WAYNE STATE UNIVERSITY

**APRIL 23, 2020**

# EtherCAT Weather Monitoring Station


Class: ECE 5620 Winter 2020

Professor: Dr. Syed Mahmud

Group 2: Mostafa AlNaimi
Matthew Hagan
Brian Yousif-Dickow



Wayne State University College of Engineering • 313-577-3780

5050 Anthony Wayne Dr. • Detroit, Michigan 48202 • https://engineering.wayne.edu/

# Table of Contents

# Abstract

This project is based on designing a weather monitoring station controlled by a master communicating with two slaves over an EtherCAT network. Our desired task is to gather information from temperature, humidity, air pressure, and ambient light sensors which are connected to our slaves. Based on the values transmitted to the master, various LED's will turn on accordingly and a display will also show the values of each sensor.

# Executive Summary

Our aim is to create a weather station that creates an easy and convenient solution to checking the weather by looking at the station or any device that is connected to the weather station. This device can be placed outside a room with a window and the user can easily look at the LED's, or the display, and know what to expect when they leave the house. The completed project can be outlined in three categories that allow the real time operating system to perform with minimal latency:

- Real Time Communication between the master, slaves, and sensors.
- The Master receiving the real time data from the slaves and sensors and updating the LEDs as well as the display.
- The system continuously looping and checking to be sure all parts stay online and act appropriately.

Overall, this project was successful in demonstrating a successful operation of a weather station using EtherCAT. They system was able to update accurately accordingly to changes in the sensor inputs due to environmental changes.

The first difficulty that our team needed to accomplish was to develop an EtherCAT network that the master and slave devices would be able to operate on. We used an EtheCAT communication network which allowed the master to send and receive data on the fly with the slaves and sensors as EtherCAT is a full-duplex system.

The communication is originated from the master and is sent to the slaves and the slaves acknowledge, update, and continue the communication cycle. The master then receives the updates (every 1ms) and updates the LED's and output values of each sensor accordingly.

The master implements three threads with the highest priority being the communication thread which loops every 5ms and assures the master maintains communication cycles to transmit and receive data with the slaves and sensors on the network. The second thread is a lower priority thread which loops every 10ms and checks for errors in the state machine and in the slave application. The third thread, which is the lowest priority thread, is a printing thread which updates the readings of all of the sensors every 100ms. These threads assure that the communication is intact, and the latency is still within tolerance as the system is being continuously updated.

# Introduction

**Overview:**

Our group created a weather monitoring station controlled by a master communicating with two slaves over an EtherCAT network. EtherCAT (Ethernet for Control Automation Technology) is a real-time Industrial Ethernet technology originally developed by Beckhoff Automation. EtherCAT is a master/slave protocol that's suitable for hard and soft real-time requirements in automation technology, in test and measurement and many other applications. The master gathers sensor data from each slave regarding the temperature, humidity, air pressure, and current ambient light. Based on the sensor data the Master turns on various LED's at each slave that correspond to the sensor data provided.

**Scope of Work:**

The largest part of the project consisted of developing the EtherCAT protocol to map out the linear topology of the network. This consisted of creating an object dictionary that mapped the signal plane from each slave to the master. The system updates the object dictionary every 1ms which provides the master with the information to run the algorithms in the background and trigger responses to the weather information.

The entire system was written in C and is ran on an RT Linux operating system which is provided via TI's website. As for the hardware implementation, the master runs on a TI Eval board AM335x processor-based ICE EVM TMDSICE3359. The master communicates with two slaves implemented on theXMC4800 which is a microcontroller built on the ARM Cortex-M4 processors. We evaluated the boards and connected various sensors over analog and SPI communication. Each slave has an implementation that accesses various registers to retrieve the data and scale it before sending it to the master over the network.

# Background

There are many different types of weather stations that are on the market that vary greatly in features, cost, and overall use.  We highlighted a couple of inventor's patents that aligns with what we would like to accomplish, in regard to creating an EtherCAT Weather Station.

## PATENTS

**Patent: CN206594319U**

**Inventor: 叶小岭程恩路杨星熊雄郝曼胡全辉**

**Year: 2017**

**Application: CN201720318588.4U**

**Application filed by: 南京信息工程大学**

**Application Date: 3-29-2017**

**Application Granted: 10-27-2017**

### Overview:

This patent belongs to a similar product in China. This patent covers a portable weather station that uses a STM32 microcontroller and GPS to locate the position of where the conditions are being taken. It then can be hooked up to a computer or smartphone to display the measurements.  It includes temperature, humidity, rainfall, air pressure, wind speed, wind direction, and intensity of illumination.

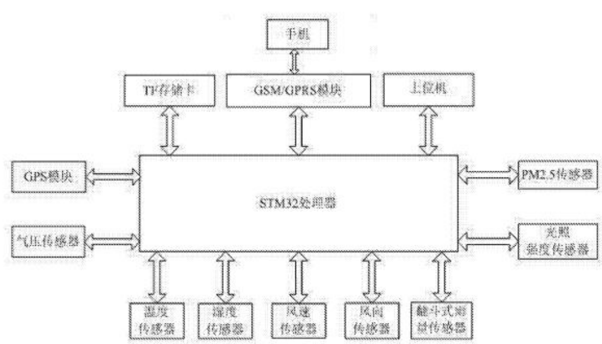


图1

**Patent: WO2010061112A1**

**Inventor:** Jean-Philippe Chazarin

**Year: 2017**

**Application: CN201720318588.4U**

**Application filed by: Institut De Recherche Pour Le Developpement**

**Application Date: 11-23-2009**

**Application Granted: 6-3-2010**

## Overview:

The invention relates to a portable weather station, including a structure and weather sensors such as humidity, pressure, temperature, cloud cover, and wind speed sensors. These sensors are arranged on the structure using a connector having a plurality of connector pins to which the sensors are connected in a predetermined manner, in order to provide an electrical interconnection between the sensors and the controller. The controller is suitable for storing data received from all sensors.

FIG. 1

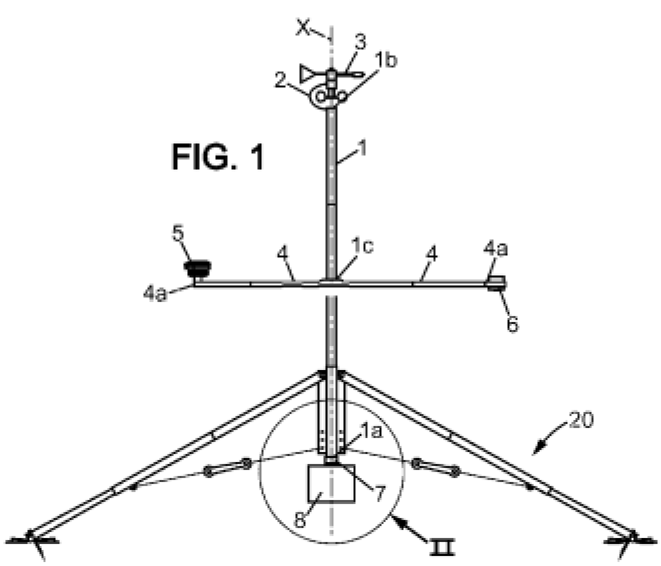


*FIGURE 2. STM32 Patent Block Diagram*

# Hardware Design Theory

The operation and design theory section analyzes the system hardware and assembly. The considerations for this project were with respect to a working prototype; therefore, there has been minimal consideration for the device working in a harsh environment. To protect the entire system from environmental dangers, the system would be placed in an weather resistant enclosure with properly seated cable connections for each sensor.

The overview of the system can be seen below. The current weather station design consists of four sensors that read temperature, ambient light, humidity, and environmental pressure. The system gathers sensory information from the environment and transmits this information to the slave. The slave will then digitize the analog signal and transmit the information over the ethernet cables. The master gathers the information and processes commands to the slaves.

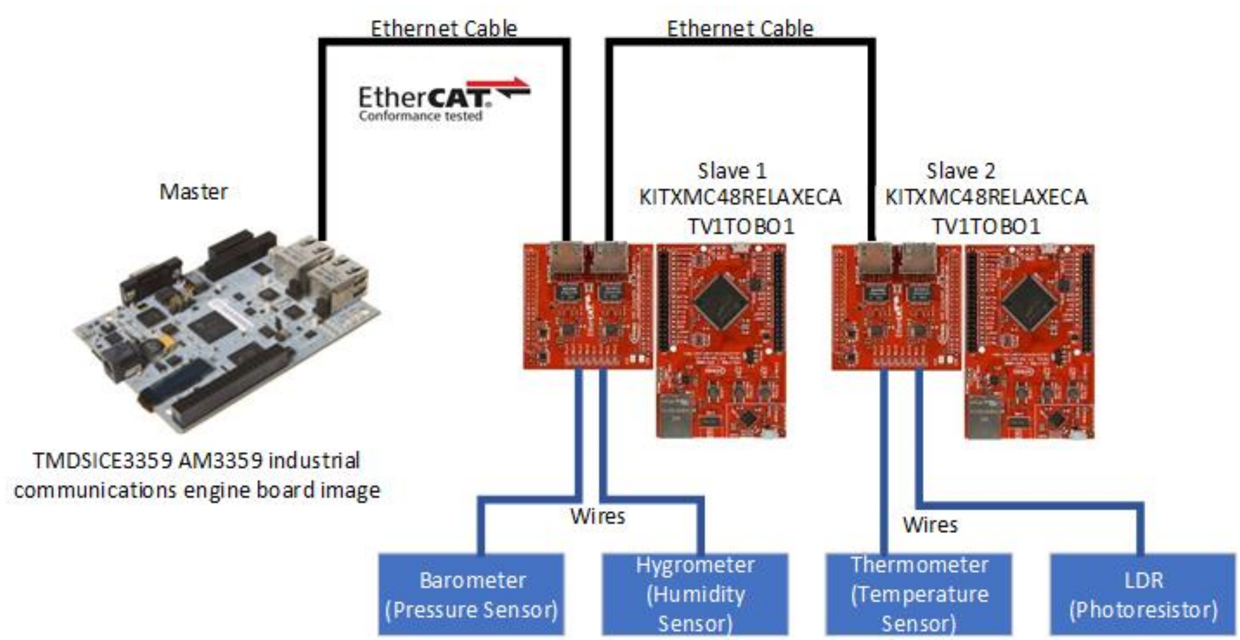


*FIGURE 3. Hardware Overview*

# EtherCAT Master – TMDSICE3359

The TMDSICE3359 is an industrial communications board with a AM3359 ARM Cortex-A8 processor created by Texas Instruments. It is equipped with DDR3, NOR Flash and SPI Flash. It has an OLED display and uses the TPS65910 chip for power management. It supports connectivity of CANOpen, Ethernet, EtherCAT, SPI, UART, JTAG and other protocols.

