible to have up to 65536 indices and up to 256 subentries at each index. The standard defines that certain addresses and address ranges must contain specific parameters.

There are two communication mechanisms for accessing the object dictionary:

1. SDO(Service Data Object)

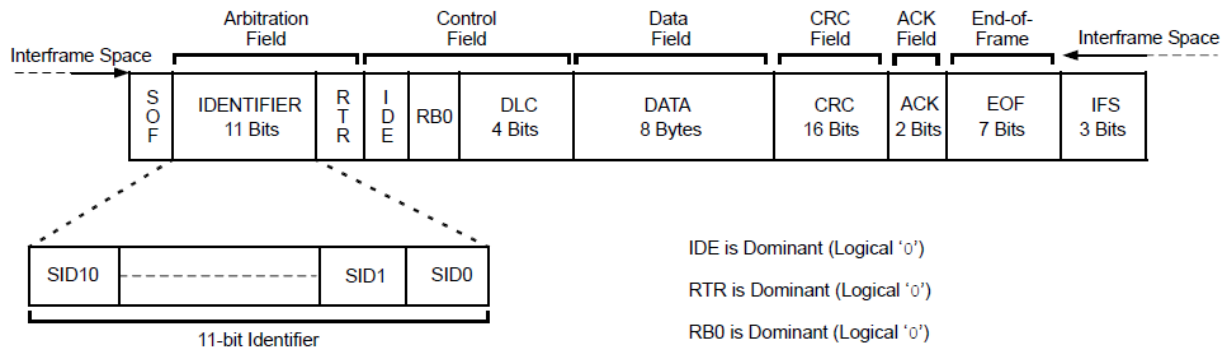
2. PDO(Process Data Object)

SDOs are not periodic messages. Data transmit or receive is not continually.

PDOs are periodic messages. PDOs are often used to transfer continually updating data.

CAN Message Format

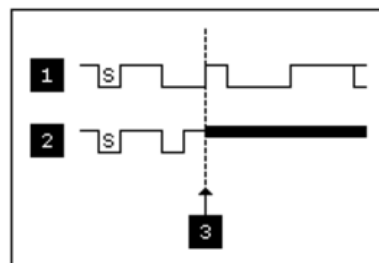
In CAN protocol, messages are sent in frame format as shown below.



SOF(start-of-frame)bit : Indicates the beginning of a message with a dominant (logic 0) bit.

IDENTIFIER(COB-ID) : Identifies the message and indicates the message's priority. Frames come in two formats. Standard, which uses an 11-bit arbitration ID, and extended, which uses a 29-bit arbitration ID.

NODE-ID : Every node has its own ID. Node-ID range is 1 to 127.



1 Device A: ID = 11001000111 (647 hex)

2 Device B: ID = 11011111111 (6FF hex)

3 Device B Loses Arbitration; Device A Wins Arbitration and Proceeds

S = Start Frame Bit

➤ This picture explains the priority of a message depends on its COB-ID.

RTR(Remote transmission request)bit : Serves to differentiate a remote frame from a data frame.

IDE(identifier extension) bit : Allows differentiation between standard and extended frames.

DLC(data length code) : Indicates the number of bytes the data field contains.

Data Field : Contains 0 to 8 bytes of data

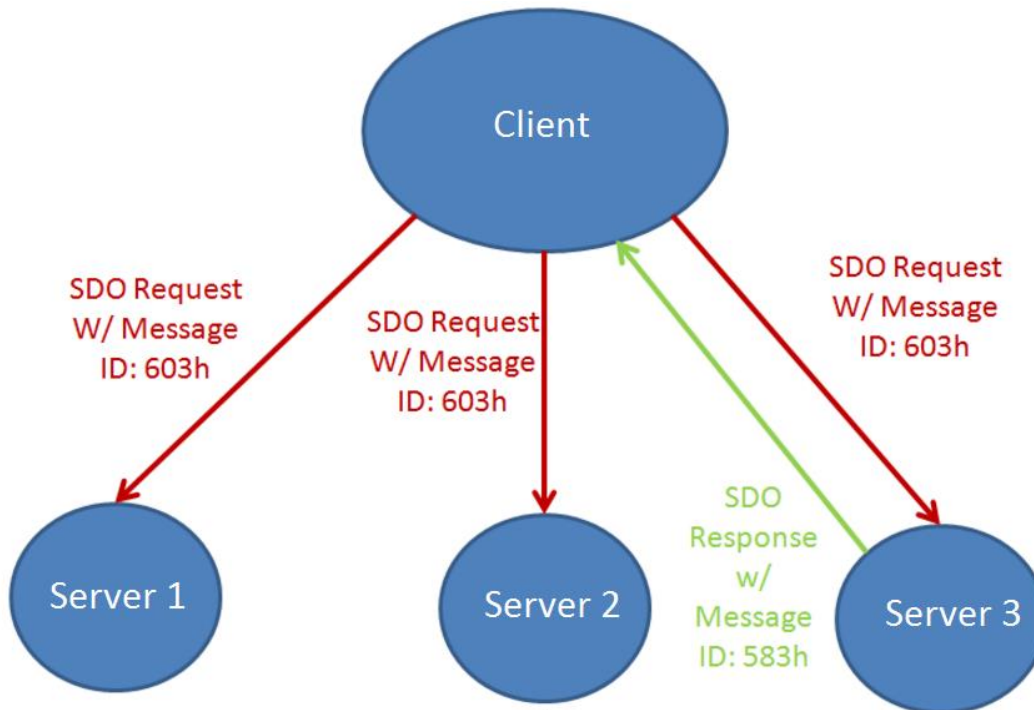
CRC(cyclic redundancy check) : Used for error detection.

ACK(ACKnowledgement) slot : Any CAN controller that correctly receives the message sends an ACK bit at the end of the message.

EOF(end of frame) : Indicates the end of the message.

Service Data Objects(SDOs)

- COB-ID of the transmitted messages from client to server is **600h+Node-ID**.
- COB-ID of the received messages from server to client is **580h+Node-ID**.



In this example, we can understand that the message is SDO message. Because transmitted message has a COB-ID of 603 and received message has a COB-ID of 583. Also we can understand the Node-ID of the message which is 3. Therefore, only, Server 3 accepts the SDO Request and sends the SDO Response to the client with its own Node-ID.

Network Management(NMT) Services

Network management services include the ability to change the state of a slave between initializing, pre-operational, operational and stopped. The NMT protocol allows for the CAN network to control the communication state of individual nodes. After Power-On, a node is in the 'Initializing' state and then switches itself to the 'Pre-operational' state automatically. The pre-operational state is mainly used for the configuration of CAN devices. In the stopped state, a node can only do node guarding or heartbeats, but cannot receive or transmit messages. PDO communication is not permitted in the pre-operational state but SDOs are allowed.

CAN Messages

SYNC: SYNC message must be sent with smaller than 4 seconds period of time to all nodes.

