

Deliverable B40.1: Simpel Prøvestand Emulator [Daniel]

Oversigt

KravsID	N/A	Opgaveld	B40.1
TestID	N/A	OpgaveNavn	Simpel Prøvestand Emulator
Hel eller delvis implementering af krav	N/A		
Udført af	Daniel	Dato	01-12-2019
Reviewet af		Dato	
Timebox	2	Udv.omr.	Backend-udvikling
Opgave-beskrivelse	Implementer og test to-vejs-kommunikation med Prøvestand med JSON objekter		
Verifikations-beskrivelse	Der er 2 funktionelle tests til verifikation af denne opgave.		
TC-1	Prøvestand Emulator modtager JSON schema med instruktion om at sætte GPIO pin "P8_11" Høj Pass hvis: P8_11 sættes Høj , måles med voltmeter eller lignende, og statusCode med bekræftelse sendes retur og udskrives på Host PC. Fail: hvis ovenstående ikke opnås Test Resultat: Passed		
TC-2	Prøvestand Emulator modtager JSON schema med instruktion om at sætte GPIO pin "P8_11" Lav Pass hvis: P8_11 sættes Lav , måles med voltmeter eller lignende, og statusCode med bekræftelse sendes retur og udskrives på Host PC. Fail: hvis ovenstående ikke opnås Test Resultat: Passed		

Introduktion og opsummering

Formålet med denne opgave er at undersøge muligheden for (proof-of-concept) at udvikle en simpel prøvestand emulator, som kan bruges i forbindelse med:

1. Testware som Team 2 Web kan bruge til at teste/udvikle Webinterface op imod.
2. implementering af kommunikation på BBB, til når Team 3 og 4 skal implementere deres prøvestande.

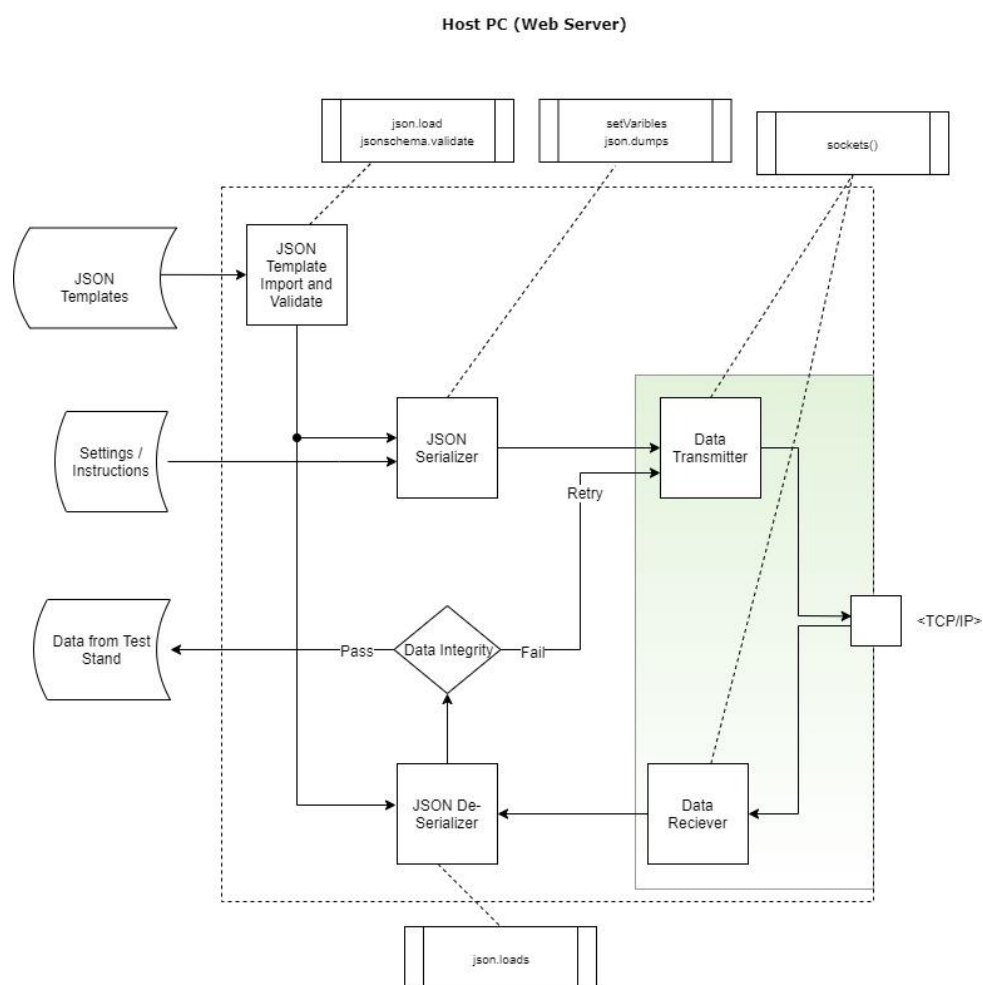
Derudover giver deliverable en demo af at interaktion med BBB hardware er muligt via Python, og umiddelbart nemmere end med C/C++.

