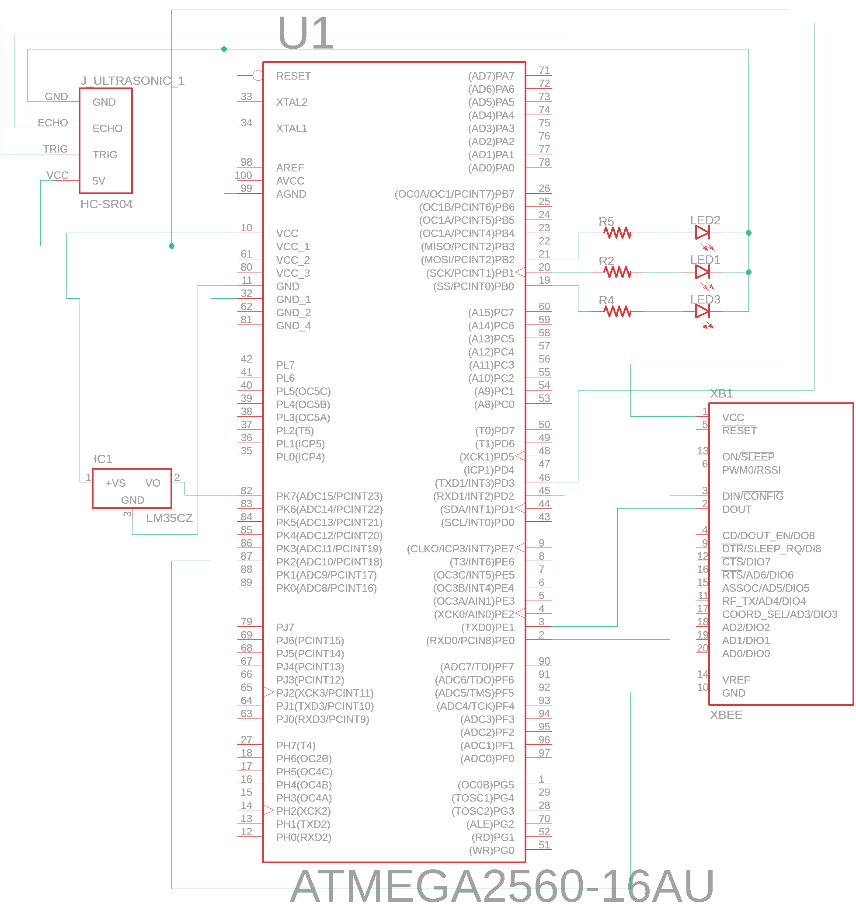
|  |
| --- |
| Benjamin Larsen  24-06-2022 |

|  |
| --- |
| **Senia** |
| Temperatur Styring |
|  |



|  |
| --- |
| **Elev:**  Benjamin Elif Larsen |
| **Projekt:**  Temperatur Styring |
| **Uddannelse:**  Datatekniker med speciale i programmering |
| **Projektperiode:**  21/06/2022 – 24/06/2022 |
| **Afleveringsdato:**  24/06/2022 |

# Titelblad



Techcollege Aalborg,

Struervej 70,

9220 Aalborg

# Forord

Det er en god ide at læse ’Case Beskrivelsen’ først, da denne forklarer hvad der skulle udvikles og hvorfor.

Hvis det ønske at benytte projektets kode, er der behov for at bruge Visual Micro for Visual Studio 2022 og have Optiboot på sin Arduino. Grunden til dette er pga. debugging blev benyttet og kodefilerne er ikke bare Arduinos Ino, men også C, CPP og Header filer. Case.py filen skal køres på en Raspberry Pi som også har docker-compose filen kørende.

# Case beskrivelse

Casens udfordring er at udvikle et system, på en Arduino, der kan sende data til et andet system, en Raspberry Pi, som sender data videre til en MQTT, som ligger på Raspberry Pi’en, kørt via Docker.

Problemet dette case skal løse er temperaturstyring i et område som et rum eller bygning. Dette kan være udluftning eller radiator. Dette kan betyde at der kan bliver brugt ressourcer der ender med at gå til spilde, hvis temperatursystemet er aktiv mens der er ingen i området. Denne case omhandler udviklingen af et system, der kan sikre sig at temperaturstyring kun sker når der er behov for det.

Dermed er denne case omkring at udvikle det del af systemet som registre temperaturen og sende den til en MQTT.

En ekstra udfordring ved denne case, er pga. kommunikation mellem de forskellige dele, betyder det at der er flere steder imellem systemer der kan fejle, hvor systemerne skal håndtere disse fejl.