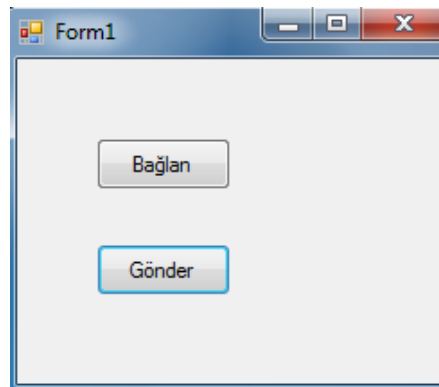
EMERGENCY: If SYNC messages cannot be sent properly to all nodes, nodes send 'EMERGENCY' message with their own NODE-ID. And nodes switch to pre-operational state from operational one.

HEARTBEAT : In the heartbeat protocol, a CAN node sends out a heartbeat message which lets the heartbeat consumer know that the node is still alive. There are three state of heartbeat messages which are operational mode heartbeat, pre-operational mode heartbeat and stop mode heartbeat.

3.3) WEEK-3

FIRST GUI OF MY PROJECT

In week 2, I did a lot of research and learned basics of C# and CAN protocol. In this week, I was ready to start my project. I will explain the purpose of my project in a technical details. There is a unit to communicate and in this unit, there are three different integrated circuits which are called KATKAS, SETKAS and motherboard. In each integrated circuit except motherboard, there are two locks called Lock-1 and Lock-2. By using a software interface created in C# and using CANBus protocol, I will send some commands and in terms of the position of the locks, I will detect whether the locks are locked or unlocked. After this detection, calibration of the locks are done automatically in a proper way by using the GUI. During the week, I worked on creating and improving my GUI and tried to communicate with a device properly. The first GUI of my project is as shown below:



For ‘Bağlan’ and ‘Gönder’ buttons, the code is as the following:

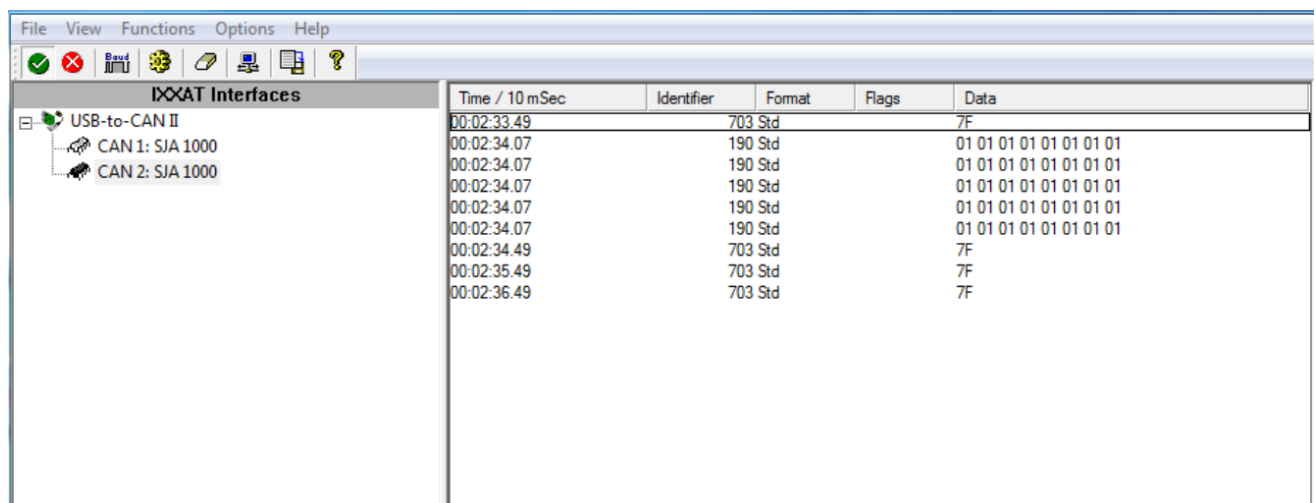
```
private void button1_Click(object sender, EventArgs e) //Gönder butonu
{
    byte[] data = new byte[8];
    int i;
    for (i = 0; i < 8; i++)
    {
        data[i] = 0x01;
    }
    CAN_Library.CANMessage msg = new CAN_Library.CANMessage(false, 400, data);

    for (int p = 0; p < 5; p++)
    {
        canComponent1.sendMessage(msg);
    }
}

private void connect_Click(object sender, EventArgs e) //Bağlan butonu
{
    canComponent1.baudrate = CAN_Library.BaudRate.b500Kbit;
    canComponent1.CANNo = (byte)0;
    canComponent1.init();
    canComponent1.start();
}
```

When we click the ‘Bağlan’ button, by using `canComponent1.init()` and `canComponent1.start()` commands, we connect to the CAN interface.

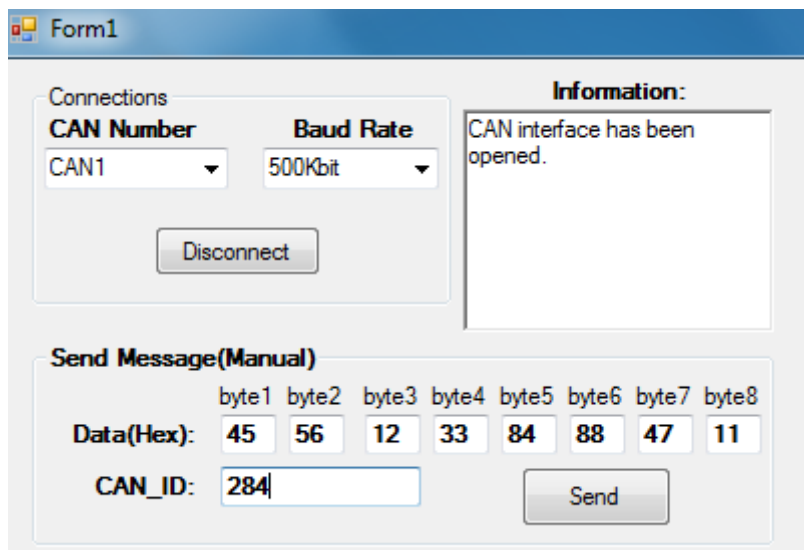
These commands are specified in the CAN library. Also baud rate value is automatically initialized to 500kbit and CAN number is set to zero which means CAN1 channel. It means that CAN1 channel is used to transmit messages and CAN2 channel is used to receive messages in this code. If we connect to the CAN interface properly, this means that the communication is started. In this situation, we can send any message to a node or receive any message from a node. I randomly initialized the 8 bytes data value to 0x01h and CAN_ID to 400. To send a message, `canComponent1.sendMessage()` command is used. It is also specified in the CAN library. The same data with its CAN_ID was sent 5 times to the node. Pre-operational heartbeat messages are received periodically from the node. After sending and receiving messages, first communication is done. Also, these messages were observed in a computer program as shown below:



Time / 10 mSec	Identifier	Format	Flags	Data
00:02:33.49	703 Std			7F
00:02:34.07	190 Std			01 01 01 01 01 01 01 01
00:02:34.07	190 Std			01 01 01 01 01 01 01 01
00:02:34.07	190 Std			01 01 01 01 01 01 01 01
00:02:34.07	190 Std			01 01 01 01 01 01 01 01
00:02:34.07	190 Std			01 01 01 01 01 01 01 01
00:02:34.49	703 Std			7F
00:02:35.49	703 Std			7F
00:02:36.49	703 Std			7F

There is no RESET Node button, RESET Communication button and their codes. Therefore, we cannot manually or automatically change the state of any node.