Analyse

I opstarten af denne opgave, blev det overvejet om udviklingen skulle tage udgangspunkt i prøvestanden, eller om det skulle være PC/Webserver-delen der skulle udvikles først. Af praktiske grunde blev det besluttet at udvikle sidstnævnte først, og dermed bygge "over mod" prøvestanden. Valget blev at udvikle hele systemet i python.

Det blev identificeret at der skulle bruges nogle specielle json moduler til at håndtere JSON filer, og hhv. oversætte JSON schemas til et python dictionary eller til en streng. Der bruges modulet "json" og "jsonschema" til denne håndtering, som hentes via pythons "pip" pakke håndteringsmodul.

Dernæst blev der lavet en hurtig 'partitionering' i form af et blok-diagram, som viser hvad første antagelse af hvad indholdet af "PC/Webserver" skulle være. Det blev til dette blokdiagram, som viser sammenhængen mellem de specielle json python-moduler og placeringen af funktionalitet, mere end det viser den endelige implementering:



Det ses at diagrammet blot er en overordnet idé at starte ud fra. Der blev arbejdet iterativt gennem hele arbejdsprocessen, og der er derfor ikke dokumentation for hver enkel iteration. Fra dette blokdiagram, blev det analyseret hvorledes strukturen i programmet skulle forløbe. Det blev med tiden til aktivitetsdiagrammet der er vist under design.

Arbejdsprocessen for de første iterationer blev grundlagt af en test-drevet fremgangsmåde, hvor der blev brugt python-modulet "unittest" til at skrive komponenttests samtidigt med at et modul blev implementeret. Der blev derfor udtænkt en teststrategi (nogle testcases), til håndteringen af JSON schemas. Denne liste med tests er vist herunder:

TDD udviklingsmetode af første komponenter til PC/Webserver-delen:

JSON Controller:

x Import Template (__init__):

- x TC1-A Kan importere template
- x TC2-A Accepterer kun kendte templates

x Validere importeret template

- x TC3-A Den Importerede template valideres mod indholdet i den lagrede
- x TC4-A Hvis en type int sættes til type string fejler validering
- x TC5-A Kan validere WebinterfaceTeststandPayload_v1
- x TC6-A Kan importere default template med argument "default"