Dermed sagt skal et system udvikles som kan:

1. Registrere at der personale i området.
2. Læse temperaturen.
3. Sende temperaturen til en bestem enhed.
4. Opsamle temperaturen, tjekke at den er fra en godkendt enhed, og sende den videre til en MQTT.

# Kravspecifikation

Kravspecifikationen, givet nedenunder, er for de vigtigste dele af systemet. Disse er kommunikation mellem Arduino og Raspberry, LED’er der benyttes til at vise tilstanden på Arduino’en. Sikkerheden er også vigtig, da systemet skal benyttes til at styre temperatur og dermed skal det undgås at andre kan påvirke temperaturstyringen.

Kravspecifikation – Kravsoversigt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| KravId | Kategori | Krav | Prioritering | Kilde | Type |
| K-1 | Kommunikation | Data kan sendes via ZigBee | 2 | Protocol | Brainstorm |
| R-2 | Kommunikation | Raspberry skal kun modtage data fra bestemte, kendte enheder | 3 | Sikkerhed | Brainstorm |
| A-1 | Software | Arduino kan læse fra HC-SR04 sensor. | 1 | Hardware | Brainstorm |
| A-1-1 | Software | Arduino læser en enkel gang og venter indtil HC-SR04 ikke længer sender at noget er foran den før arduino læser igen | 1 | Funktion | Brainstorm |
| A-2 | Software | Arduino kan tænde og slukke LEDerne som benyttes til at vise state | 1 | Hardware | Brainstorm |
| A-2-1 | Software | Slukning og tænding sker ud fra bestemte states i koden | 1 | Funktion | Brainstorm |
| A-3 | Software | Den skal benytte state machine til at styre hvad kode der skal køres | 1 | Arkitektur | Brainstorm |
| A-4 | Kommunikation | Arduino kan kun sende til en enkel, bestemt, Xbee | 3 | Sikkerhed | Brainstorm |
| A-5 | Fejlhåndtering | Arduino skal kunne håndtere hvis ZigBee fejler | 2 | Klient | Brainstorm |
| R-1 | Kommunikation | Raspberry skal kunne sende data til MQTT | 1 | Protocol | Brainstorm |
| R-1-1 | Fejlhåndtering | Raspbery skal kunne håndtere hvis den ikke kan kontakte MQTT | 2 | Server | Brainstorm |
| R-3 | Fejlhåndtering | Raspberry skal kunne håndtere hvis den ikke kan åbne Xbee porten | 2 | Server | Brainstorm |
| A-6 | Software | Arduino skal kunne læse fra en temperatur sensor | 1 | Funktion | Brainstorm |
| A-6-1 | Software | Arduino skal have en timer, 10 sekunder, mellem temperaturlæsningerne | 1 | Funktion | Brainstorm |
| R-1-2 | Kommunikation | Raspbery skal kunne sende data til det rigtige topic | 1 | Funktion | Brainstorm |

Sikkerheden i system skal teste for at sikre at andre ikke kan udnytte det. På sammen tid skal der undersøges om både Arduino’en og Raspberry Pi’en kan håndtere at sende og modtage data, da dette er en vigtig del af systemet. Arduion’en skal kunne benytte LED’erne for at vise sin tilstand, da det ikke vil være muligt at debugge den i det færdige produkt, samt at temperatur delen virker korrekt.

Det meste af systemet skal teste via debugging. Koden der kører på Raspberry Pi’en har ikke de samme debugging muligheder, men der kan, via prints og if-statments, tjekke hvad kode der køre og hvad data’en er. Der er muligt at ’debugge’ MQTT via en subscription, i en konsol, på det topic der sendes til, da det er dermed muligt at se om den får modtaget data.