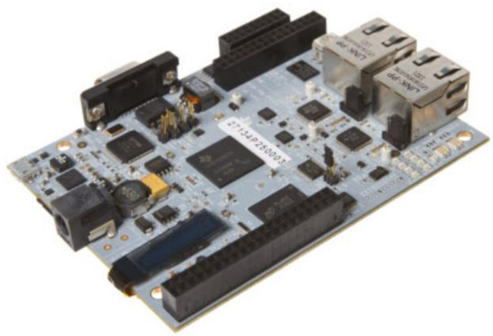

*FIGURE 4. TMDSICE3359 EtherCAT Master [TI]*

The master is responsible for starting the communication to the slaves to retrieve all sensor information. The master communicates to the slaves through EtherCAT communication and reports all values to the computer over UART. The block diagram for the board can be seen in the figuree below:

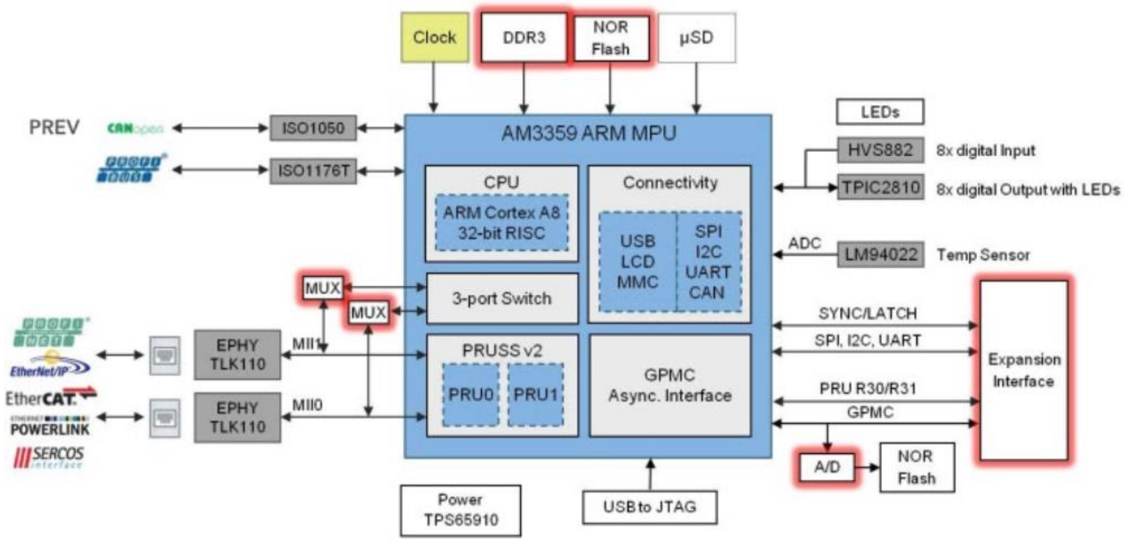


*FIGURE 5. TMDSICE3359 Block Diagram [TI]*

# EtherCAT Slaves – XMC4800

The XMC4800 Relax EtherCAT Kit from Infineon features an XMC4800-F144 microcontroller based ARM Cortex, integrated EtherCAT Slave Controller. It features an Arduino compatible 3.3v pinout, real time clock crystal, quad SPI Flash, CAN node and EtherCAT node.

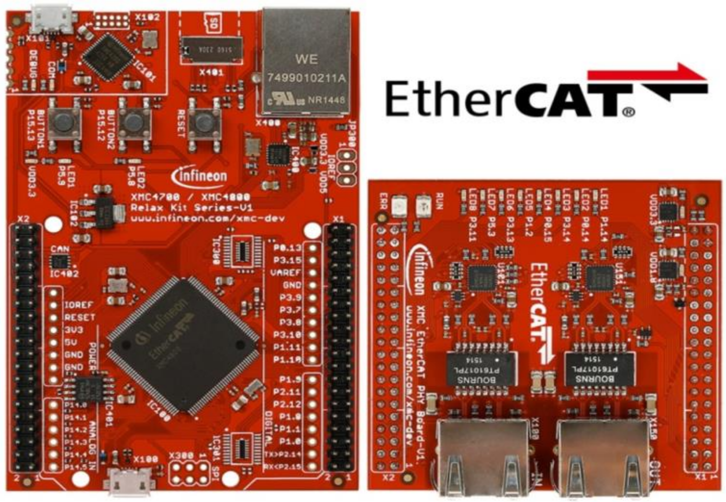


*FIGURE 6. XMC4800 EtherCAT Slave*

The slave is responsible in retrieving and sending all sensor information to the master. It is linked to the master via wired ethernet and to our sensors using the analog pins.

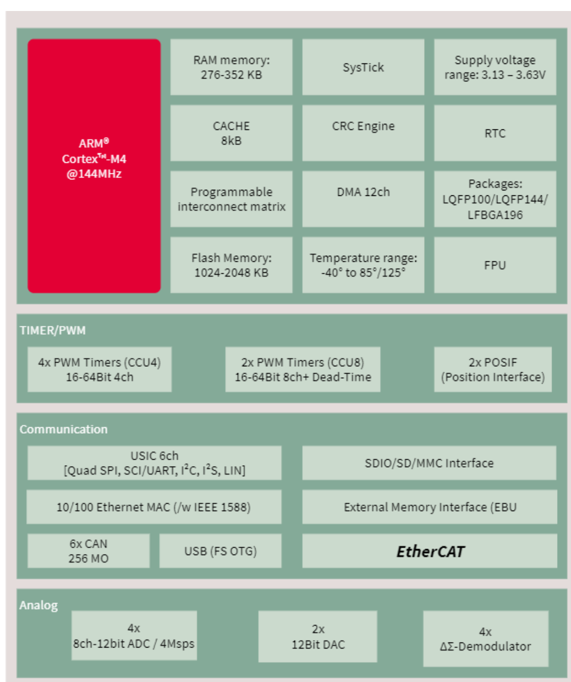


*FIGURE 7. XMC4800 Block Diagram – Infineon*

To bring in the analog signals on slaves one and two. Two sensors were connected to each slave. On both slaves' analog pins p14.3 and p14.5 were used along with the 5V and ground signals.

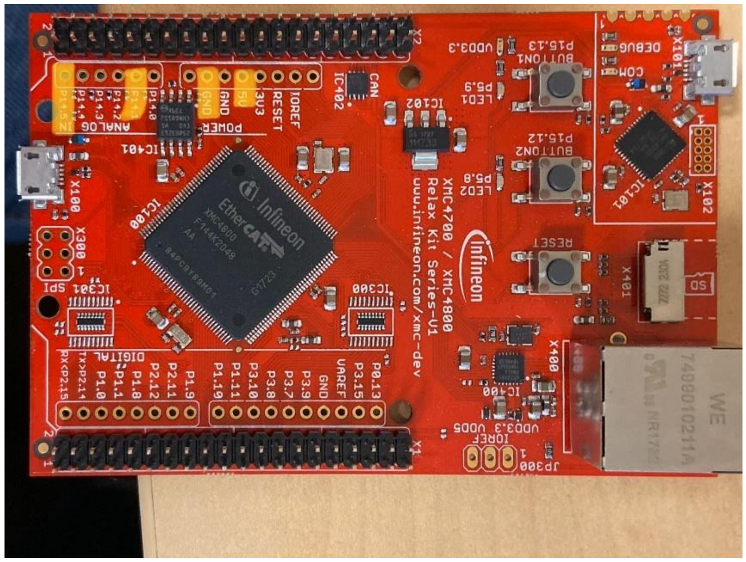

*FIGURE 8. XMC4800 Pins Used*

## Temperature Sensor - Thermistor

To measure the environmental temperature a thermistor was used as the transducer. A thermistor is a device that is responsive to temperature changes in a predictable manner. The basic principle is that the semiconductor's resistance changes greatly in response to minor temperature changes. In short, as temperature increases the resistance decreases. However, these changes occur in a nonlinear fashion as the graph depicts in the figure below:

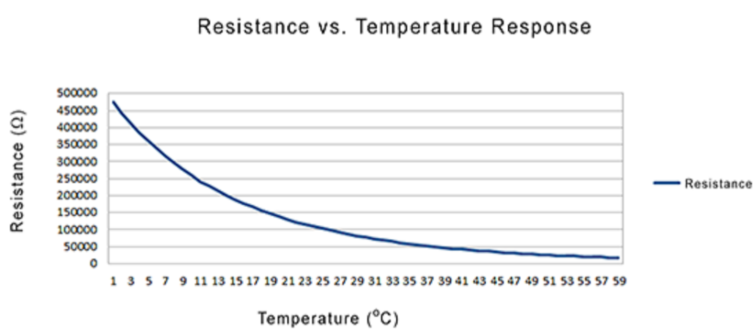


*FIGURE 9. [Thermistor]*

To accommodate for the nonlinearity of the change in resistance to temperature we used the Steinhart-Hart equation to more closely relate the temperature to the resistance change. A table of values is another method for scaling the signal.

**Steinhart - Hart Equation $1/T = A+B(LnR)+C(LnR)^3$**

**For a 10 kohm thermistor, the value of constants A, B and C are:**

**A = 0.001125308852122**
**B = 0.000234711863267**
**C = 0.000000085663516**

*FIGURE 10. Steinhart- Hart Equation for Thermistor Modeling with Constants for a 10kohm Thermistor*

*FIGURE 11. 10kohm Thermistor*

The 10kohm thermistor used has an accuracy of +/- 1 degree Celsius and a range of 0-70 degrees Celsius. The circuit design to condition the incoming signal is similar to a voltage divider. As the resistance drops very low the 10kohm resistor protects the circuit from shorting to ground.

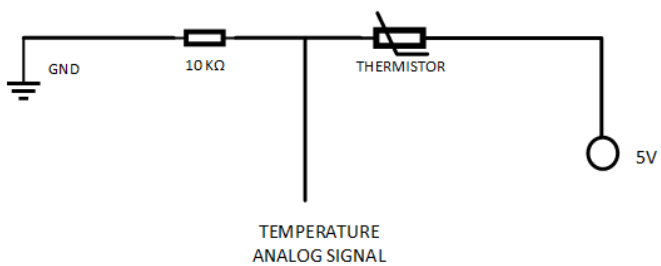


*FIGURE 12. Thermistor Circuit Design*

# Light Sensor – Photoresistor

A photoresistor also known as a light dependent resistor was used as our ambient light sensor. A photoresistor is a semiconductor that responds to the ambient light in the environment. The more photons the sensor absorbs the less resistant the device becomes.

*FIGURE 13. Photoresistor*

For the application the a 5V source was driven across the photoresistor with a 10kohm balance resistor. The photoresistor used has a resistance range between 200k ohms to 300 ohms when sensing minimal and max light respectively. A depiction of the photoresistor circuit can be seen in the figure below with a table of common environments with a predicted response of the circuit:

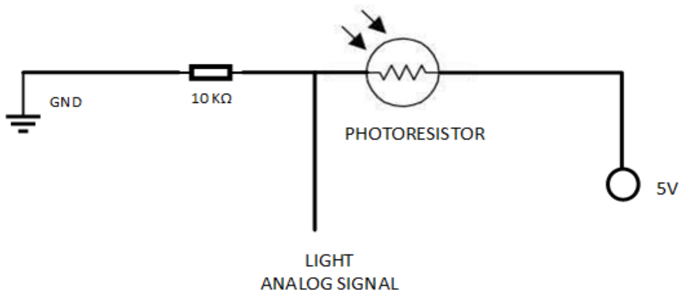


*FIGURE 14. Photoresistor Circuitry*

| Ambient light like... | Ambient light (lux) | Photocell resistance (Ω) | LDR + R (Ω) | Current thru LDR +R | Voltage across R |
|---|---|---|---|---|---|
| Dim hallway | 0.1 lux | 600KΩ | 610 KΩ | 0.008 mA | 0.1 V |
| Moonlit night | 1 lux | 70 KΩ | 80 KΩ | 0.07 mA | 0.6 V |
| Dark room | 10 lux | 10 KΩ | 20 KΩ | 0.25 mA | 2.5 V |
| Dark overcast day / Bright room | 100 lux | 1.5 KΩ | 11.5 KΩ | 0.43 mA | 4.3 V |
| Overcast day | 1000 lux | 300 Ω | 10.03 KΩ | 0.5 mA | 5V |

*FIGURE 15. Example Values of Photoresistor Response*

# Humidity Sensor – HIH-4000

The HIH-4000 is a through hole board mount humidity sensor produced by Honeywell. The analog sensor operates between 4-5.8v with a typical current draw of 200uA. It measures RH (Relative Humidity) between 0% - 100% with an accuracy of +/- 3.5%. The analog Vout signal ranges from .826V to 5V across an 80 kohm resistor. The resistor is in place for scaling the output signal and disapating energy from the circuit.