- Serialize et "JSON" Dict

- x TC7-A Kan serialize dict til string

- De-Serialize en "JSON" String

- x TC8-A Kan De-serialize en string til dict

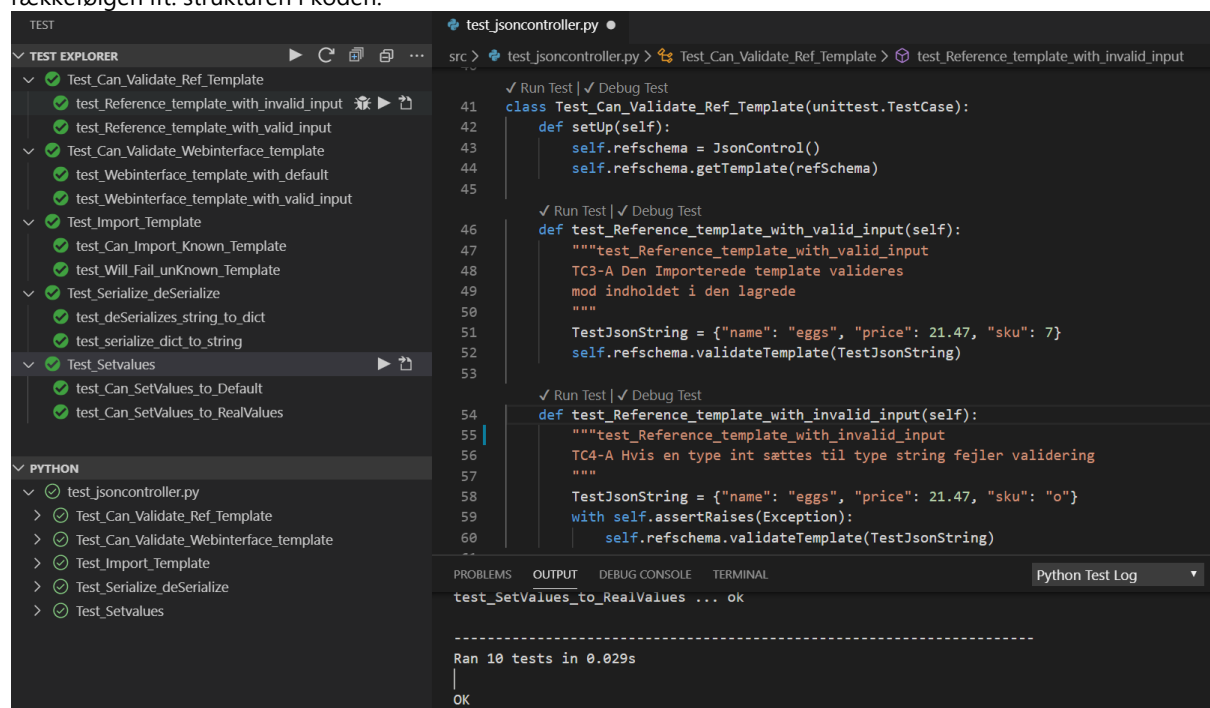
- SetDefaultsValues

- x TC9-A Kan sætte værdierne i et JSON schema til default værdier

- SetValues

- x TC10-A Kan sætte værdierne i et JSON schema til "rigtige" værdier

Her er vist det ovenstående forløb med en testrunner, dog kan "Test explorer" godt finde på at blande rundt i rækkefølgen ift. strukturen i koden:



```
TEST EXPLORER
├── Test_Can_Validate_Ref_Template
│   ├── test_Reference_template_with_invalid_input
│   └── test_Reference_template_with_valid_input
├── Test_Can_Validate_Webinterface_template
│   ├── test_Webinterface_template_with_default
│   └── test_Webinterface_template_with_valid_input
├── Test_Import_Template
│   ├── test_Can_Import_Known_Template
│   └── test_Will_Fail_unKnown_Template
├── Test_Serialize_deSerialize
│   ├── test_deSerializes_string_to_dict
│   └── test_serialize_dict_to_string
├── Test_Setvalues
│   ├── test_Can_SetValues_to_Default
│   └── test_Can_SetValues_to_RealValues
└── PYTHON
    ├── test_jsoncontroller.py
    ├── Test_Can_Validate_Ref_Template
    ├── Test_Can_Validate_Webinterface_template
    ├── Test_Import_Template
    ├── Test_Serialize_deSerialize
    └── Test_Setvalues
```

```
test_jsoncontroller.py
41 class Test_Can_Validate_Ref_Template(unittest.TestCase):
42     def setUp(self):
43         self.refschema = JsonControl()
44         self.refschema.getTemplate(refSchema)
45
46     def test_Reference_template_with_valid_input(self):
47         """test_Reference_template_with_valid_input
48         TC3-A Den Importerede template valideres
49         mod indholdet i den lagrede
50         """
51         TestJsonString = {"name": "eggs", "price": 21.47, "sku": 7}
52         self.refschema.validateTemplate(TestJsonString)
53
54     def test_Reference_template_with_invalid_input(self):
55         """test_Reference_template_with_invalid_input
56         TC4-A Hvis en type int sættes til type string fejler validering
57         """
58         TestJsonString = {"name": "eggs", "price": 21.47, "sku": "o"}
59         with self.assertRaises(Exception):
60             self.refschema.validateTemplate(TestJsonString)
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Python Test Log
test_SetValues_to_RealValues ... ok
Ran 10 tests in 0.029s
OK
```

Man kan se at "sku": er forskellig i testdata, og det forventes at valideringen fejler hvis "sku": får en streng i stedet for en integer.

for måske at nå hurtigt frem til en implementering, blev det efterfølgende valgt at skifte udviklingsmetode til en mere traditionel metode, hvor test forekommer efter implementeringen til sidst.