Accepttestoversigt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| KravId | Testbeskrivelse | Testkriterier | Bemærkninger | Status |
| K-1 | Der undersøges om Xbee'en på Arduino'en kan sende til raspberry pi'ens | Arduino kan sende en pakke som modtages korrekt af Raspberry Pi |  |  |
| R-2 | Der undersøges om Raspberry Pi kan droppe det data der sendes af en ukendt Xbee | Data skal ikke analysers og sendes til MQTT, hvis sender addressen ikke er i bestemt liste | Kræver en 3rd Xbee eller at adgive en anden MAC. |  |
| A-1 | Der undersøges om Arduino kan læse fra HC-SRO4 sensor | Data skal være læst korrekt og uden forsinkelser | Benyt en debugger |  |
| A-1-1 | Der undersøges om Arduino kan registrere når der ikke er noget foran sensoren | Data fra sensoren skal føst læse og derefter skal den vente til der ikke er noget foran den |  |  |
| A-2 | Der undersøges om Arduino'en kan starte og slukke for dens state LED’er | Dette skal gøres via en enkel metode kald der kan bruges for alle LED'er |  |  |
| A-2-1 | Der undersøges om Arduino kan benytte statemachine til at tænde og slukke for de rigtige LED'er | Kun en enkel LED skal være tændt ad gangen. Hver state skal tænde og slukke for sin egen LED | LED'en position og port skal være en del af et array |  |
| A-3 | Der undersøges om den rigtige kode er kaldt I Arduino'en ud fra hvad state machine siger | Den rigtige kode kaldes ud fra state machinen |  |  |
| A-4 | Der undersøges om Arduino kun sender data til en bestemt XBEE | Arduino sender til en MAC addresse og ikke broadcast |  |  |
| A-5 | Der undersøges om Arduino kan håndtere fejl i ZigBee protokollen | Arduino er ikke koordinator og skal ikke bryde ned hvis den fejler at sende |  |  |
| R-1 | Der undersøges om Raspbery Pi kan sende data til MQTT | Det sendte data skal kunne læses fra en subscription og passe overens med det sendte data |  |  |
| R-1-1 | Der undersøges om Raspbery Pi håndterer hvis den fejler i at sende til MQTT | Programmet skal kunne forsætte hvis den fejler |  |  |
| R-3 | Der undersøges om Raspberry Pi kan håndtere hvis den ikke kan åbne XBEE Port | Programmet kan give en brugbar fejlbesked |  |  |
| A-6 | Der undersøges om Arduino'en kan læse korrekt fra en temperatur sensor | Den læste temperatur skal passe overens med rummets |  |  |
| A-6-1 | Der undersøges om Arduino'en kan læse på de korrekte tidspunkter | Der skal gå 10 sekunder + kode køre tid mellem hver læsning |  |  |
| R-1-2 | Der undersøges om Raspberry Pi'en kan sende til MQTT på de rigtige topics | Raspberry Pi skal kunne sende på 'Temperatur' og en subscription læser fra dette topic |  |  |

# Hardware opstilling

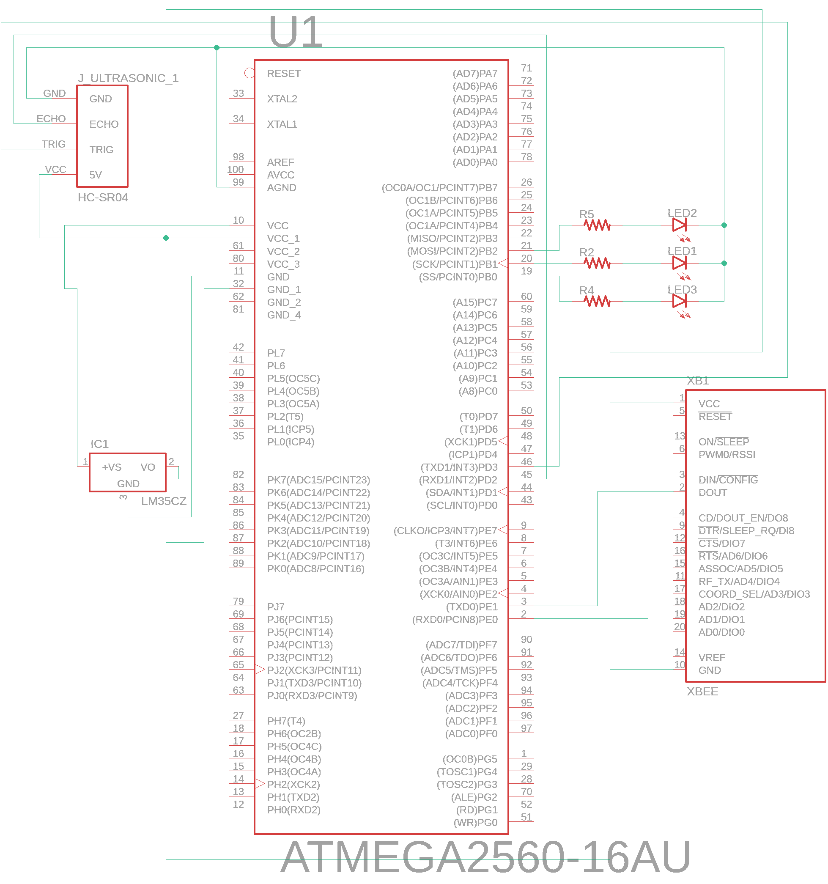
I forhold til hardware er der to mikrocontrollers, en Arduino Mega 2560 og en Raspberry Pi. Alle hardware komponenter var tilsluttet Arduinon’en medundtagen af en enkel XBEE S2C som var tilsluttet Raspberry Pi’en.