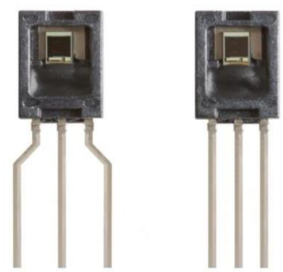

*FIGURE 16. HIH 4000 Humidity Sensor*

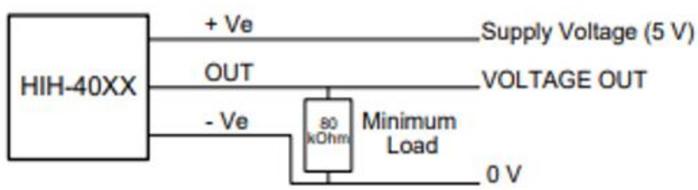


*FIGURE 17. HIH 4000 Humidity Sensor Circuitry*

# Pressure Sensor – KP236N6165XTMA1

The KP236N6165XTMA1 by Infineon Technologies is a board mounted  miniaturized analog barometric air pressure sensor.  The calibrated transfer function converts a pressure range of 60 kPa to 165 kPa (+/- 1kPa)  into a voltage range of .2V to 4.8V. The sensor is able to operate in a temperature range of  -40 to 125 degrees Celsius. The outline of the control circuitry and pin layout is denoted below.

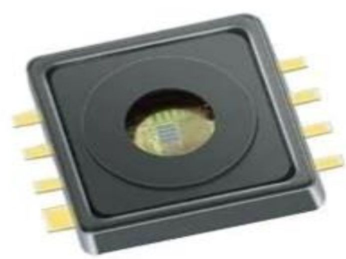

*FIGURE 18. KP236N6165XTMA1 Pressure Sensor*

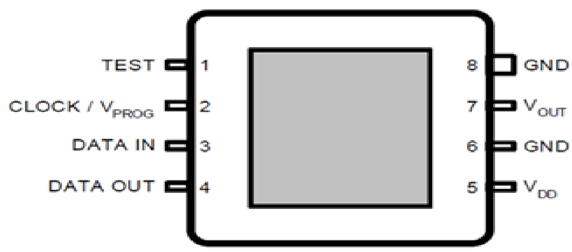


*FIGURE 19. KP236N6165XTMA1 Pressure Sensor Pin Layout*

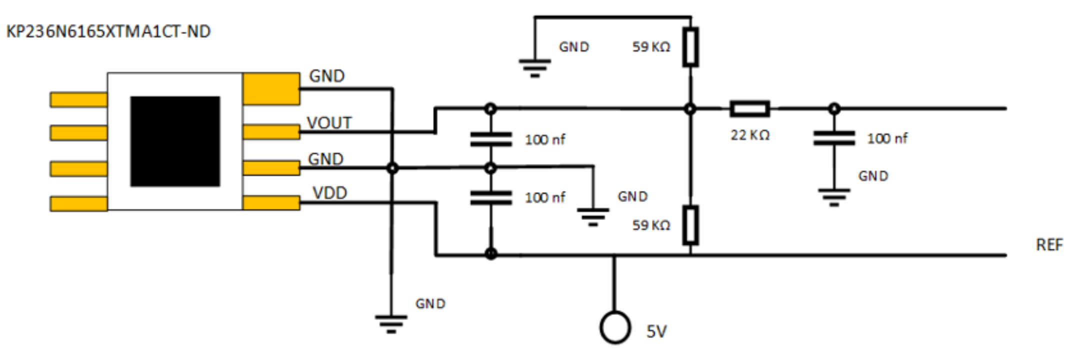


*FIGURE 20. Pressure Sensor Signal Circuitry*

# Hardware Assembly

Putting it all together the sensor hardware interfaces to the slaves in the following diagram shown below:

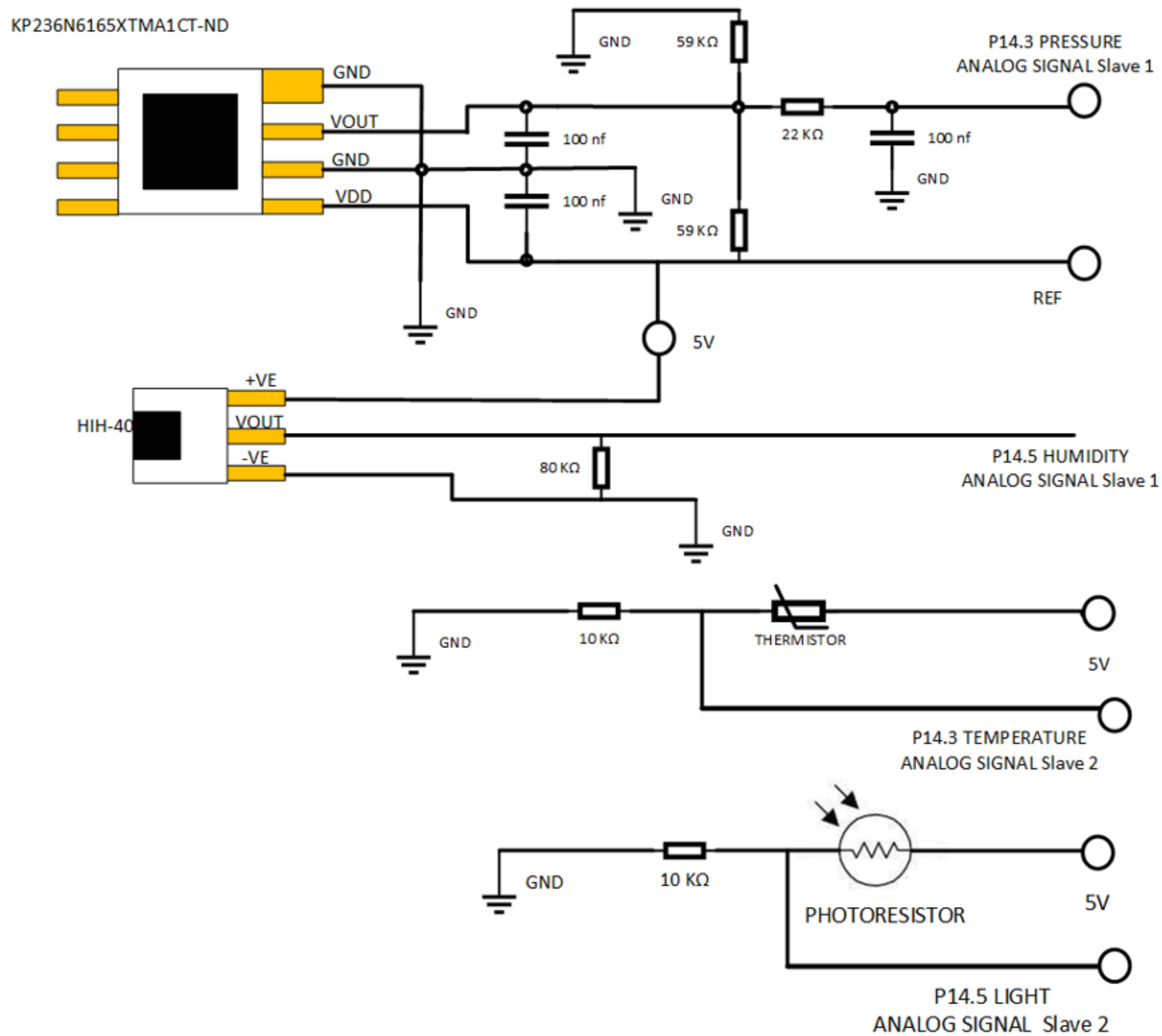


*FIGURE 21. Prototype Board Layout*

# Software Design Theory

The software design theory section analyzes the system's software principles and properties. This section will primarily outline the details of EtherCAT and the weather station application.

## EtherCAT Background

EtherCAT is a real-time Ethernet-based fieldbus system originally developed by Beckhoff Automation in 2003. The name EtherCAT is actually an acronym for **Ethernet** for **Control Automation Technology**. The industrial network was designed around providing a low jitter and accurate synchronizing system that can accommodate short cycle times (less than 100us) at low hardware cost. Another major benefit is that its master to slave communication is transmitted over regular ethernet cables with RJ-45 connectors which are very common and relatively inexpensive and robust. The hardware and software achitecture make EtherCAT a suitable solution for both hard and soft real-time requirements in automation technology. Primarily EtherCAT is used in test, measurement, and Servo Drive applications throughout various industries.

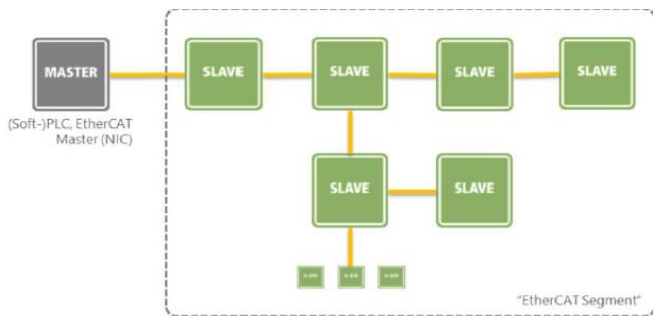


*FIGURE 22. Possible EtherCAT Network*

## EtherCAT Principles

EtherCAT is an internationally standardized open technology for a Master-Slave protocol. The unique way EtherCAT  processes frames, makes it the fastest industrial Ethernet technology on the market. It boasts accurate synchronization as jitter is less than 1us. These speeds are 100x faster than traditional CAN and 20x faster than CAN FD coming in at 100mbps.

These speeds are only acheivable by processing the data **on the fly** by the quick switching FPGA's in thee hardware of the device. EtherCAT is also Full-Duplex meaning information can flow BOTH ways at the SAME time. Because the system is based on Ethernet all inherent collision avoidance standards CDMS/CD are built in through the IEE 802.3 standard.

Another benefit to EtherCAT is that it can support up to 65535 nodes per segment, meaning one master can handle that many slaves.

## EtherCAT Framework

The EtherCAT frame contains the frame header and one or more datagrams. The network embeds its payload into a standard Ethernet frame. The datagram header indicates what type of access the master device would like to execute: read, write, or read-write. To access to a specific slave device, the system can use direct addressing, or access to multiple slave devices through implicit addressing, or a broadcast frame to all slave devices.

Implicit addressing is used for the cyclical exchange of process data. Each datagram addresses a specific part of the process image in the EtherCAT segment, for which 4 Gbytes of address space is allocated.

During the network startup, each slave device is assigned one or more addresses in the global address space. If multiple slave devices are assigned addresses in the same area, they can all be addressed with a single datagram. A block diagram of the EtherCAT frames is shown in the figure below.