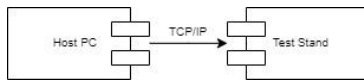
Da PC/webserver-delen var testet OK, og integration med Prøvestand skulle udvikles, blev det identificeret at der skulle bruges et python-modul til at styre GPIO portene på Beaglebone black. Dette modul er "Adafruit_BBIO" som indeholder et sub-modul der hedder GPIO. Dvs. det endelige modul der skal bruges til at styre GPIO portene er "Adafruit_BBIO.GPIO" som kan hentes vha. "pip".

Kravene til det endelige system er at det skal være pålideligt fremfor hurtig performance. Derfor er der ikke sket særlige overvejelser om at optimere kode-strukturen under implementeringen, ej heller efter succesfuld system test.

Design

Systemet er designet ud fra det viste aktivitetsdiagram:

Overordnet Koncept:



Forklaring til farvede pile i aktivitetsdiagrammet:



Command Line Interface

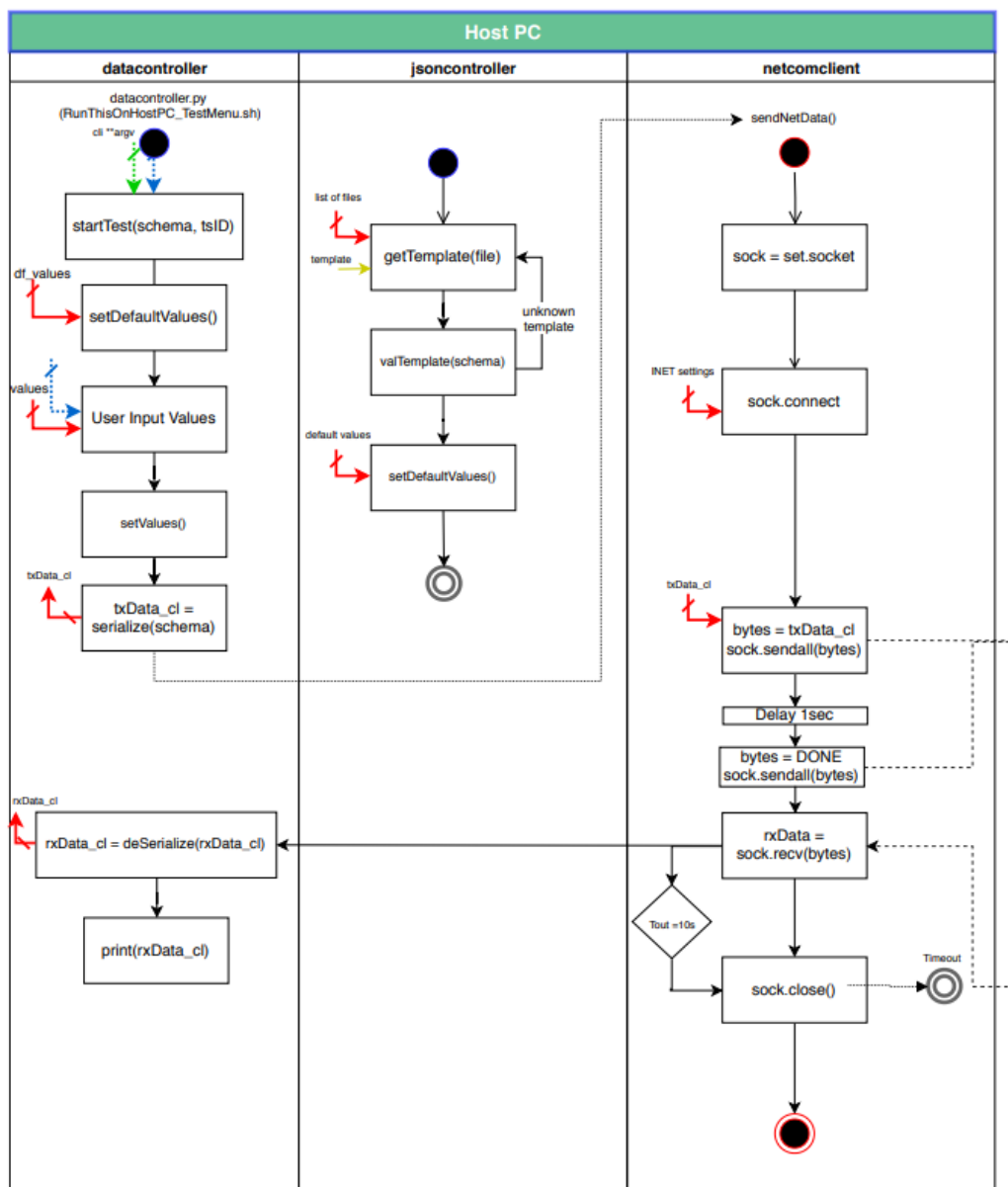
Graphical User Interface (windows)