In the forthcoming days, I made some improvements on the interface as the following:



I added two combobox and 'Connect' button to the 'Connections' groupbox. These are 'CAN Number' and 'Baud Rate'. We can choose the CAN Number and Baudrate from comboboxes instead of changing the code every time.

When we click the 'Connect' button, CAN interface is opened and 'Connect' button turns into 'Disconnect' button. 'Information' richtextbox was created and some informations are automatically written in the richtextbox. When 'Connect' button is clicked, if the CAN connection is achieved, 'CAN interface has been opened' is written in the information box. If the CAN connection is failed, 'CAN interface cannot be opened' is written in the information box. When 'Disconnect' button is clicked, if the CAN disconnection is achieved, 'CAN interface has been closed' is written in the information box. If the CAN disconnection is failed, 'CAN interface cannot be closed' is written in the information box. For sending messages manually, I also created 'Send Message(Manual)' groupbox and added data and CAN_ID textboxes for typing the values on the screen. When we click the 'Send' button, code sends the data and its CAN_ID to all nodes. If we try to send a message without connected to the CAN interface, a messagebox appears on the screen and warns us to connect to the interface. There is no communication log to see all messages on the screen so that we use a computer program to observe the messages.

3.4) WEEK-4

THE PROJECT(Continue)

In comparison with week 3, I added some things to the interface and made improvements. The hardest part of this week for me was to convert data types each other and print them on the screen. I will continue to explain my project in details below.

The screenshot shows a Windows application window titled "Form1". It contains several sections:

- Connections:** A dropdown for "CAN Number" set to "CAN1", a dropdown for "Baud Rate" set to "500Kbit", and a "Disconnect" button.
- Information:** A text box displaying "CAN interface has been opened."
- Send Message(Manual):** A section with "Data(Hex)" fields for bytes 1 through 8 (values: 11, 22, 33, 34, 67, 88, 46, 21), a "CAN_ID" field with the value "123", and a "Send" button.
- COMMUNICATION LOG:** A table with columns: TX-RX, ID, D1, D2, D3, D4, D5, D6, D7, D8, and Message Type. The log shows a series of RX and TX messages, all with "SYNC" as the message type.
- Right Panel:** A section with "SYNC sending at every" set to "500" ms, a checked "Show Sync" checkbox, and a "Refresh" button.

As I mentioned before, I used a computer program to see all messages which are received and sent. If we observe a lot of messages instead of one or two, it is very difficult to follow which message is sent or received. Because the program to observe the messages that I mentioned only shows us data and its id and we have to find the message type by looking at the data and id value one by one. We can even receive 20 messages in one second if there are lots of nodes.

To follow our messages easily and indicate the type of the message only using our interface,I added ‘COMMUNICATION LOG’ screen on the interface. There are some titles on the log to indicate the meaning of the communication. ‘TX-RX’ shows whether a message is sent or received. TX means transmitted message and RX means received message. ‘ID’ shows the CAN_ID of a transmitted or received message in decimal. ‘D1,D2,...,D8’ shows data of a message in byte(hexadecimal). Data length is maximum 8 byte for my project. ‘Message Type’ shows the meaning of a message. It is actually enough to look at ‘TX-RX’ and ‘Message Type’ titles to understand the message.

```
private void timer1_Tick(object sender, EventArgs e)
{
    //Timer enable olduğu sürece bu fonksiyona girer.
    //Belirlenen interval süresi kadar program buraya tekrar tekrar girer.
    if (button2.Text.Equals("Disconnect") && checkBox5.Checked && numericUpDown1.Value!=(int)0)
    {
        for (int d = 0; d < 8; d++)
        {
            data[d] = 0x00; // Sıfır data bilgisi gönder.
        }
        uint id_sync = 80; //CAN SYNC mesajı id değeri 80 dir.
        string x = Convert.ToString(id_sync); //id değeri ekrana yazdırmak için string ifadeye çevrilir.

        CAN_Library.CANMessage msg = new CAN_Library.CANMessage(false, 80, data);
        canComponent1.sendMessage(msg); // CAN SYNC mesajı periyodik olarak yollanır.

        richTextBox2.AppendText("RX" + "\t" + x + "\tSYNC");
        richTextBox2.AppendText("\n"); //COMMUNICATION LOG ekranına SYNC mesajının id ve data bilgisi yazılır.
    }
}
```

I created ‘timer1_Tick’ function to send SYNC message periodically. SYNC message must be sent with smaller than 4 seconds period of time to all nodes. If it is sent properly ,all nodes work in operational mode. I added an option to change the period of time of SYNC message on the interface. By changing numericupdown in ms, we can set the timer interval. The code goes inside the timer1_Tick function every timer interval which is set and send SYNC message periodically. SYNC message has no data and it must be sent automatically. There is also a filter whether to show the SYNC message on the communication log by clicking the ‘Show Sync’ checkbox. I added this filter because SYNC is periodically written on the screen. Once we ensure the SYNC message is sent properly and see it on the communication log, then we don’t want to see it on the screen every time. Even if we click the checkbox to stop typing the SYNC message on the communication log, the code always send it to all nodes in the background. ‘Refresh’ button is added to clear COMMUNICATION LOG. I also researched some commands in C# which are frequently used in my project. They are ‘Stringbyteparse’, ‘Convert.Tostring’ and ‘.Split’. ‘Convert.Tostring’ is used to convert integer values to string. In communication log, everything is actually a string. To see these values on the log screen, we have to use Convert.Tostring. ‘Stringbyteparse’ is to convert string value to byte. ‘.Split’ is generally used to separate the values in a convenient point.

In the forthcoming days,I made some improvements on the interface as the following:

The screenshot shows a software interface titled 'Form1' with several functional areas:

- Connections:** Includes dropdowns for 'CAN Number' (set to CAN1) and 'Baud Rate' (set to 500Kbit), along with a 'Disconnect' button.
- Information:** A text box stating 'CAN interface has been opened.'
- SKT-1:** A sub-interface containing 'KİLİTLE' and 'AÇ' buttons, an 'Information:' label, and a text box displaying 'KİLİT1 KAPANIYOR...'
- Send Message(Manual):** Features input fields for 'Data(Hex):' (with byte1 through byte8 labels) and 'CAN_ID(Dec):', and a 'Send' button.
- COMMUNICATION LOG:** A table with columns for TX-RX, ID, D1-D8, and Message Type. It includes a 'Show Sync and Heartbeat' checkbox (checked) and a 'Refresh' button. The log shows a series of TX and RX messages, primarily SYNC and HEARTBEAT_PRE_OPMODE_SKT1.
- RESET:** A button located at the bottom right of the interface.