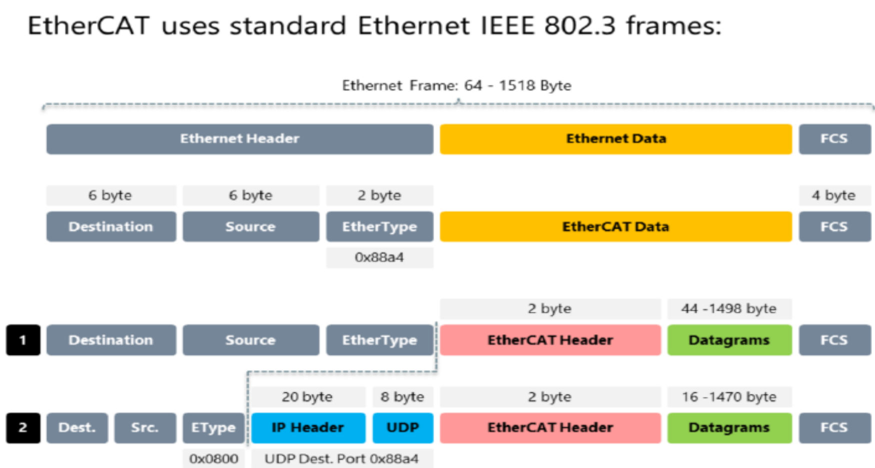


*FIGURE 23. EtherCAT Framework*

# EtherCAT Master – Block Diagram

- Establish NIC socket
- Configure Slave(s)
- Operates the network
- Send Cyclic & acyclic data
- State Machine
- Provides an interface

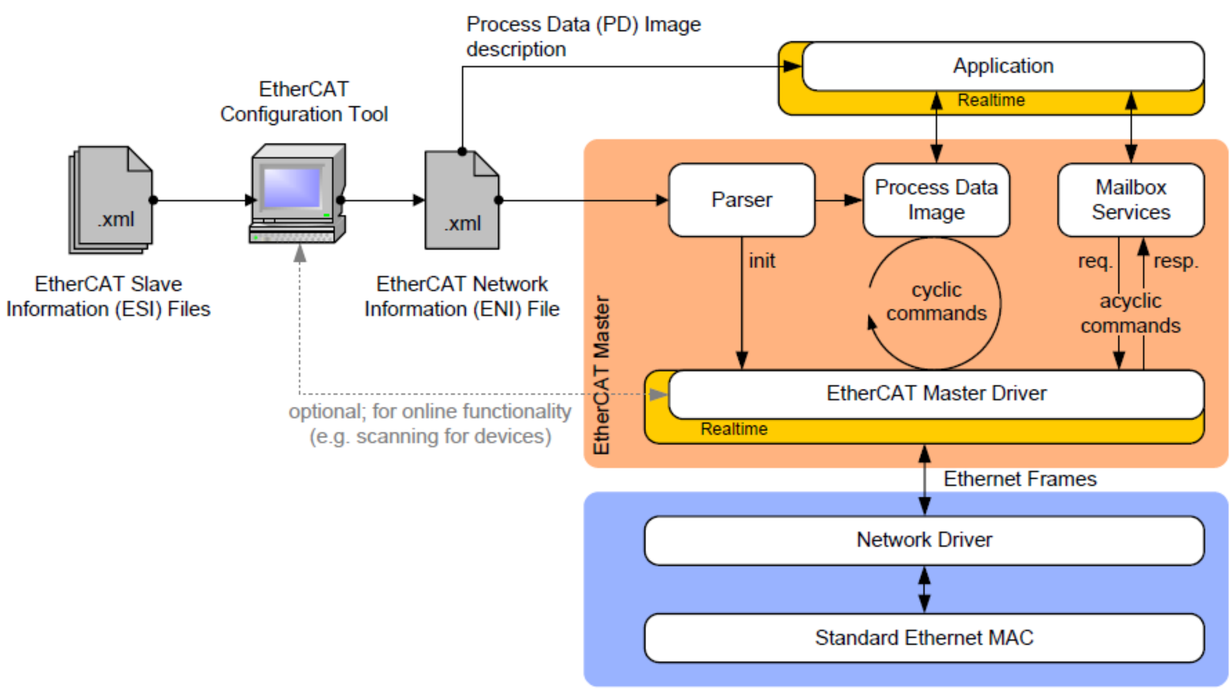


*FIGURE 24. EtherCAT Master Block Diagram*

# Network Topologies

One major benefit of using an EtherCAT network is that the developer is not limited to one network topology. For the weather station a linear topology was used; however, there are many other mesh networks options that can be supported to fit any application. A few common topologies are shown below:

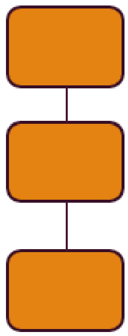

*FIGURE 25. Linear or Line Network*

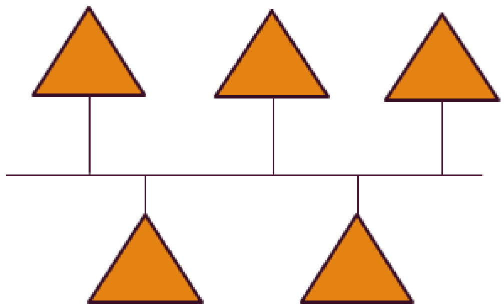

*FIGURE 26. Bus Network*

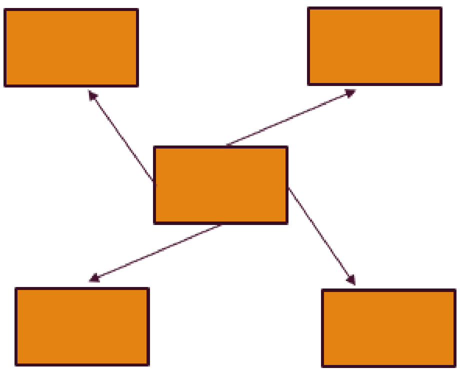

*FIGURE 27. Star Network*

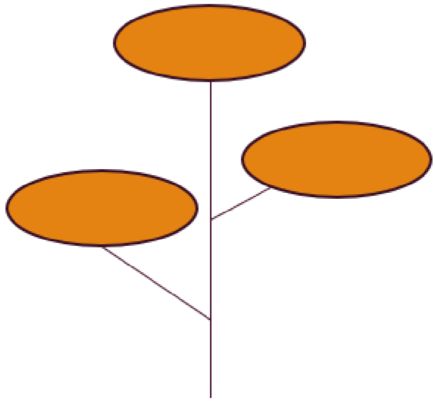

*FIGURE 28. Tree Network*

## Stack Delay

When comparing Ethernet and EtherCAT there is a big difference when it comes to the stack delay. In common industrial ethernet protocols the stacks are fairly large. Because of the size, it takes a long time to process large software stacks for small embedded CPUs. In contrast, EtherCAT limits the stack to a maximum stack size to around 70KB where digital I/O is zero. The figure below gives common stack times for EtherNet/IP and EtherCAT communication protocols.

| | EtherNet/IP | EtherCAT |
|---|---|---|
| Stack Time | Ethernet/IP | EtherCAT |
| Average | 1.89 ms | 0.11 ms |
| Max. | 2.96 ms | 0.18 ms |
| Min. | 1.23 ms | 0.05 ms |

*FIGURE 29. Industrial Communication Stack Times*

## State Machine

EtherCAT uses four main states and 1 optional state. The EtherCAT Master requests all slaves to transition their states during initialization procedure before it can move to the Pre-Operational state. Once the system has move to the PreOP the state machine can start up the Safe-Operational. If no errors have occurred the system will move into the Operational state.

State Machine Technique: Init → PreOP → SafeOP → OP.

**Init** = Initialization; **PreOP** = Pre-Operational; **SafeOP** = Safe-operational; **OP** = Operational;

**Boot** = Bootstrap

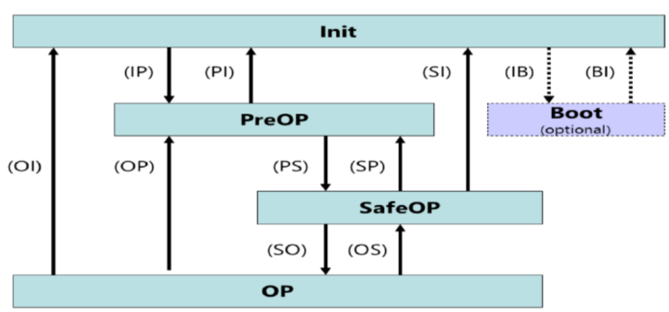


*FIGURE 30. EtherCAT State Machine*

# Working Counter

The Working Counter is used as a safe-guard or redundancy to track frames and validate their success. Each EtherCAT Datagram that is sent, includes a Working Counter (WKC). The WKC tracks of all operational slaves in the network and increments after each successful access. The Master will then compare WKC against the expected value to validate successful accesses. The figures below depict the outline of an EtherCAT Datagram and how the WKC increments based on the command from the Master

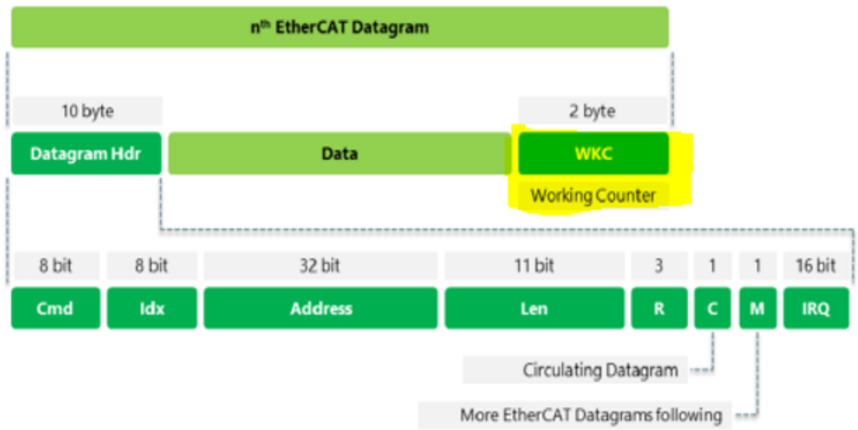


*FIGURE 31. EtherCAT Datagram Outline*

| Command | | Increment |
|---|---|---|
| Read | xxRD | +1 |
| Write | xxWR | +1 |
| Read Write<br>→ Read<br>→ Write | xxRW | <br>+1<br>+2 |

*FIGURE 32. Working Counter Incrementation Based on Command*

# Software Implementation

EtherCAT can be implement in most environments. There are only a few restrictions on the types of chips used for the Ethernet physical layer; however, the overall implementation can be summed in the following bullet points.

- EtherCAT uses **RAW socket** to send packets over an EtherCAT thread.
- EtherCAT is OS independent.
- The Master does not require any special drivers or controllers.

- Slaves do not have to be specialized controllers, but rather designed with **ESC (EtherCAT Slave Controller)** stack.
- Only the Master initiates packets to broadcast to all slaves in the network.

```
/* we use RAW packet socket, with packet type ETH_P_ECAT */
*psock = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ECAT));

timeout.tv_sec =  0;
timeout.tv_usec = 1;
r = setsockopt(*psock, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout));
r = setsockopt(*psock, SOL_SOCKET, SO_SNDTIMEO, &timeout, sizeof(timeout));
i = 1;
r = setsockopt(*psock, SOL_SOCKET, SO_DONTROUTE, &i, sizeof(i));
```

*FIGURE 33. Master Software Implementation*

EtherCAT uses Cyclic and Acyclic messages to communicate with the slaves. There are two forms of messages: cyclic and acyclic.