LED’erne var placeret på sit eget breadboard og LM35’en var placeret på et andet breadboard. HC-SR04’en sad ikke fast til noget. Arduino’en og Raspberry Pi’en kommunikede med hinanden via Xbees som benytter ZigBee protokollen. Begge Xbee var sat til API 2. Raspbery Pi Xbeen var koordinatoren for netværket.

Se billag A for optilling.

**Komponent liste**

* Raspberry Pi
* Arduino Mega 2560
* LED x 3 (Grøn, gul og rød)
* LM35
* HC-SR04
* XBEE S2C x 2 (Coordinator x 1)
* Modstande x 3 (1k Ohm)



Figur 1 Eagle Diagram

# Program dokumentation

Casen indeholder to programmer, et Python program, kørt på en Raspberry Pi, og et Arduino program, kørt på en Arduino Mega 2560.

Raspberry Pi’en kørte også en docker container som indeholdte en MQTT, Mosquitto, som Python programmet sendte til.

## Arduino

Dette program benytter en statemachine til at styre hvad kode der skal køres, samt to timer til at styre hvornår kode køres. Der er tre states, IDLE, READING\_TEMPERATURE og TRANSMITTING\_DATA. IDLE staten benyttes til at læse fra HC-SR04’en og hvis noget er tæt nok på vil den aktivere de to timer. Den første timer styrer hvornår IDLE kan læse fra HC-SR04’en igen, da den kan afslutte den sidste timer. Den sidste timer vil, når den trigger, skifte state til READING\_TEMPERATURE. Se kodeblokken nedenunder for den nævnte kode kode.

READING\_TEMPERATURE staten læser fra temperaturmåleren, LM35, og skifte state til TRANSMITTING\_DATA. TRANSMITTING\_DATA staten vil sende data ud via den tilsluttede XBee og skifte tilbage til staten IDLE.

De to timer benytter interrupts til at køre bestemte metoder, dermed kan disse metoder køre ligegyldig hvad for noget kode der ellers bliver udført af microchippen.

Pga. hvordan koden er sat op, er koden nedenunder det eneste der skal kaldes via loop’et til at køre alle states i programmet, hvilket gør det let at udvide.

void loop()

{

TimerUpdates();

MachineStateInRAM\_Pointer = StateAllocateMemoryInRamAndGetCopyFromFlashProm();

MachineStateInRAM\_Pointer->FunctionPointer();

}

void DistanceHandling(void)

{

while (IDLE == State && true == CanReadHC)

{

TimerUpdates();

int distance = ReadHCSR04();

if (10 <= distance && 100 >= distance)

{

CanReadHC = false;

timerRead.start();

if (!TimerTempActive) {

timerTemp.start();

TimerTempActive = true;

}

else {

timerTemp.stop();

TimerTempActive = false;

}

}

}

}

## Python

Python koden er meget mere simple end Arduino koden. Det primære kode er givet i kodeblokken nedenunder. Som det kan ses benyttes en while loop, hvor der læses fra Xbee’en. Når den læser data, skriver den remotens MAC i hex og hvis det er den tilladte MAC så skriver programmet den modtaget værdi ud og sender den til MQTT’en. Grunden til at clinet.connect og client.disconnect bliver kaldt hele tiden er fordi clienten vil time out efter noget tid, hvis den ikke sender data.

while(True):

message = device.read\_data()

if message != None:

data = message.data

remote = message.remote\_device.get\_64bit\_addr().address.hex()

print("Remote: ")

print(remote)

if remote == "0013a2004106f9fc":

mes = data.decode()

print("PERMITTED")

print(mes)

client.connect("localhost",1883)

client.publish("temp", payload=mes, qos=1)

client.disconnect()

else:

print("NOT PERMITTED")

# Konklusion

Det endelige produkt opholder problemformuleringen og casen. Arduino’en kan læse temperaturen og sende det data videre via dens Xbee og læse/sende delen af koden kan (de)aktiveres via en HC-SR04. Raspberry Pi’en kan modtage data og sende det vider til MQTT’en. På sammen tid passer den læste temperatur overens med den rigtige temperatur og data er korrekt, når den er sendt over til Raspberry Pi’en/MQTT’en.

Der var en udfordring i udviklingen af Arduino-delen af produktet, da den kode der skulle styre DDR og PORT ikke fungeret korrekt. Den blev ved med at vælge en andre adresse, så det var nødsaget til at benytte pinMode og digitalWrite.