TX-RX	ID	D1	D2	D3	D4	D5	D6	D7	D8	Message Type
TX										SYNC
RX										HEARTBEAT_PRE_OPMODE_SKT1
TX										SYNC
RX										HEARTBEAT_PRE_OPMODE_SKT1
TX										SYNC
RX										HEARTBEAT_PRE_OPMODE_SKT1
TX										SYNC
RX										HEARTBEAT_PRE_OPMODE_SKT1
TX										SYNC
RX										HEARTBEAT_PRE_OPMODE_SKT1
TX										SYNC
RX										HEARTBEAT_PRE_OPMODE_SKT1
TX										SYNC
RX										HEARTBEAT_PRE_OPMODE_SKT1
TX										SYNC

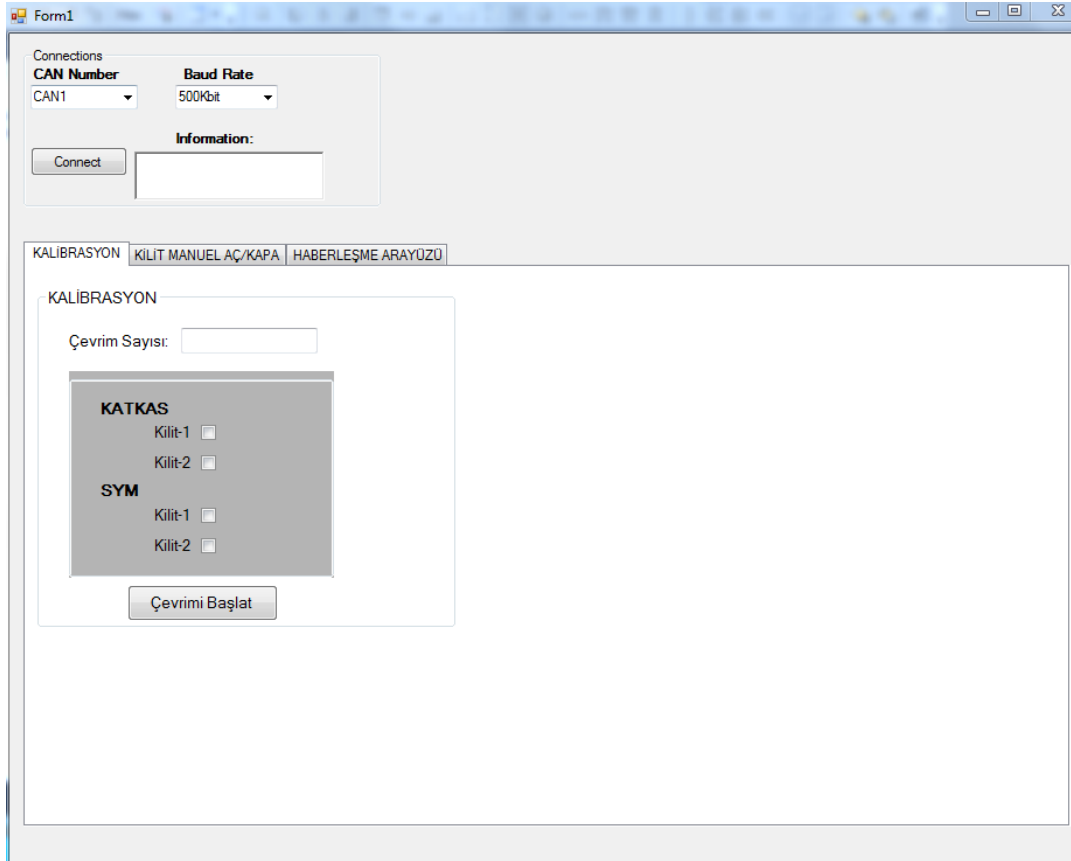
Default value of CAN number and baud rate are initialized as CAN1 and 500kbit respectively on the interface.If SYNC message is sent with smaller than 4 seconds period of time,it means there is no problem.Therefore,I decided to remove numericupdown for changing the period of the SYNC on the screen.Its default value is set to 1 second in the code.Some known messages are added to the code to filter and show it on the screen.For SETKAS and KATKAS ,three states of heartbeat message which are pre-operational,operational and stop mode and can_boot messages are filtered in the code.If these messages are received ,the code automatically filters these messages and shows on the communication log screen as shown above.Both SYNC and heartbeat messages can be filtered whether to show on the log screen by clicking the 'Show Sync and Heartbeat' checkbox.The reason to filter the heartbeat messages on the log screen is the same as the SYNC message.We don't always want to see heartbeat messages.While SYNC messages are periodically sent messages,heartbeat messages are also periodically received messages(every 1 second).'Reset' button is added to the interface.When we click this button,all object dictionary parameters turn into default values and state of the hearbeat messages turns into operational mode.

For KATKAS Lock-1,manual lock/unlock screen(groupbox) is added to the interface.There are two buttons which are 'KİLİTLE' and 'AÇ'. 'Lock' and 'Unlock' are two different messages.We call these messages 'Command message' .When we click 'KİLİTLE' or 'AÇ' buttons,command message is sent to the node which is KATKAS .If lock or unlock command is sent properly,'Confirmation message' is received and it is understood by the code.If the confirmation message is received,'KİLİT1 AÇILIYOR...' or 'KİLİT1 KAPANIYOR' is automatically written in the information box which is inside the SKT-1 groupbox.

3.5) WEEK-5

THE PROJECT(Continue)

In this week,I changed a lot of things on the interface, made improvements and solved some problems in the code.I split the interface into three categories.I explained the details below.



The first tab is 'KALİBRASYON'. This part is the main purpose of my project. For KATKAS and SETKAS, there are two locks for each. We can choose one lock or more than one at the same time. Also we can indicate the cycle number by typing into the textbox. One cycle means that first lock and then unlock. When we click the 'Çevrimi Başlat' button, calibration starts and lock or unlock command is done automatically. I changed some parts of the code. If we want to calibrate more than one lock at the same time, I observed that timer object didn't work. Because timer doesn't work parallel in the code. It means that if the code is inside one of the timer function, then the program cannot go to other lines simultaneously in the code. There is also the same problem for SYNC messages. We have to send SYNC messages periodically even if the code goes to other lines. It again means the program has to work parallel for different works. To solve this problem, I used 'thread' instead of 'timer'. There are 4 different threads for the locks which works parallel in the code. These threads only start when at least one of the checkboxes is chosen and the 'Çevrimi Başlat' button is clicked.