PDO (Process Data Object) - Cyclic messages.
SDO (Service Data Object) - Acyclic (event driven) messages

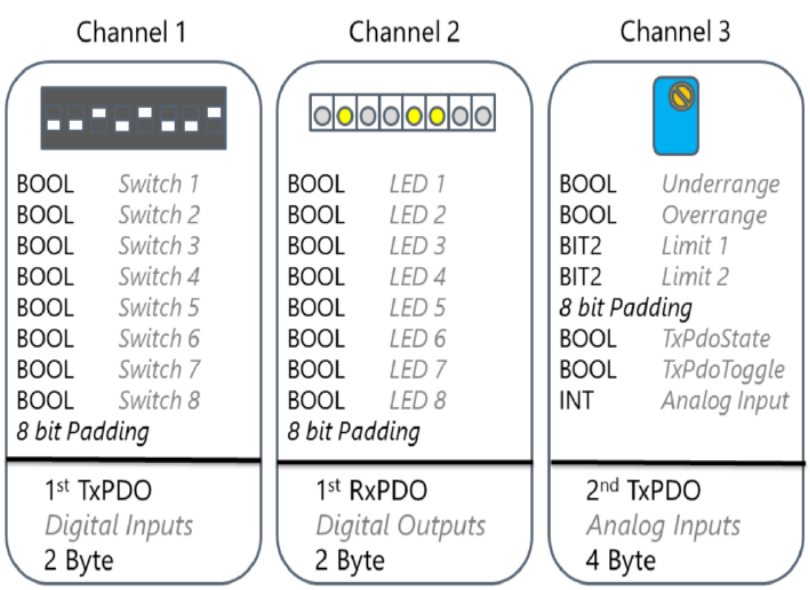


*FIGURE 34. Master Software Implementation*

## Software Layer Architecture

The software layer architecture involves implementing following procedure:

**NIC Driver:**

- Create and bind socket on the NIC port
- Send & receive datagrams on the NIC
- Order received frames and compare with sent

**EtherCAT Protocol Master Layer:**

- Define all slave(s) data structure
- Creating/Mapping all necessary memory for defined messages
- Provide an interface functionality to the application layer

**Application Layer:**

- Read & writing data to be sent/received by the Master
- Keeping local IO data synchronized with the global IOmap
- Detecting errors reported by slave(s)
- Managing errors reported by slave(s)

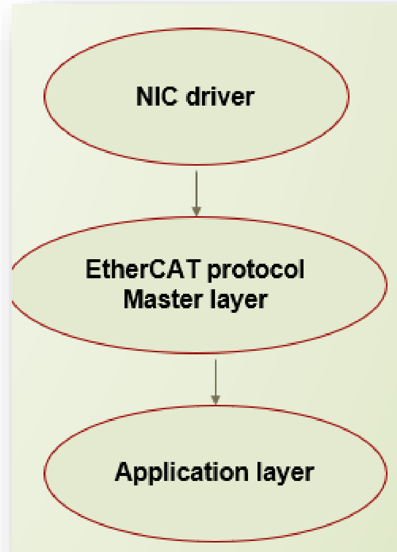


*FIGURE 35. Software Implementation Flow*

## Application Layer Architecture

**Master Initialization & States Transitions:**

Create and Bind socket to NIC port ---- Init State

Discover all Slaves in the network, update Working Counter, and request PreOP

Configure MailBox operations, IOMAP, Clock Sync, and request SafeOP

Send Valid data, calculate expected WKC, and request OP state

**Communication Loop:**

Send & Receive Cyclic frames (PDOs TX/RX)

Send & Receive Acyclic messages (SDOs)

**State & Error handling Thread:**

Lower Priority Thread

Monitor states and WKC

Set error flags when needed

Take actions based on events

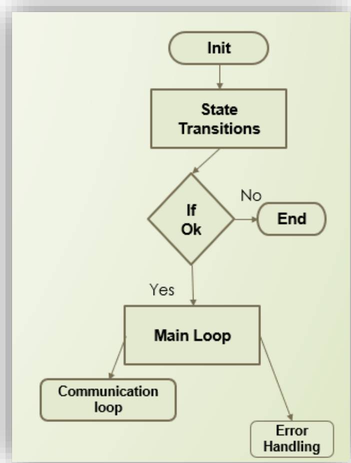


*FIGURE 36. Application Layer flowchart*

## Creating the Application

The application exists under the SOEM (Simple Open EtherCAT Master) library tree, to build the application with the necessary drivers under Linux using the GCC compiler, the following steps must be followed:

1- The application resides under SOEM/applications/EtherCAT/EtherCAT_app.c
2- To build the static library and the application, you need to run **CMAKE** command to create the build top level makefile. There is a **readme_build.txt** under SOEM folder also details the quick steps on building the application.
3- Under Linux go to SOEM and create build directory (mkdir build), and switch to that directory.
4- Apply cmake command (**sudo cmake ..**), and make sure you run sudo for admin privileges.
5- After that run (**sudo make**) to call the top level makefile that will build the driver and the application.
6- After successful build, you can run the application directly by SOEM/build/application/EtherCAT/EtherCAT_app **ifname** (Where ifname is the NIC port name, e.g eth0).
7- To figure out the NIC port names are used in the system, the user could use (ifconfig) command that will show all the network interfaces.

# Analysis and Working Results

Below are summary of the application results and analysis:

1- The application is successfully establishing EtherCAT socket connection on the NIC port, discovering and initializing slave(s) on the network.
2- The application can map and allocate all necessary PDOs (Process Data Object) messages will be used for cyclic TX/RX communication.
3- The application implements three threads (Communication, Error checking, and Printing) threads. The communication thread is running at 5ms that's driving the EtherCAT cyclic communication.
4- All measurements for the four sensors are showing successfully on the screen with the appropriate scaling factors applied to each sensor.
5- The Thermistor and the Photocell sensors were used to drive the LEDs on the Slave devices. The LEDs reflect changes on the two sensors dividing that into intervals.

# Relevant OSHA/FCC Regulations

**OSHA: Occupational Safety and Health Administration**

Due to low voltages there seem to be no applicable OSHA regulations or constraints that our Weather Station violates or needs to be within.

**FCC: Federal Communications Commission Radio Frequency Interference Statement**

"EtherCAT has received FCC approval and has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense (Beckoff)."

# Design Alternatives Considered

If our current design for this project were to be completely optimized and we had the time to do so, we would also look into adding a WiFi adapter and making it so the readings can be sent to a mobile device or a computer to enhance ease of use for the end user. We would also look into making a mobile device application that can easily be opened to monitor in real time the output of each sensor.

If this were scaled for mass production, there would have to be a cheaper alternative used as EtherCAT is very expensive and is too advanced to be used for such a simple application. One comparable alternative that would cost about 10% of what the EtherCAT system does is an Arduino MEGA. It may not be as fast as EtherCAT but in this application it can perform the tasks with no real difference to be seen by the end user.

In addition to not being as expensive, the Arduino boasts more capabilities for cheaper as it can be scaled to work with WiFi and Bluetooth shields cheaper than getting compatible shields for the Infineon/TI hardware. Also, with the Arduino there is no need to have any hardwired Ethernet cables which are lengthy and rather cumbersome to deal with in the application of a weather station.

# Overcoming Challenges

| Challenge | Solution |
|---|---|
| A big challenge our group ran into was trying to coordinate the hardware build up while not being able to meet in person. | The best way we found overcoming the issue of hardware bring up was to pick the chipset together over a zoom meeting. From there the work was delegated as Matt attached wires and modified the hardware based on Brian's schematics. |
| Another challenge we encountered was trying to write software without ever physically meeting. | Mostafa took the lead on software due to his tremendous background in writing software and passed use of EtherCAT systems. He was able to write code and have sporadic meetings showing Matthew and Brian the logic of the code as well as syntax. |
| Merging the EtherCAT masters and slaves with the sensors that are not physically in the same area. | With the lockdown order issued, sensors were stuck at Matthews house for quite some time. In this Mostafa was able to create the skeleton of the code and use his coding skills to simulate the sensors so integration would be seamless once the sensors arrived. |
| Shipping time on sensors was longer than expected. | With shipments being delayed, our group was awaiting 3 of the 5 sensors but were able to work through by using a photoresistor and thermistor that came with our Arduino set. |

# Conclusion

Overall, this project was successful in demonstrating the operation of a weather station using an EtherCAT communication system. We achieved our main objectives of creating a full duplex EtherCAT system that is able to support two slaves and four different sensors. The system was able to maintain a consistent and reliable communication between the master, slaves, and sensors. It was also able to update and output all information as intended. The system was able to update accurately accordingly to changes in the sensor inputs due to environmental changes and is a success in all aspects.

# References

Adafruit. “Photocells.” *Adafruit*, www.mouser.com/datasheet/2/737/photocells-932884.pdf. [Accessed 6 April 2020].

*Beckhoff Information System - English*, infosys.beckhoff.com/english.php?content=..%2Fcontent%2F1033%2Fcx8010_hw%2F3672258955.html&id=.

CSEstack. 2020. 16 Advantages And Disadvantages Of Ethernet | With Its Characteristics. [online] Available at: <https://www.csestack.org/advantages-and-disadvantages-of-ethernet-characteristics-types/> [Accessed 30 March 2020].

CSEstack. 2020. 16 Advantages And Disadvantages Of Ethernet | With Its Characteristics. [online] Available at: <https://www.csestack.org/advantages-and-disadvantages-of-ethernet-characteristics-types/> [Accessed 30 March 2020].

GitHub. 2020. Openethercatsociety/SOEM. [online] Available at: <https://github.com/OpenEtherCATsociety/SOEM> [Accessed 30 March 2020].

“HIH-4000-001.” *DigiKey*, www.digikey.com/product-detail/en/honeywell-sensing-and-productivity-solutions/HIH-4000-001/480-2905-ND/859092?utm_adgroup=Sensors%2B%26%2BTransducers&utm_source=google&utm_medium=cpc&utm_campaign=Dynamic%2BSearch&utm_term=&utm_content=Sensors%2B%26%2BTransducers&gclid=Cj0KCQjwpfHzBRCiARIsAHHzyZru1VViZvYz4yfzaaf-4sL0eUMy7gwOp4tosscZVyu8Z7DpjOuK2EgaAsxeEALw_wcB. [Accessed 6 April 2020].

Infineon. *AM335x ICE EVM Rev2.1 Hardware User's Guide*. www.ti.com/lit/ug/spruip3/spruip3.pdf . [Accessed 6 April 2020].

Infineon. *KP236N6165 Analog Absolute Pressure Sensor*. www.infineon.com/dgdl/Infineon-KP236_N6165-DS-v01_00-en.pdf?fileId=db3a30432ad629a6012af68133600b1a. [Accessed 6 April 2020].

Infineon. *XMC4800 Relax EtherCAT Kit*. www.infineon.com/dgdl/Infineon-Board_User_Manual_XMC4700_XMC4800_Relax_Kit_Series-UM-v01_02-EN.pdf?fileId=5546d46250cc1fdf01513f8e052d07fc. [Accessed 6 April 2020].

PX. *PX Series Precision Interchangeable Thermistors*. www.mouser.com/datasheet/2/240/Littelfuse_Leaded_Thermistors_Interchangeable_Ther-1372423.pdf. [Accessed 6 April 2020].

“THERMISTOR BASICS.” *Wavelength Electronics*, Wavelength Electronics, 11 Feb. 2020, www.teamwavelength.com/thermistor-basics/.