De fleste krav blev løst, dog blev kravet A-1-1 ændret lidt, da det blev valgt at udvikle koden sådan at Arduion’en ikke kan læse for 2 sekunder efter den læste at noget var tæt nok på. Kravet R-3 blev ikke opfyldt, da Python programmet vil stoppe hvis den ikke kan åbne porten, dog vil programmet returnere en brugbar fejlbesked via Xbee biblioteket. For dette lille case program kan det anses for at være fint nok at programmet stopper, siden fejlen vil for det meste opstå pga. den forkerte port er angivet eller at der er noget galt med Xbee’en.

En udfordring i forhold til Python koden var at få fat i MAC’en på remoten på sådan en måde at den kunne blive sammenlignet med den kendte MAC. Dog kom det meste af problemet fra ikke at have godt kendskab til Python og ny til Python Xbee biblioteket.

Det er et mindre problem i forholdet til HC-SR04’en, hvilket er at nogle genstande kan få den til at læse den forkerte afstand, hvilket kan aktivere eller deaktivere sendingen af data.

## Diskussion

Arduinoprogrammet har den fordel at det er let at udvide den med flere states, hvis det skulle blive nødvendigt.

Ulemper ved løsningen er at koden i Raspberry Pi’en skal ændres når der skal tilhøre en ny XBEE S2C, da den ikke tillader alle at snakke med den via ZigBee, dette blev dog gjort pga. sikkerhed. Som koden er sat op lige nu er den hardcoded til at tillade data fra en enkel MAC. Dette burde nok blive udvidet til at loade en liste ind ved program start og så gå igennem den indtil programmet finder en tilladt MAC. På sammen tid er MAC’en som Arduino’en sender til også hardcoded ind i Arduino’en.

En HC-SR04 blev benyttet, en ultra-sonic sensor, hvilket har den betydning at materialet der er foran den, kan påvirke læsningen, og dermed gives en forkert værdi.

Videre udvikling af produktet kunne f.eks. at have sættet flere Arduino’er op eller havde sat den nuværende op sådan at den ville tjekke om en person var på vej ind eller ud af området via flere HC-SR04’er. Lige nu tjekke den bare om nogen kommer forbi den. På sammen tid kunne er det muligt at have sat noget kode op til at benytte det data der blev sendt hen til MQTT’en. I forhold til selve hardware’en på Arduino’en været udvidet med en ekstra LED der indikeret om hvorvidt state skifter timeren var aktiv eller ej, da der går 10 sekunder mellem LED’erne for temperaturlæsning og data afsending bliver aktiveret. Arduino’en kunne har været sat op sådan at en seriel forbindelse kunne tilsluttes til den, så der kunne læse hvad der skete i programmet (state, vigtige værdier og sådan), samt at det var muligt at give den en ny MAC at sende til. Selvfølgelig ville dette kræve ekstra sikkerhed, kodeord og brugernavn, så uønskede personer ikke kunne få adgang.

Der kunne overvejes at fjerne LED’en for temperaturmåling, da dette tager så kort tid at den ikke kan observere at blive tændt og dermed kunne dele LED med TRANSMITTING\_DATA staten. Ellers kunne denne LED bliver brugt i tilfælde af fejl med at sende data, da der lige nu ikke er en indikator for dette.

Lige nu har Arduino’en det problem at hvis state skifter timeren er aktiv og Arduino’en genstarter, f.eks. strømsvigt, så vil timeren bliver deaktiveret. Dermed burde visse værdier, f.eks. en bool der indikerer at timeren er aktiv eller ej, bliver lageret i eepromen og benyttet i programmet.

I forhold til interrupt benyttes to timer interrupts til at sætte state og en til at genaktivere læsning af HC-SR04’en. Det er muligt at sætte HC-SR04’en på en timer og en normal interrupt. Dette kunne forbedre en lille del af programmet, da det nuværende kode ikke vil registrer noget foran sensoren, når programmet læser fra temperatursensoren og sender data. Hvis interrupts blev brugt kunne dette problem bliver løst.

Anskaffet viden, fra denne case, er hvordan det er muligt at omdanne en float til et bytearray, da temperaturkoden returner en float, mens Xbee kræver et bytearray. Der er også blevet øvet state machine og pointer programmering i Arduino/C.

De teknologier der blev benyttet for denne case, virker som de skal. En Arduino Mega 2560 viste sig at være meget vigtig at benytte for denne case, da Xbee’en og debuggeren kræver forskellige serielporte, da de ellers forstyrre hinanden, og de fleste typer af Arduino’er har kun en seriel port.

# (Bilag)

Indholdsfortegnelse

Bilag A - Opstilling**9**

#### Opstilling