Form1

Connections

CAN Number: CAN1

Baud Rate: 500Kbit

Connect

Information:

KALIBRASYON KİLİT MANUEL AÇ/KAPA HABERLEŞME ARAYÜZÜ

SKT-1

KİLİTLE AÇ

Information:

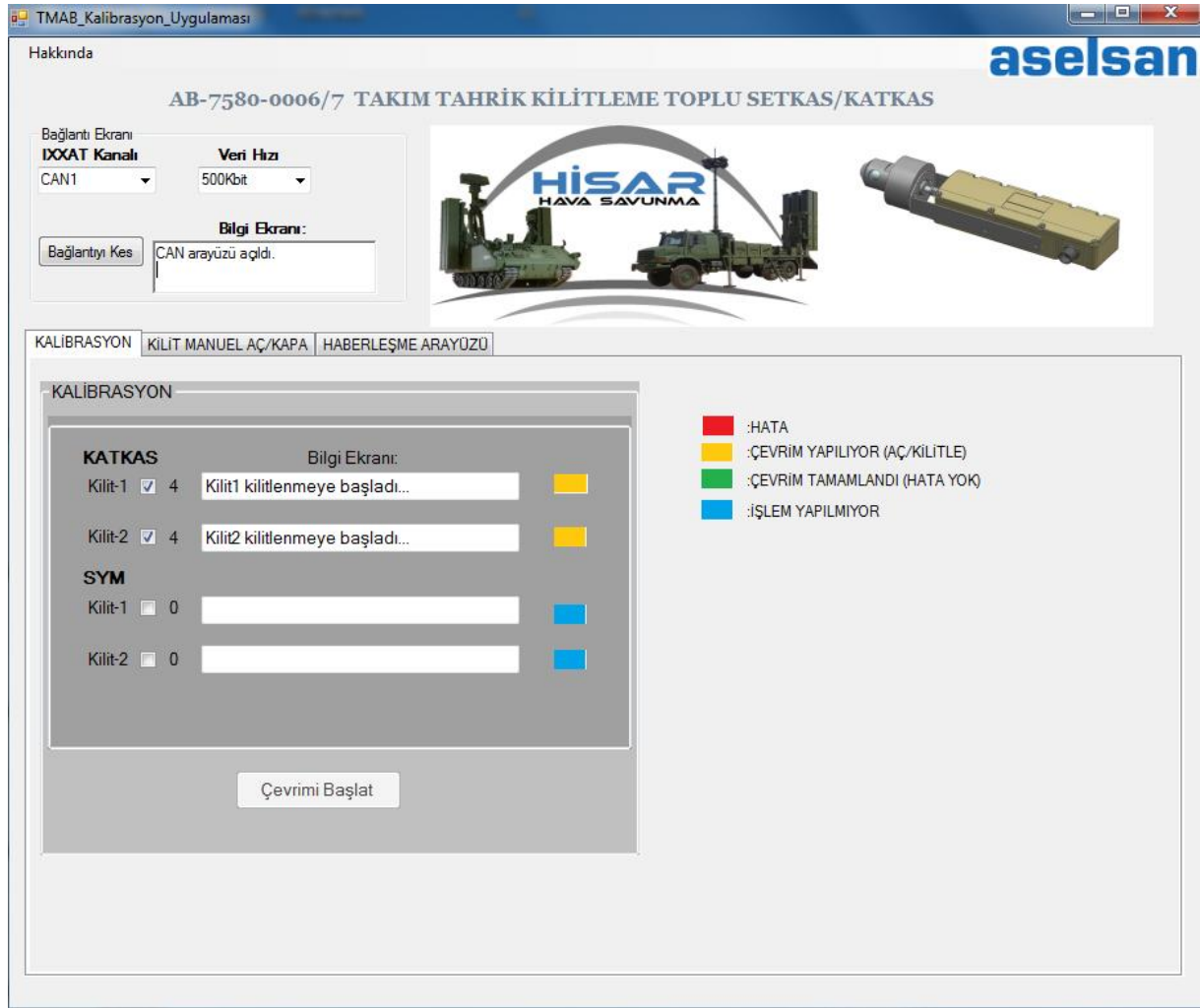
In this tab, we can lock or unlock KATKAS Lock-1 manually. I created a thread called Continuous_Tx_Thread. In this thread, the code sends SYNC message every 3 seconds and also two different messages are added to send once in a second in the thread. These messages are 'status inquiry' and 'error inquiry'. Status inquiry is to learn whether the KATKAS Lock-1 is locked or unlocked. In the embedded software, locked and unlocked position threshold value was set to 3A9h as default. If the Lock-1 position is greater than 3A9h, it means it is locked. If the Lock-1 position is less than 3A9h, it means it is unlocked. There is no undefined situation because threshold value is the same for both locked and unlocked situations. When 'KİLİTLE' or 'AÇ' button is clicked, 'KİLİT AÇIK' or 'KİLİT KİLİTLİ' is written in the information richtextbox.

In this tab,I can only send messages manually and also can see all communication messages which are received or transmitted in the communication log.I added some filters to the code.If I receive a message that is not interested to me ,it is filtered in the code and printed in the communication log as ‘unknown message’.Emergency messages, error messages,state of the locks such as locked ,unlocked can also be filtered and printed in the log screen.Data length of the reset message is 2 bytes but in week 4,I sent it 8 bytes.I observed that RESET button didn’t work properly and fixed this problem in the code.Also I seperated the RESET button into two forms as shown above.