# Appendix I – Parts List

| Items | Manufacturer | Value (USD) | Quantity | Subtotal | Description |
|---|---|---|---|---|---|
| AM3359 Industrial Communications Engine (ICE) | TI | $189.00 | 1 | $189.00 | ICE is development platform targeted based on Sitara AM335x ARM® Cortex™-A8 Processors. The target will be installed with RealTime Linux kernal that will act as the EtherCAT Master in the communication network |
| XMC4800 Relax EtherCAT Kit | Infineon | $57.50 | 2 | $115.00 | XMC4800-F144 Microcontroller based on ARM® Cortex®-M4@144MHZ, integrated EtherCAT® Slave Controller |
| Thermistor | Elegoo | $0.43 | 1 | $0.43 | Sensor comes with the Elegoo UNO Project Super Starter Kit |
| Photoresistor (Photocell) | Elegoo | $0.95 | 1 | $0.95 | Sensor comes with the Elegoo UNO Project Super Starter Kit |
| IC ANLG BAROMETRIC SNSR DSOF8-16 | Infineon | $14.64 | 1 | $14.64 | Pressure Sensor 8.7PSI ~ 23.93PSI |
| HIH-4000-001 (Humidity Temperature Sensor) | Honeywell | $17.17 | 1 | $17.17 | Humidity Temperature Sensor 0 ~ 100% RH Analog Voltage ±3.5% RH 5s Through Hole |
| Total | - | - | - | $337.19 | |

# Appendix II – Cost Analysis

After finishing the project, the total cost came out to $337.19

The largest portion of the cost accrued came from the EtherCAT system totaling a whopping $304. This came from the EtherCAT Master (AM3359 Industrial Communications Engine (ICE)) at $189.00 and the EtherCAT slaves (XMC4800 Relax EtherCAT Kit) coming in at a hefty $115.00.