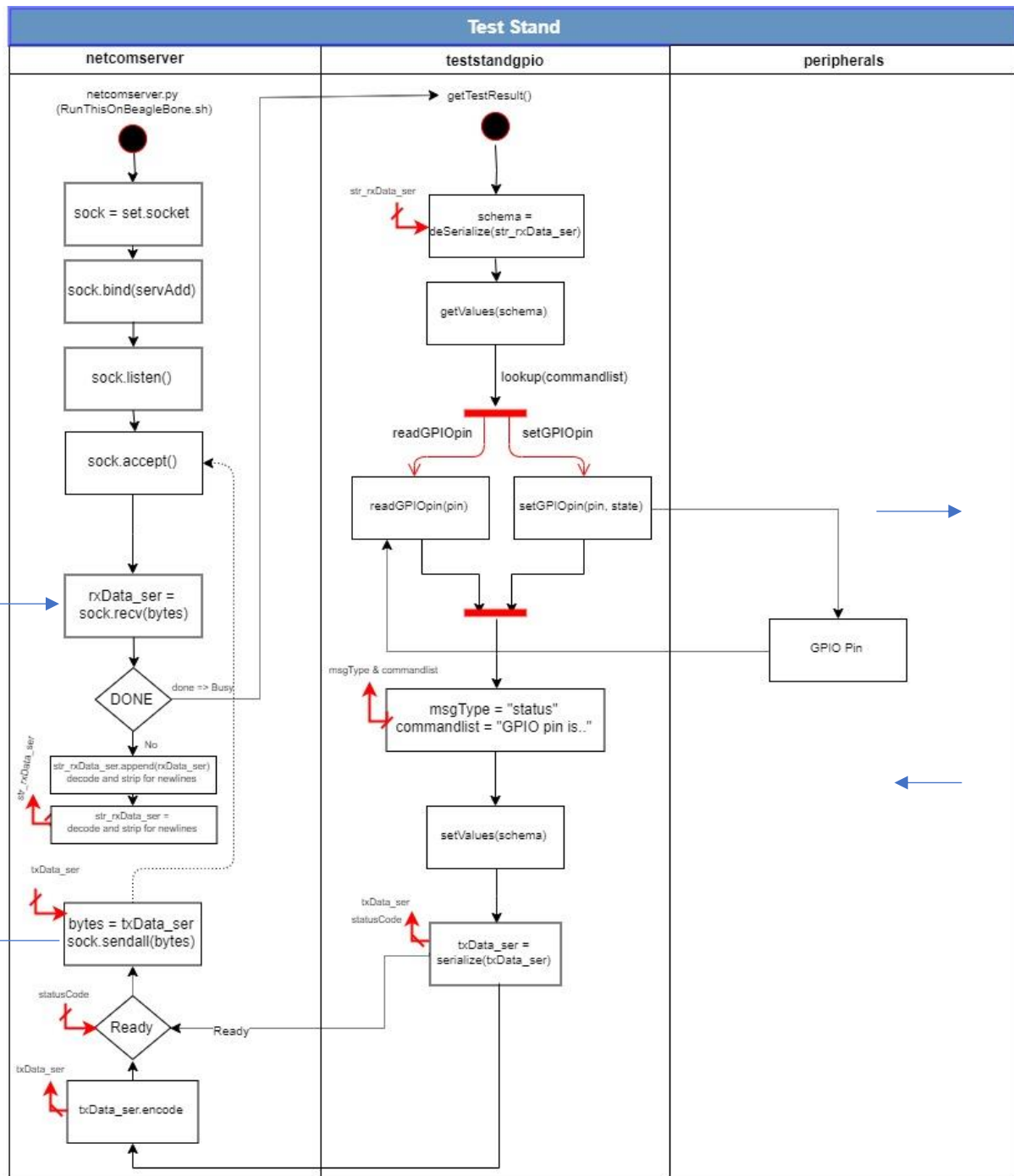
Settings.py er et modul med fælles variable og indstillinger til de øvrige moduler