3.6) WEEK-6

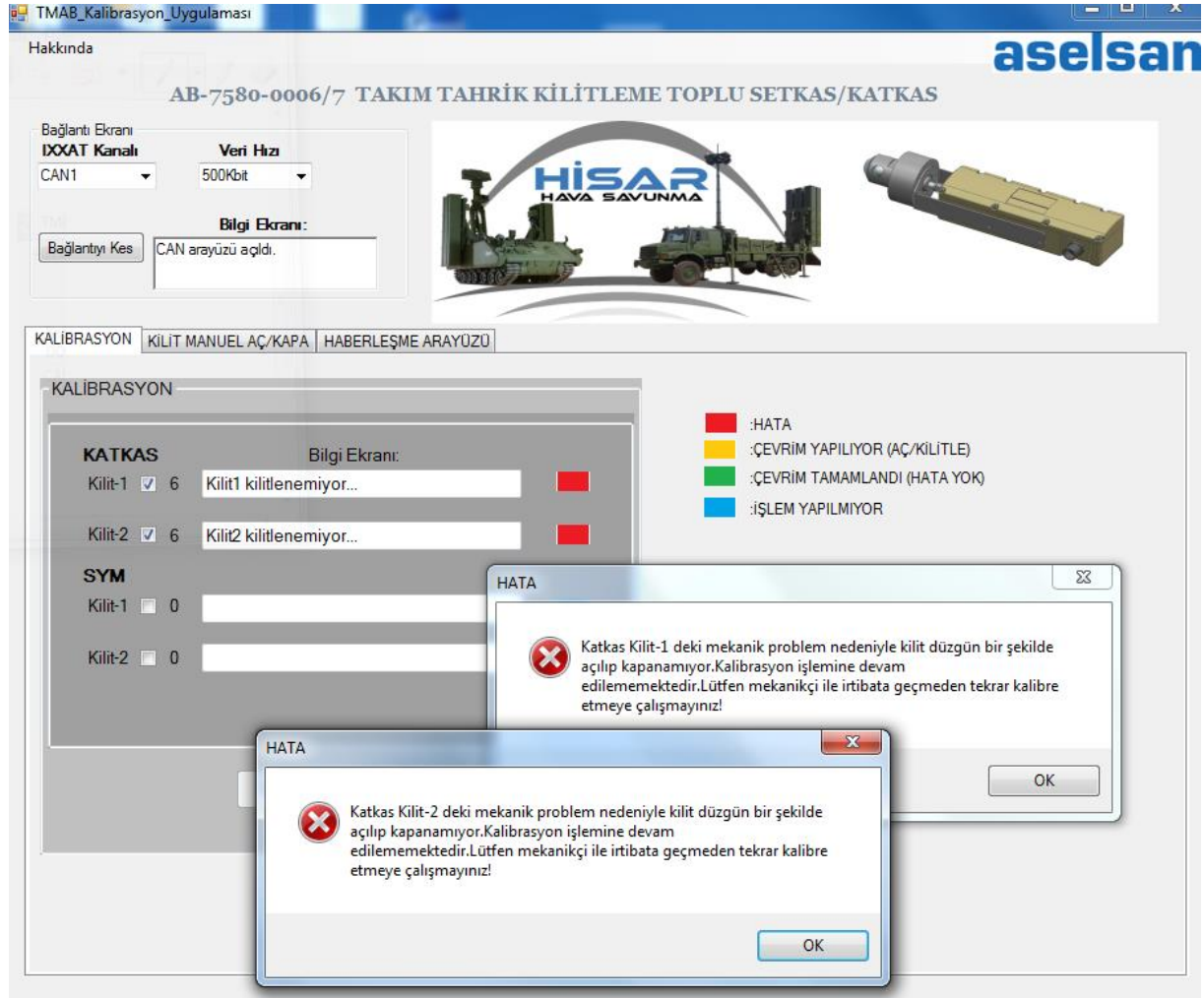
END OF THE PROJECT

This is the last week of my internship.In this week,I completely finished my project and changed a lot of things on the GUI.I also tried to solve some problems in the code.I translated all texts on GUI into Turkish.I will explain the general algorithm and show the final GUI of my project step by step.



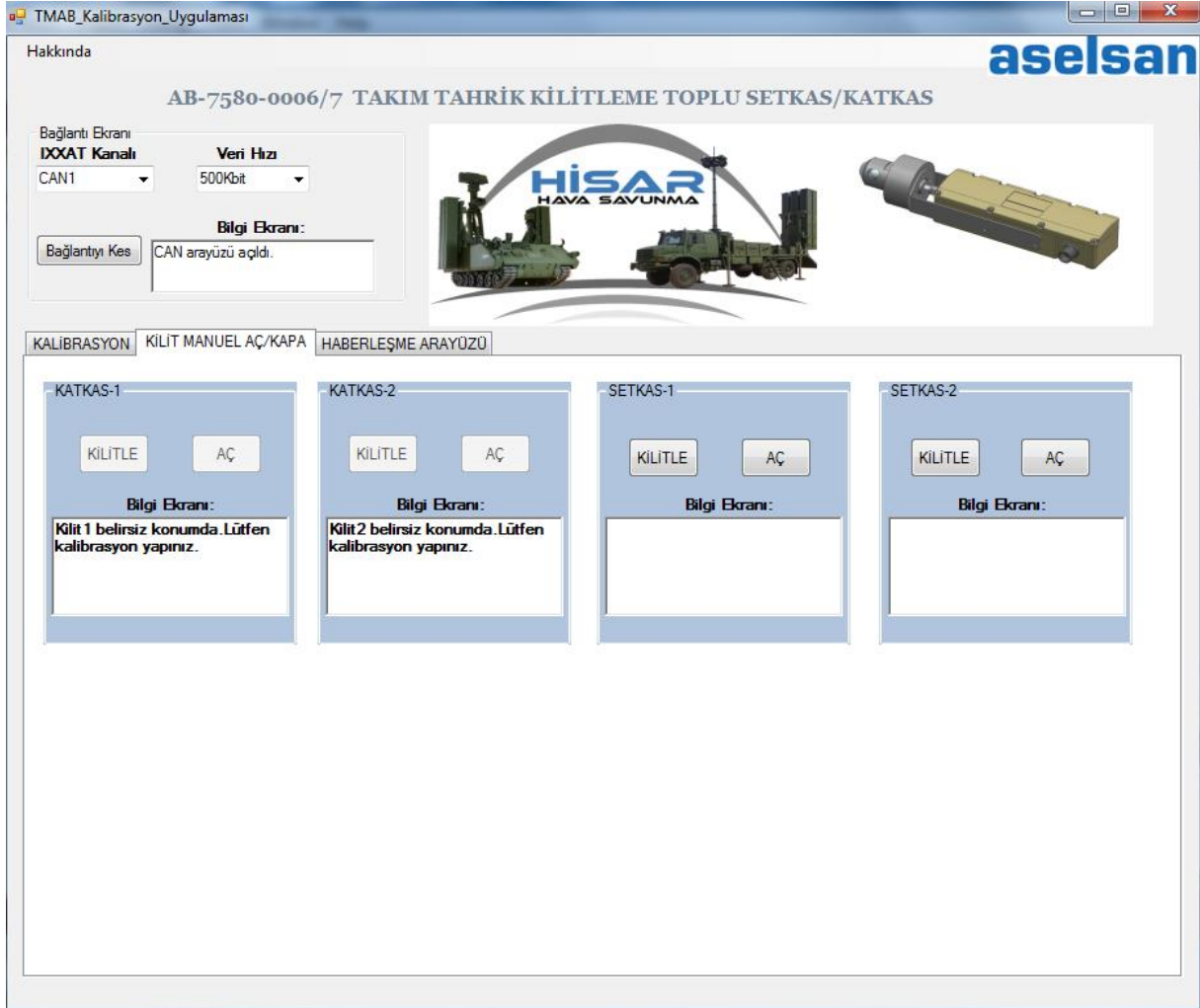
When we open this program, we first see this tab. After choosing IXXAT channel and baud rate, to connect to the CAN interface, we click the 'Bağlan' button and we can see the information about connection process. If the connection is not achieved, we cannot do anything on GUI. Locked position threshold value which is 0x0000h and unlocked position threshold value which is 0xFFFFh are defined as default in the code. If we say a lock is locked, it means its position is greater than the locked threshold value or if we say a lock is unlocked, it means its position is less than the unlocked threshold value. In this situation, the lock seems both locked and unlocked. Therefore, embedded software for KATKAS and SETKAS is rearranged and after that the locks seem unlocked in the beginning as default. After the connection to the CAN interface is achieved, by the thread that I created in the code which is called 'Continuous_Tx_Thread', status inquiry, error inquiry and SYNC messages are sent to all nodes periodically once in a second. Calibration cycle value is set to 7 as default in the code. In the 'KALİBRASYON' screen, before starting to calibrate the locks, we choose at least one lock or choose more locks as shown above and then click the 'Çevrimi Başlat' button. This button can only be clicked once. Each lock can be calibrated once. I used threads for calibration steps, therefore all locks can be locked or unlocked at the same time. After starting the calibration, we can see the last state of the locks in the 'Bilgi Ekrani' simultaneously and calibration cycle number such as 4.

I added some pictures to show the calibration steps visually. If calibration starts and continues, 'YELLOW' picture is shown. If there is an error while the locks are being locked or unlocked, 'RED' picture is shown. If all calibration steps are done properly, 'GREEN' picture is shown. Before starting to calibration, 'BLUE' picture is shown.