# Appendix III – Project Code

```
/**********************************************************************
 * NAMES: Mostafa AlNaimi, Matthew Hagan, and Brian Yousif-Dickow
 * FILE: EtherCAT_app.c
 * PURPOSE: ECE5620 - Final Project
 * CONTENTS:  EtherCAT Master application to control a weather station.
 *            Linear Topology contains a master and two slave devices.
 *            The sensors (Thermistor, Photocell, Humidity, and Pressure)
 *            will be connected to the Analog interfaces on the slave
devices.
 *            Analog interfaces will be mapped and transfer to the master
 *            through EtherCAT communication. The application uses SOEM
 *            Open Source library for establishing the EtherCAT
communication.
 *
 * PROJECT:   Weather Station Control using EtherCAT communication
 *
 * Usage : ./EtherCAT_app [ifname1]
 *
 * NOTES:
 *  (1) ifname is NIC interface, f.e. eth0
 *  (2) User could get ifname using Linux by ifconfig command
 *  (3) To close the application, Enter quit in the terminal
 *  (4) Measurements for all sensors will be shown on the screen
 *  (5) LEDs flashing only reflect Temperature and Light sensors
 *
 * KNOWN PROBLEMS:
 *
 * PORTABILITY CONCERNS:
 *
 * ENTRY POINTS (callable functions):
 *
 *      main( int argc, char *argv[] )
 *      ecat_init( ecat_master_drv_t *drvr_data_p )
 *      comm_thread( void *p )
 *      ecat_err_check( void *ptr )
 *      ecat_print( void *p )
 *      scale_sensors( uint16_t reading, int flag )
 *
 **********************************************************************/

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <inttypes.h>

#include  <stdlib.h>
#include  <unistd.h>
#include  <sys/time.h>
#include  <time.h>
```

```
#include  <pthread.h>
#include  <sched.h>

#include "ethercat.h"

/* Define SUCCESS and FAIL will be used for status throughout the
application */
#define SUCCESS 0
#define FAIL -1

/*
 * Reconfiguration Timeout in us
 */
#define  EC_TIMEOUTMON                           500

/*
 * Max I/O MAP SIZE
 */
#define  MAX_IO_MAP                              2048

/*
 * Max SAFE_OP Tries
 */
#define  MAX_SAFE_OP_TRY                         50

/*
 * Network interfaces
 */
#define  MAX_ECAT_NIC_PORT                       2
#define  MAX_ECAT_NIC_NAMELEN                    50

/*
 * Slave Device Information
 */
#define  MAX_SLAVE_NAMELEN                       40
#define  MAX_NUM_SLAVE                           20

/* EtherCAT Error checking, Printing & Communication threads priorities*/
#define  ECAT_PRINT_THREAD_PRIO              20
#define  ECAT_ERR_CHECK_THREAD_PRIO          30
#define  ECAT_COMM_THREAD_PRIO               40

/* EtherCAT State Test Time in us */
#define ECAT_TEST_TIME                           500

/* Light Sensor Reading Scale */
#define LOW_RANGE  1000
#define HIGH_RANGE 3500

/* Temperature Sensor Reading Scale */
#define TEMP_LOW_RANGE  70.0
#define TEMP_MID_RANGE  74.0
#define TEMP_HIGH_RANGE 78.0
```

```
/* Sensors flags will be used for scaling function */
#define TEMP_SENSOR  1
#define LIGHT_SENSOR 2
#define PRESS_SENSOR 3
#define HUMID_SENSOR 4

/*
 * Data used by the driver for each instance of the driver.
 */
#define ERR_STR_LEN                  100

/* Define global structures will be used throughput the application. */
typedef struct ecat_slave ecat_slave_t;

typedef struct ecat_slave {
    char                slave_name[MAX_SLAVE_NAMELEN];   /* Slave name per
ESI file */
    uint32_t            slave_id;       /* Slave ID per ESI file */
    int                 slave_index;    /* Index of the slave in the list
*/
    char                *out_addr;      /* Pointer to the first byte of the
output buffer */
    char                *in_addr;       /* Pointer to the first byte of the
input buffer */
    int                 curr_state;    /* Slave current state */
    int                 req_state;     /* Slave requested state */
    float               temp_sensor;   /* Reading temperature sensor */
    float               light_sensor;  /* Reading photocell sensor */
    float               press_sensor;  /* Reading pressure sensor */
    float               humd_sensor;   /* Reading humdity sensor */
    char                leds;          /* Writing LED values */
    char                buttons;       /* Reading buttons values */
    uint16_t            sensor_data;
    uint16_t            sensor_data1;
    int                 sm;             /* Number of SyncManagers
configured */
    int                 fmmu;           /* Number of FMMUs configured */
    int                 dc;             /* Flag if slave supports
Distributed Clock */
    int                 lost_connection; /* Flag is set when slave lose
connection */
    int                 rx_size;        /* RX PDOs size */
    int                 tx_size;         /* TX PDOs size */
} ecat_slave;

typedef struct {

    int              status; /* Status variable to control the application
*/
    int              redundant_mode; /* Copy of Application Param */
    char             err_detail[ERR_STR_LEN + 1]; /* for file parsing
errors */
```

```
    char              nic_port[MAX_ECAT_NIC_NAMELEN];   /* NIC port name */
    char              nic_red_port[MAX_ECAT_NIC_NAMELEN];   /* NIC port
name */
    pthread_t         errcheck_tid;  /* EtherCAT error check thread */
    pthread_t         printing_tid;  /* EtherCAT error check thread */
    pthread_t         rcv_tid;     /* EtherCAT communication thread */
    int               op_flag;      /* Operational state flag */
    volatile int      wkc;       /* Working Counter = Number of Slaves
found */
    volatile int      expectedWKC; /* Expected Working Counter when frame
is received */
    char              IOmap[MAX_IO_MAP]; /* Pointer to the I/O MAP */
    int               IOmap_size;  /* Size of I/O MAP for all Slaves*/
    ecat_slave        *targets[MAX_NUM_SLAVE];   /* Copy of slaves
structure */
    int               num_slaves; /* Number of slaves configured */
    double            curr_time;  /* Current Clock Time */
    double            prev_time;  /* Previous Clock Time */
    struct            timeval tv; /* Time spec */
    int               read_wkc;  /* Monitor Working Counter */
    unsigned long     SimCounter;  /* Counter to keep track of running
cycles */

} ecat_master_drv_t;

static char exp_exit_cmd[]="quit";

/* Define function calls and threads will be used in the application */
static int ecat_init(ecat_master_drv_t *drvr_data_p);
static int ecat_app_close(ecat_master_drv_t *drvr_data_p);
static void *comm_thread(void *p);
static void *ecat_err_check( void *p );
static void *ecat_print( void *p );
static float scale_sensors( uint16_t reading, int flag );

/**************************************************************************
**
 *  scale_sensors()
 *
 *     This function will be used to scale the sensors from the
 *     raw reading values.
 *

*************************************************************************/
static float scale_sensors( uint16_t reading, int flag )
{
    /* Defining Constants for Conversion & temp variables */
    float R1 = 100000;
    float logR2, R2, T;
    float c1 = 1.009249522e-03, c2 = 2.378405444e-04, c3 = 2.019202697e-
07;
    float humdity_const = 0.0048875;
    const int zero = 102, span =  819;
```

```
    float humdity_conversion;
    float pressure;

    /* Scaling Temperature sensor analog values */
    if (flag == TEMP_SENSOR){
        R2 = R1 * (4095.0 / (float)reading - 1.0);
        logR2 = log(R2);
        T = (1.0 / (c1 + c2*logR2 + c3*logR2*logR2*logR2));
        T = T - 273.15;
        T = (T * 9.0)/ 5.0 + 32.0;
        T *= -1 ;
        return T;
    }

    /* Scaling Pressure sensor analog values */
    else if (flag == PRESS_SENSOR){
        pressure = (reading - zero)  * 15.0 / span;
        return pressure;
    }

    /* Scaling Humidity sensor analog values */
    else if (flag == HUMID_SENSOR){
        humdity_conversion = humdity_const * (float)reading;
        return (float)((humdity_conversion-0.86)/0.03);
    }

    return SUCCESS;
}
/*************************************************************************
 *  ecat_print()
 *
 *     This thread handles printing EtherCAT information, Slave(s) info,
 *     and all necessary counters and Data.
 *

*************************************************************************/
static void *ecat_print( void *p )
{
    ecat_master_drv_t *drvr_data_p;

    /* Copy local data structure */
    drvr_data_p = (ecat_master_drv_t *) p;

    while (1){

        /* Print sensors and cycle information */
        printf("Num of Slaves(s):%d ", drvr_data_p->num_slaves); /* Number
of slaves */
        printf(" | Temp:%f | Light:%f ", drvr_data_p->targets[0]-
>temp_sensor,
            drvr_data_p->targets[0]->light_sensor); /* Temperature and
Light sensors */
```

```
        printf(" | Pressure:%f | Humidity:%f ", drvr_data_p->targets[1]-
>press_sensor,
            drvr_data_p->targets[1]->humd_sensor); /* Pressure & Humidity
sensors */
        printf(" | Num cycles:%ld\r", drvr_data_p->SimCounter); /* Cycle
count */

    /* Sleep 100ms */
    osal_usleep( 100000 );
    }
    return NULL;
}
/***************************************************************************
 *  ecat_err_check()
 *
 *     This thread handles monitor EtherCAT state machine. Take actions
 *     upon found errors and bad states. The thread sleeps for only 10ms
 *

***************************************************************************/
static void *ecat_err_check( void *p )
{
    ecat_slave     *target;
    int            safe_count, i, num_err_slave;
    ecat_master_drv_t *drvr_data_p;

    /* Copy local data structure */
    drvr_data_p = (ecat_master_drv_t *) p;

    /* Initialize local variable */
    safe_count = 0;
    num_err_slave = 0;

    while(1)
    {
        /* Monitor the state machine of each slave */
        for(i = 0; i < drvr_data_p->num_slaves; i++){
            target = drvr_data_p->targets[i];
            if (target != NULL){
                target->curr_state = ec_statecheck(target->slave_index,
EC_STATE_OPERATIONAL, EC_TIMEOUTRET);
                /* If the state is not operational then set lost
connection flag */
                if (target->curr_state != EC_STATE_OPERATIONAL) target-
>lost_connection = 1;
            }
            /* Count how many slaves are lost connection */
            if (target->lost_connection){
                num_err_slave++;
            }
        }

        /*
```

```
         * Verify working counter.
         * Check if all slaves are not in operational mode.
         */
        if(drvr_data_p->wkc < drvr_data_p->expectedWKC && num_err_slave ==
drvr_data_p->num_slaves)
        {
            /* One or more slaves are not responding */
            for (i = 0; i < drvr_data_p->num_slaves; i++){

                target = drvr_data_p->targets[i];
                if (target != NULL){
                    /* Slave is not in OPERATIONAL mode */
                    if ( ec_slave[target->slave_index].state !=
EC_STATE_OPERATIONAL ) {

                        /* Attempting ACK and set to Init state */
                        if (ec_slave[target->slave_index].state ==
(EC_STATE_SAFE_OP + EC_STATE_ERROR))
                        {
                            /* Acknowledge error and set the state to Init
*/
                            printf("ERROR : slave %d is in SAFE_OP +
ERROR, attempting ack.\n",
                                        target->slave_index);
                            ec_slave[target->slave_index].state =
(EC_STATE_SAFE_OP + EC_STATE_ACK);
                            ec_writestate(target->slave_index);
                            ec_slave[target->slave_index].state =
(EC_STATE_INIT);
                            ec_writestate(target->slave_index);
                        }
                        /* Slave is in SAFE_OP. Try to switch to
OPERATIONAL state */
                        else if(ec_slave[target->slave_index].state ==
EC_STATE_SAFE_OP)
                        {
                            /* When maximum tries take a place, switch to
init state */
                            if ( safe_count >= MAX_SAFE_OP_TRY ){
                                ec_slave[target->slave_index].state =
(EC_STATE_INIT);
                                ec_writestate(target->slave_index);
                                printf("WARNING : slave %d exceeded MAX
Number for SAFE_OP. Switch to Init\n",
                                            target->slave_index);
                                drvr_data_p->op_flag = 0;
                            }
                            else{
                                /* If the slave is in safe operational
mode, then switch to Operational */
                                printf("WARNING : slave %d is in SAFE_OP,
change to OPERATIONAL.\n",
                                            target->slave_index);
```

```
                                ec_slave[target->slave_index].state =
EC_STATE_OPERATIONAL;
                                ec_writestate(target->slave_index);
                                target->lost_connection = 0;
                                safe_count = safe_count + 1;
                            }
                        }

                        /* Try to re-config Slave */
                        else if(ec_slave[target->slave_index].state >
EC_STATE_NONE)
                        {
                            if (ec_reconfig_slave(target->slave_index,
EC_TIMEOUTMON))
                            {
                                ec_slave[target->slave_index].islost =
FALSE;
                                printf("Slave %d is
reconfigured\n",target->slave_index);
                            }
                        }
                        else if(!ec_slave[target->slave_index].islost)
                        {
                            /* Slave is lost and re-checking the state */
                            ec_statecheck(target->slave_index,
EC_STATE_OPERATIONAL, EC_TIMEOUTRET);
                            if (ec_slave[target->slave_index].state ==
EC_STATE_NONE)
                            {
                                ec_slave[target->slave_index].islost =
TRUE;
                                printf("ERROR : slave %d lost\n",target-
>slave_index);
                            }
                        }
                    }
                    /* Can't find the slave on the network, totally lost
*/
                    if (ec_slave[target->slave_index].islost)
                    {
                        if(ec_slave[target->slave_index].state ==
EC_STATE_NONE)
                        {
                            if (ec_recover_slave(target->slave_index,
EC_TIMEOUTMON))
                            {
                                ec_slave[target->slave_index].islost =
FALSE;
                                target->lost_connection = 0;
                                printf("MESSAGE : slave %d
recovered\n",target->slave_index);
                            }
                        }
```

```
                        else
                       {
                            ec_slave[target->slave_index].islost = FALSE;
                            target->lost_connection = 0;
                            printf("MESSAGE : slave %d found\n",target-
>slave_index);
                        }
                    }
                }
            }
        }
        /* Sleep 10ms */
        osal_usleep( 10000 );
    }

    return NULL;
}
/***********************************************************************
 *  comm_thread()
 *
 *     This thread handles EtherCAT Communication for PDOs read/write.
 *     The thread sleeps only for 5 milliseconds time.
 *
 ***********************************************************************/
static void *comm_thread(void *p)
{
    ecat_slave     *target;
    int i;
    ecat_master_drv_t *drvr_data_p;
    char *data_ptr;
    char *data_ptr1;
    char led_data;

    /* Copy application data structure */
    drvr_data_p = (ecat_master_drv_t *) p;

    /*
     * Monitor Operational flag for slave(s) during runtime.
     */
    while ( drvr_data_p->op_flag ){

        /*
         * Do the write for PDOs only when Slave is in Operational mode.
         */
        for (i = 0; i < drvr_data_p->num_slaves; i++){
            target = drvr_data_p->targets[i];
            if (target != NULL && ec_slave[target->slave_index].state ==
EC_STATE_OPERATIONAL){
                /* Write Data to PDOs */
                if (target->slave_index == 1){

                    /* Scale light sensor raw data based on 3 intervals */
                    /* High range interval - Bright */
```

```
                    if (target->light_sensor > HIGH_RANGE){
                        led_data = 0xff;
                        memcpy(drvr_data_p->targets[1]->out_addr,
&led_data, sizeof(char));
                    }
                    /* Mid range interval - Normal room light */
                    else if (target->light_sensor > LOW_RANGE && target-
>light_sensor < HIGH_RANGE){
                        led_data = 0x0f;
                        memcpy(drvr_data_p->targets[1]->out_addr,
&led_data, sizeof(char));
                    }
                    /* Low range interval - Dark */
                    else{
                        led_data = 0x00;
                        memcpy(drvr_data_p->targets[1]->out_addr,
&led_data, sizeof(char));
                    }

                    /* Scale Temperature values based on 4 intervals */
                    /* Room Temperature */
                    if (target->temp_sensor > TEMP_LOW_RANGE && target-
>temp_sensor < TEMP_MID_RANGE){
                        led_data = 0x07;
                        memcpy(target->out_addr, &led_data, sizeof(char));
                    }
                    /* Nice weather Temperature */
                    else if (target->temp_sensor > TEMP_MID_RANGE &&
target->temp_sensor < TEMP_HIGH_RANGE){
                        led_data = 0x1F;
                        memcpy(target->out_addr, &led_data, sizeof(char));
                    }
                    /* Warm Temperature */
                    else if (target->temp_sensor > TEMP_HIGH_RANGE){
                        led_data = 0xFF;
                        memcpy(target->out_addr, &led_data, sizeof(char));
                    }
                    /* Cold Temperature */
                    else if (target->temp_sensor < TEMP_LOW_RANGE) {
                        led_data = 0x00;
                        memcpy(target->out_addr, &led_data, sizeof(char));
                    }
                }
            }
        }
        /* Transmit Process Data for PDOs */
        ec_send_processdata();

        /* Receive Process Data for PDOs*/
        drvr_data_p->wkc = ec_receive_processdata(EC_TIMEOUTRET);

        /*
         * Do the read for PDOs only when Slave is in Operational mode.
```

```
         */
        for (i = 0; i < drvr_data_p->num_slaves; i++){
            target = drvr_data_p->targets[i];
            if (target != NULL && ec_slave[target->slave_index].state ==
EC_STATE_OPERATIONAL){
                /* For slave 1, copy reading for Light and Temperature
sensors */
                if (target->slave_index == 1){

                    data_ptr = target->in_addr;
                    memcpy(&target->sensor_data, data_ptr,
sizeof(uint16_t));
                    data_ptr += sizeof(uint16_t);
                    memcpy(&target->sensor_data1, data_ptr,
sizeof(uint16_t));
                    data_ptr += sizeof(uint16_t);
                    memcpy(&target->buttons, data_ptr, sizeof(char));

                    target->light_sensor = (float) (target->sensor_data);
                    /* Convert temperature reading scale */
                    target->temp_sensor = scale_sensors(target-
>sensor_data1, TEMP_SENSOR);
                }

                /* For slave 2, copy reading for Pressure and Humidity
sensors */
                else if (target->slave_index == 2){

                    data_ptr1 = target->in_addr;
                    memcpy(&target->sensor_data, data_ptr1,
sizeof(uint16_t));
                    data_ptr1 += sizeof(uint16_t);
                    memcpy(&target->sensor_data1, data_ptr1,
sizeof(uint16_t));
                    data_ptr1 += sizeof(uint16_t);
                    memcpy(&target->buttons, data_ptr1, sizeof(char));

                    /* Convert Pressure & Humidity */
                    target->press_sensor = scale_sensors(target-
>sensor_data, PRESS_SENSOR);
                    target->humd_sensor = scale_sensors(target-
>sensor_data1, HUMID_SENSOR);
                }
            }
        }
        /*Increment Simulation Counter */
        drvr_data_p->SimCounter = drvr_data_p->SimCounter + 1;

        /* Sleep for 5ms */
        osal_usleep( 5000 );
    }

    return (NULL);
```

```
}                               /* comm_thread */

/***********************************************************************
*
 *
 *   ecat_app_close() - Request all slave(s) to Init state, close EtherCAT
 *       socket connection on NIC port, free all driver data structures,
 *       and cancel all threads.
 *

***********************************************************************
/
int ecat_app_close(ecat_master_drv_t *drvr_data_p)
{
    ecat_slave *target;
    int i;

    /* Request Init State for all Slaves */
    drvr_data_p->op_flag = 0;
    ec_slave[0].state = EC_STATE_INIT;
    ec_writestate(0);

    /* stop SOEM, close socket */
    ec_close();

    /* Cancel all active threads */
    if ( drvr_data_p->rcv_tid ){

        pthread_cancel(drvr_data_p->printing_tid);
        usleep(5000);
        pthread_cancel(drvr_data_p->errcheck_tid);
        usleep(5000);
        pthread_cancel(drvr_data_p->rcv_tid);
        usleep(5000);
    }

    /* Free slave(s) data structure */
    for(i = 0; i < drvr_data_p->num_slaves; i++){
        target = drvr_data_p->targets[i];
        if (target != NULL)
            free(target);
    }

    /* Free application data structure */
    free(drvr_data_p);

    return SUCCESS;

}
/***********************************************************************
*
 *
 *   ecat_init() - Initialize NIC port for EtherCAT, Configure Slaves
```

```
 *               and I/O MAP for PDOs, and Request OP state for Slaves.
 *

***************************************************************************
/
int ecat_init ( ecat_master_drv_t *drvr_data_p ){

    ecat_slave *target;
    ec_adaptert *ret_adapter;
    int i, status, states, adapter_found, red_adapter_found;

    /* Initialize local variable */
    adapter_found = 0;
    red_adapter_found = 0;

    /*
     * Find available adapters on the system.
     * Compare Found adapters with the passed NIC port.
     */
    ret_adapter = ec_find_adapters();
    if ( ret_adapter == NULL ){
        printf("Can't find available network adapters on the system\n" );
        return FAIL;
    }

    /* Match Network adapter with the one passed through the argument */
    for ( ; ret_adapter != NULL; ret_adapter = ret_adapter->next ){

        if ( strcmp(drvr_data_p->nic_port, ret_adapter->name) == 0) {
            adapter_found = 1;
            break;
        }
    }

    /* Couldn't find matched network adapters */
    if ( !adapter_found ) {
        printf("Can't find matched network adapter\n" );
        return FAIL;
    }

    /*
     * Check for redundancy port
     * Not supported in this application
     */
    if ( drvr_data_p->redundant_mode ){
        ret_adapter = ec_find_adapters();
        for ( ; ret_adapter != NULL; ret_adapter = ret_adapter->next ){

            if ( strcmp(drvr_data_p->nic_red_port, ret_adapter->name) ==
0) {
                red_adapter_found = 1;
                break;
            }
```

```
        }

        /* No redundant network adapter was found */
        if ( !red_adapter_found ) {
            printf("Can't find matched redundant network adapter\n" );
            return FAIL;
        }
    }

    /*
     * Initialize and bind socket to nic_port on a single port, or
     * check for redundant mode (second port).
     */
    if ( drvr_data_p->redundant_mode ){
        status = ec_init_redundant( drvr_data_p->nic_port,
                                    drvr_data_p->nic_red_port );
    }

    /* Single NIC mode */
    else {
        status = ec_init( drvr_data_p->nic_port );
    }

    /*
     * Continue to discover and configure slave(s) if
     * Socket connection of the EtherCAT was established successfully.
     */
    if ( status )
    {
        /* SOEM init is done and NIC is up */

        /*
         * Broadcast read request to all functional slaves in the network.
         * Update Working Counter and sets up Mailboxes. Request PRE OP
state.
         */
        if ( ( drvr_data_p->wkc = ec_config_init(FALSE) ) > 0 ){

            /*
             * Copy Slave name, Slave ID, and store Slave index
             * Store Slave information (FMMU, Distributed Clock, and
number of SyncManger)
             */
            drvr_data_p->num_slaves = ec_slavecount;
            i = 1;
            while ( i <= ec_slavecount ){
                /* Allocate Slave data structure */
                target = (ecat_slave *) calloc(1, sizeof(ecat_slave));
                if (target == NULL){
                    printf("Can't allocate slave data structure\n");
                    return FAIL;
                }
                /* Slave index */
```

```
                target->slave_index = i;
                /* Slave ID */
                target->slave_id = ec_slave[i].eep_man;
                /* Copy Slave name */
                strncpy(target->slave_name, ec_slave[i].name,
strlen(ec_slave[i].name));
                /* Check if Slave has a distributed clock */
                target->dc = ec_slave[i].hasdc;
                /* Store slave structure in the local application
structure */
                drvr_data_p->targets[i-1] = target;
                i++;
            }

            /* Can't find slave(s) devices in the network */
            if ( drvr_data_p->num_slaves < 1 ){

                printf(drvr_data_p->err_detail, "Can't find EtherCAT
Slave(s) in the network\n");
                return FAIL;
            }

            /* Request all slaves to reach PRE_OP state */
            states = ec_statecheck(0, EC_STATE_PRE_OP,  EC_TIMEOUTSTATE *
4);
            if ( states != EC_STATE_PRE_OP ){

                /* Slave(s) can't reach Pre operational state */
                printf("Slaves can't reach PRE_OP state\n" );
                return FAIL;
            }

            /*
             * Create I/O MAP for slaves. Configure SyncMangers and FMMUs.
             * Update expected Working Counter and request SAFE_OP state.
             */
            drvr_data_p->IOmap_size = ec_config_map(&(drvr_data_p-
>IOmap));

            if (!(drvr_data_p->IOmap_size)){
                printf("Can't create IO map for PDOs\n");
                return FAIL;
            }

            /* Store copies of the I/O address will be accessed to
write/read PDOs */
            for ( i = 0; i < drvr_data_p->num_slaves; i++){

                target = drvr_data_p->targets[i];
                target->out_addr = (char *) ec_slave[target-
>slave_index].outputs;
                target->in_addr = (char *) ec_slave[target-
>slave_index].inputs;
```

```
                /* Copy number of bytes of TX PDOs */
                if (ec_slave[target->slave_index].Obytes)
                    target->tx_size = ec_slave[target-
>slave_index].Obytes;
                /* Num of bits instead */
                else
                    target->tx_size = 1;

                /* Copy number of bytes of RX PDOs */
                if (ec_slave[target->slave_index].Ibytes)
                    target->rx_size = ec_slave[target-
>slave_index].Ibytes;
                /* Num of bits instead */
                else
                    target->rx_size = 1;
            }
            /* Configure Distributed Clock it's used in any slave */
            ec_configdc();

            /* Request all slaves to reach SAFE_OP state */
            states = ec_statecheck(0, EC_STATE_SAFE_OP,  EC_TIMEOUTSTATE *
4);
            if ( states != EC_STATE_SAFE_OP ){

                printf("Slaves can't reach SAFE_OP state\n" );
                return FAIL;
            }

            /* Calculate expected working counter for all slaves in the
network */
            drvr_data_p->expectedWKC = (ec_group[0].outputsWKC * 2) + \
                ec_group[0].inputsWKC;

            for ( i = 0; i < drvr_data_p->num_slaves; i++ ){

                target = drvr_data_p->targets[i];

                if (target != NULL){
                    /* Set Current state for available slaves */
                    target->curr_state = states;

                    /* request OP state for all slaves */
                    ec_slave[target->slave_index].state =
EC_STATE_OPERATIONAL;
                    ec_writestate(target->slave_index);

                    /* send one valid process data to keep slaves happy */
                    ec_send_processdata();
                    ec_receive_processdata(EC_TIMEOUTRET);

                    if (ec_slave[target->slave_index].state !=
EC_STATE_OPERATIONAL){
```

```
                        printf("Slave %d can't reach OP state\n", target-
>slave_index);
                        return FAIL;
                    }
                    /* Set Current state  to Operational */
                    target->curr_state = EC_STATE_OPERATIONAL;
                }
            }

            /* Set Operational Flag */
            drvr_data_p->op_flag = TRUE;
            /* Read system clock */
            gettimeofday(&(drvr_data_p->tv), NULL);
            drvr_data_p->prev_time = (double)drvr_data_p->tv.tv_sec +
(double)drvr_data_p->tv.tv_usec/1000000.0;
        }

        else {
            /* No slave(s) is/are found in the network */
            printf("Can't find Slave(s) in the network\n" );
            return FAIL;
        }
    }
    /* Couldn't establish EtherCAT socket on the NIC port(s) */
    else {
        printf("Can't connect to NIC port(s)\n" );
        return FAIL;
    }
    return SUCCESS;
}

/***************************************************************************
******
 *
 * int main(int argc, char *argv[])
 *
 *   Main function that will drive the Master application.
 *

***************************************************************************
****/
int main(int argc, char *argv[])
{
    pthread_attr_t  attr;
    struct sched_param   sched_param;
    ecat_master_drv_t *drvr_data_p;
    char exit_cmd[20];

    printf("Weather Station - EtherCAT Master application\n\n");
    printf ("Enter: 'quit' to exit the application\n\n");
```

```
    /* Allocate application data structure will be used throughout the
application */
    drvr_data_p = (ecat_master_drv_t *) calloc(1,
sizeof(ecat_master_drv_t));

    if (drvr_data_p == NULL)
    {
        printf("EtherCAT_app FAIL. Couldn't allocate drvr_data_p
structure\n");
        return FAIL;
    }

    /* Copy NIC port name passed through the argument when running the
application */
    if (argc > 1)
    {
        /* Copy NIC port name to local data structure */
        strncpy(drvr_data_p->nic_port, argv[1], strlen(argv[1]));
        printf("Passed NIC port name = %s \n", drvr_data_p->nic_port);
        if (drvr_data_p->nic_port == NULL){
            printf("EtherCAT_app FAIL. Couldn't copy NIC port name\n");
            return FAIL;
        }
    }
    else
    {
        printf("EtherCAT_app FAIL. No NIC name was passed\n");
        printf("Usage: ./EtherCAT_app [ifname], ifname = eth0 for
example\n");
        return FAIL;
    }

    /*
     * Call ecat_init() to establish, initialize, discover and
     * set all slave(s) configuration. Move slave(s) to operational state.
     */
    drvr_data_p->status = ecat_init(drvr_data_p);

    if (drvr_data_p->status < SUCCESS)
    {
        printf("EtherCAT_app FAIL. EtherCAT Master initialization has
failed\n");
        return FAIL;
    }

    /*
     * Create threads for error checking and Communication Cycle.
     */
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
    pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy(&attr, SCHED_RR);
    pthread_attr_getschedparam(&attr, &sched_param);
```

```
    /* Set priority and create communication thread */
    sched_param.sched_priority = ECAT_COMM_THREAD_PRIO;
    pthread_attr_setschedparam(&attr, &sched_param);
    pthread_create(&(drvr_data_p->rcv_tid), &attr,
                comm_thread, (void *) drvr_data_p);

    /* Set priority and create error monitor thread */
    sched_param.sched_priority = ECAT_ERR_CHECK_THREAD_PRIO;
    pthread_attr_setschedparam(&attr, &sched_param);
    pthread_create(&(drvr_data_p->errcheck_tid), &attr,
                ecat_err_check, (void *) drvr_data_p);

    /* Set priority and create Printing thread */
    sched_param.sched_priority = ECAT_PRINT_THREAD_PRIO;
    pthread_attr_setschedparam(&attr, &sched_param);
    pthread_create(&(drvr_data_p->printing_tid), &attr,
                ecat_print, (void *) drvr_data_p);

    /* Wait until user inserts quit to exit out of the application */
    while (1){
        scanf("%s", exit_cmd);
        if (strncmp(exit_cmd, exp_exit_cmd, strlen(exp_exit_cmd)) == 0)
break;
    }

    /* Close application, EtherCAT socket */
    drvr_data_p->status = ecat_app_close(drvr_data_p);
    if (drvr_data_p->status < SUCCESS){
        printf("EtherCAT_app FAIL. ecat_app_close()has failed\n");
        return FAIL;
    }

    printf("\nEtherCAT_app program end\n");
    return SUCCESS;
}
```

# Appendix IV – Partner Contribution

| Task | Matthew Hagan | Mostafa AlNaimi | Brian Yousif-Dickow |
|---|---|---|---|
| Project Brainstorm | 33% | 33% | 33% |
| Procuring Hardware | 40% | 20% | 40% |
| Hardware Assembly | 50% | 20% | 30% |
| Code for Temperature Sensor | 33% | 33% | 33% |
| Code for Pressure Sensor | 33% | 33% | 33% |
| Code for Light Sensor | 33% | 33% | 33% |
| Main Code (Tying all subroutines together) | 20% | 60% | 20% |
| Testing | 33% | 33% | 33% |
| Troubleshooting | 33% | 33% | 33% |
| Hardware Configuration | 33% | 33% | 33% |
| OS Configuration | 33% | 33% | 33% |
| Hardware Schematics | 45% | 10% | 45% |
| Project Management | 33% | 33% | 33% |
| Research | 33% | 33% | 33% |
| Project Report | 33% | 33% | 33% |
| Overall Effort with Respect to Group | 100% | 100% | 100% |