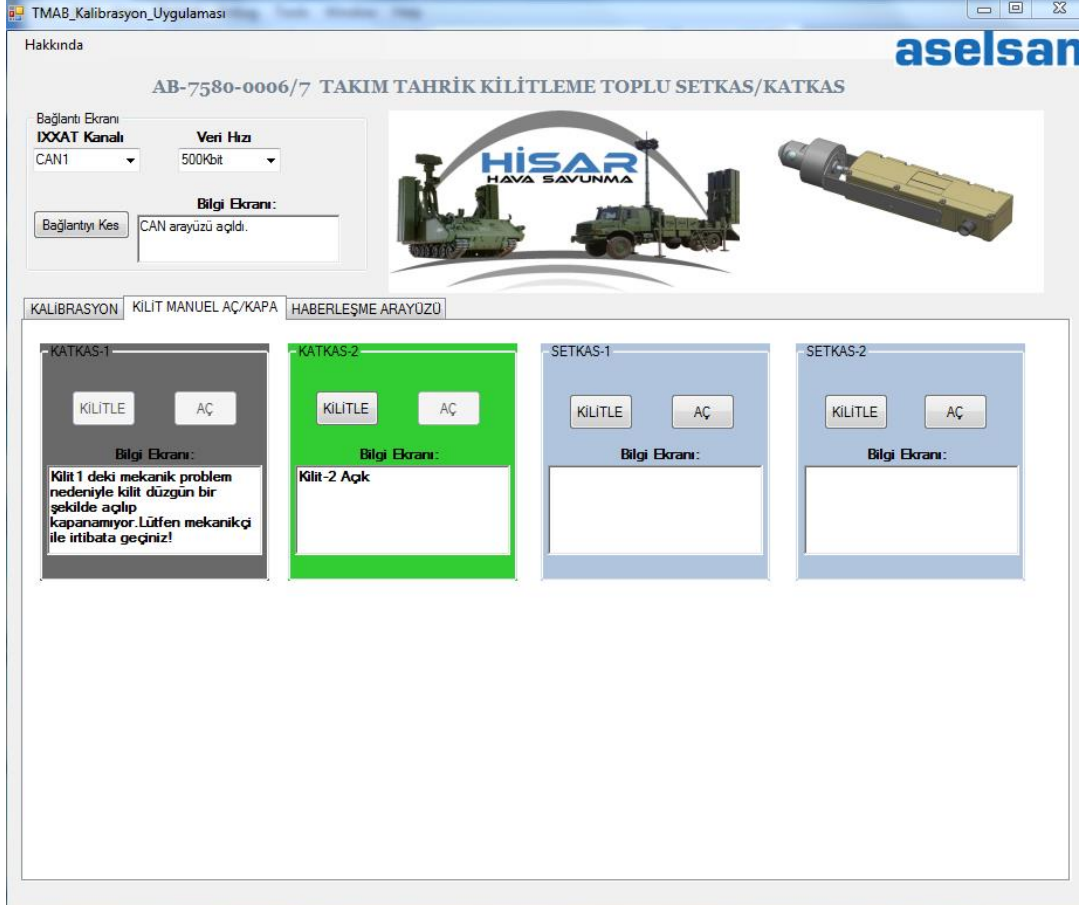
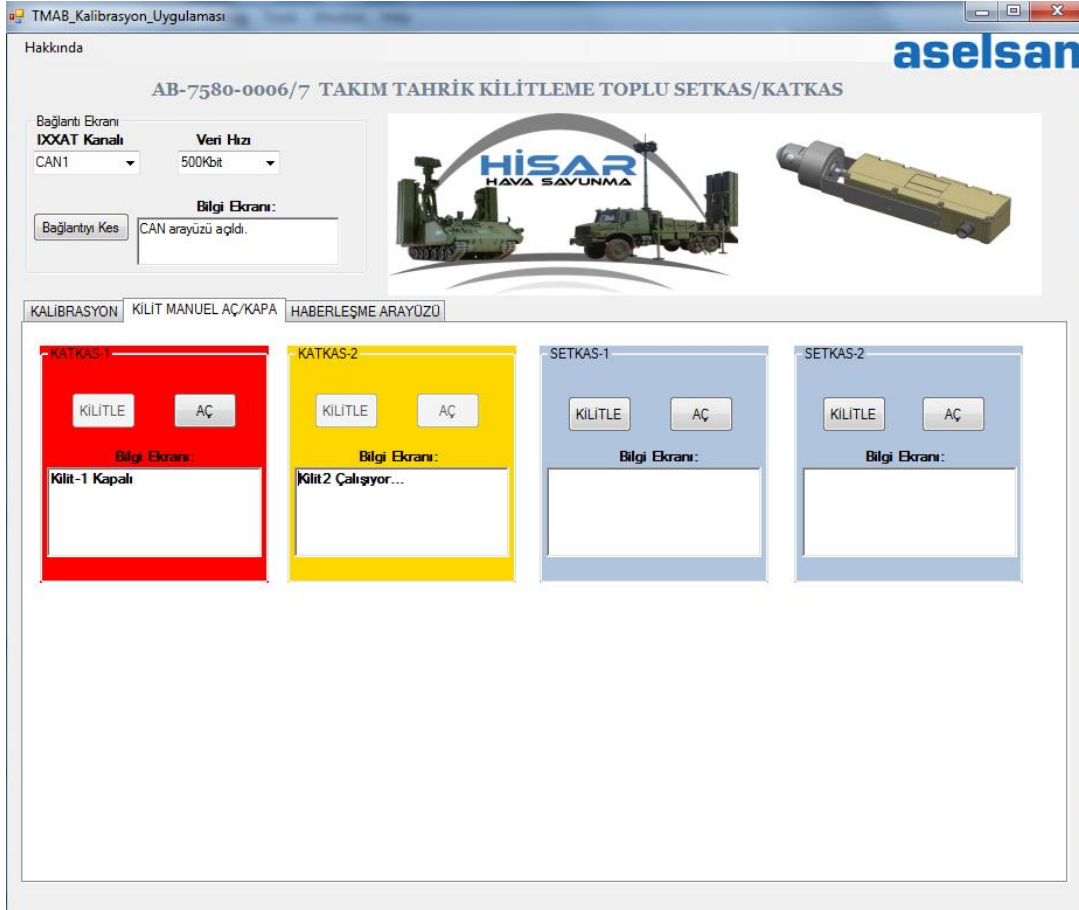


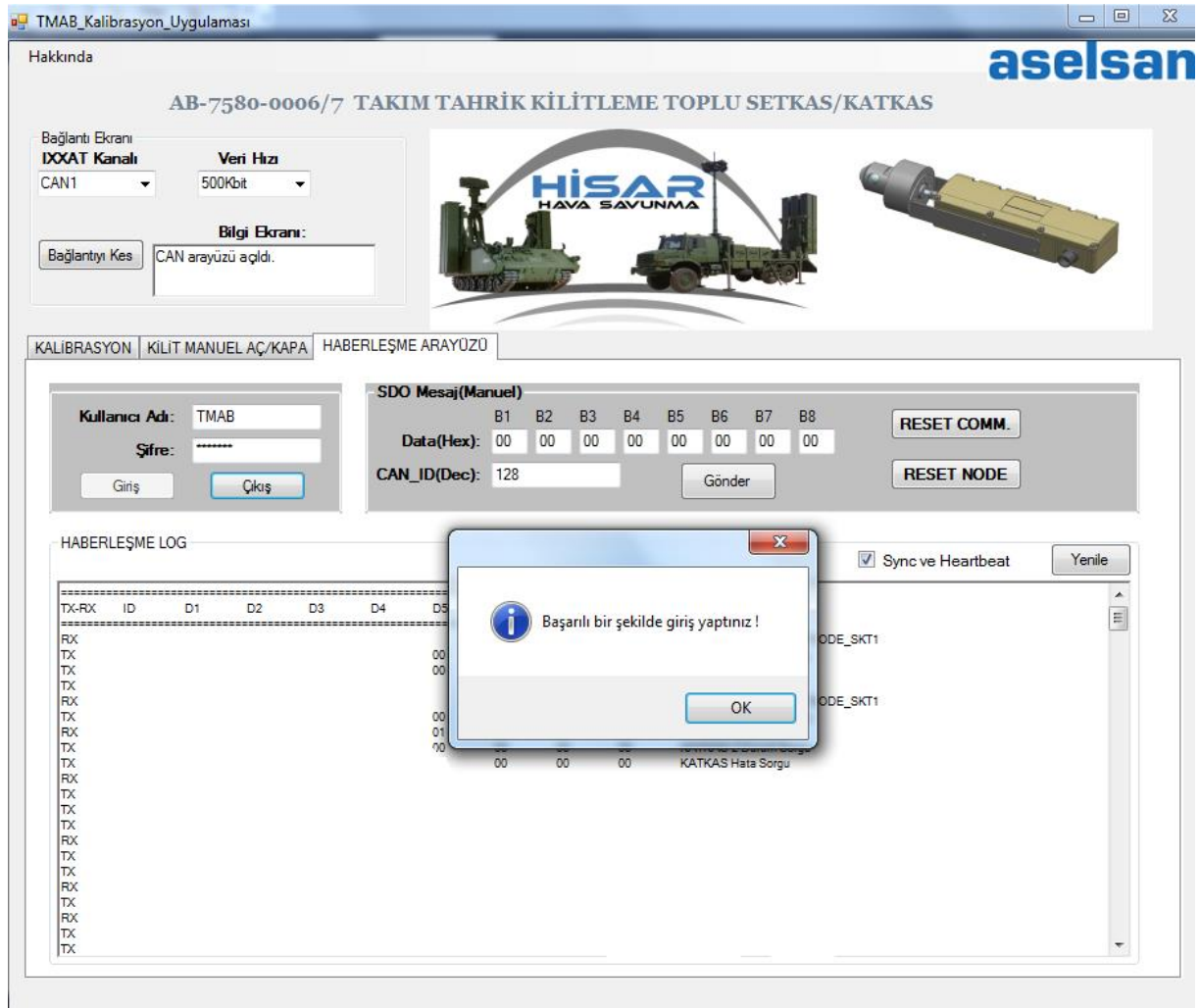
I actually separated calibration code into two parts. First part is to find the minimum and maximum threshold values for the locked and unlocked positions and its cycle value is set to 5 as default. Before calibration, all locks must be in the 'unlocked' position in real because embedded software is arranged for this situation to start the calibration. In this part, default threshold values are sent to SETKAS or KATKAS. I saved each locked and unlocked positions at the end of each cycle in this part. We don't expect any error in this part because the purpose of this part is to indicate the threshold values properly to start the actual calibration in the second part. If the locks are not locked or unlocked properly, a messagebox warns us on the screen and says that there is a mechanical problem as shown above. In the information box, it says 'Kilit kilitlenemiyor...' or 'Kilit açılmıyor...'. The calibration stops in this situation. If the first step of the calibration is achieved, we find the new threshold values of the locks and the program can continue to start the second calibration part. In the second part, the calibration continues until it reaches the indicated total cycle value. I added some error handling algorithms to the code. If the lock position is not greater than the new locked threshold value or the lock position is not less than the new unlocked threshold value, the warning is shown on the screen as shown above and the calibration stops.

After sending lock or unlock commands to all nodes, if we cannot receive locked or unlocked messages from any node after 15 seconds, the program again warns us and calibration stops. There is also a third error situation. If the difference between two consecutive unlocked position values or the difference between locked position value and unlocked position value in the same cycle are greater than we expect, the program again warns us and calibration stops. If there is no problem in all cycles, the calibration finishes.



This tab is used for manual lock or unlock commands. I added 4 groupbox for each locks. Before calibration, we cannot use the buttons 'KİLİTLE' and 'AÇ'. In the information box, it first warns us to calibrate the locks. If the calibration is done correctly, we can lock or unlock the locks separately in this tab. After the correct calibration, all locks are in the unlock position. I used some colors to indicate the operations visually. When we click the 'KİLİTLE' or 'AÇ' button, these colours are automatically shown in the screen depends on the position of the locks. If the lock is locked, background color turns into red. If the lock is unlocked, background color turns into green. If the lock or unlock operations continue, background color turns into yellow. Also we can read the last state of the locks in the 'Bilgi Ekrani'. Even if we calibrate the locks properly, there might be still mechanical problems when we lock or unlock manually. If this happens, the warning is written in the 'Bilgi Ekrani' and we cannot click both 'KİLİTLE' and 'AÇ' buttons. Background color turns into grey. These situations are shown in the next page.





This tab is the communication interface of my project. I mentioned the functions of the tab in the previous weeks. I only added one thing to the tab. To open 'SDO Mesaj(Manuel)' groupbox, we have to enter the password and user name. If we enter correctly, 'SDO Mesaj(Manuel)' groupbox opens and we can manually send messages. By doing so, I prevented to be sent wrong messages by any person who doesn't know how to use this program.

4) CONCLUSION

I had a great experience during my internship. Hour of work starts at 7.30 am and finishes at 4.30 pm in ASELSAN. It was very difficult to work early for me in the beginning and then I got used to wake up and to work early. I realized that I was very productive in early hours and also I could set aside time for myself because hour of work finishes early. There are flexible working hours here. There are so many projects in different departments. Everyone has one or more projects to work on it. If you finish your project, then you move to other different projects. Therefore, engineers have a chance to learn different informations and always improve themselves in ASELSAN.

They gave me a project and I didn't know where to start. I learned how to solve problems and how to find informations that are related to my work. Before my internship, I used to try to learn all informations about a work before starting and it takes a lot of time to start a work. But, after my internship, I realized that If you start a work directly, you already start to learn and your work finishes faster. During my project, I observed that there is no work that cannot be finished or solved. The employees are very helpful and happy with their jobs. It was very fast to learn the work for me. Because there are really good engineers to ask the questions and they always helped me to learn my job. The employees are always concentrate on their jobs during work hours and they are in very good communication with each other. My working area was spacious, light, clean and high-ceiled. This is actually very important for me to work more productive. There are also free breakfast, lunch and transportation opportunities in ASELSAN. It was also an advantage for all employees and me.

5) **REFERENCES**

<http://www.aselsan.com.tr/tr-tr/Sayfalar/default.aspx>

<https://www.theworkplacedepot.co.uk/safety-signs-labels>

<http://www.ni.com/white-paper/14162/en/>

<http://www.onlineteachinghub.com/c-data-types-and-variables/>

<http://www.completecsharp tutorial.com/basic/variables-datatypes/>