Volume indeholdende JSON templates som er kendte (accepteret) af systemet.

```

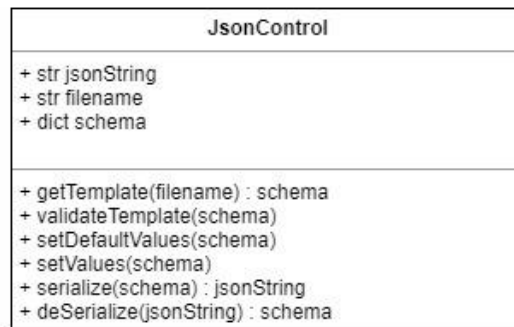
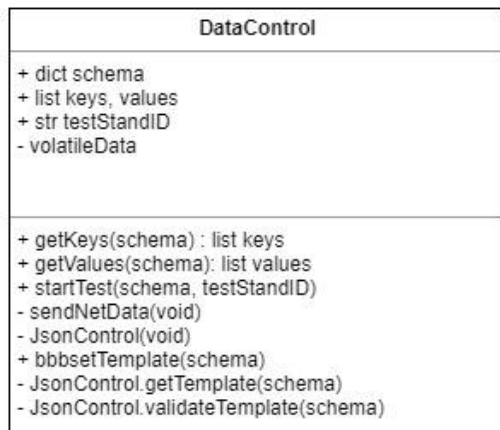
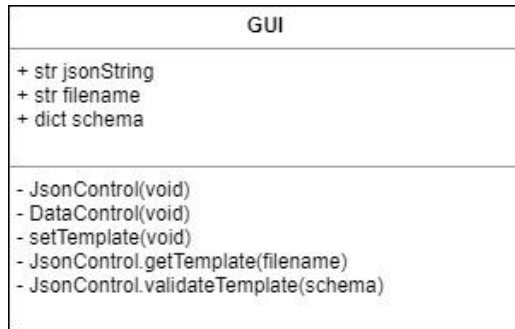
**argv: -teststandid -msgtype -commandlist"
teststandid: "Tsl1" | msgtype: command | commandlist: "setGPIO_P8_11R", "setGPIO_P8_11L"
  
```



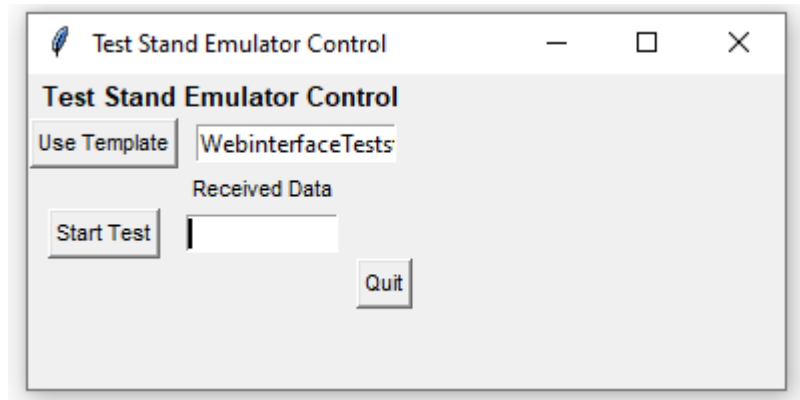


Dette viser hele systemet bestående af Host PC/Web-server som er forbundet med prøvestand via websockets, samt hvilke metoder/funktioner og variabler der benyttes til hver enkelt instruktion eller beslutning. Hver swimlane hvis et modul, hvori der kan være specificeret en bestemt klasse. F.eks. er der til at håndtere JSON schema lavet en klasse der tager sig af det, med de metoder der understøtte netop det som er nødvendigt for implementeringen.

Det resulterede sig i et klassediagram der viser de enkelte klasser, der blev designet for at implementere den ønskede funktionalitet. Det kan ses at "DataControl" arver nogle metoder fra JsonControl, getTemplate og validateTemplate for at holde funktionalitet kapslet inde i hver sin klasse. Men det betyder dog også, at der naturligvis skal være et JsonControl objekt, førend der kan processeres data vha. et DataControl objekt, men det er muligt at validere et JsonSchema uden der er oprettet et DataControl objekt.



GUI -klassen er ikke komplet udviklet, men er brugt i forbindelse med modul test af PC/Webserver, hvor websockets blev loopet tilbage således en terminal kører som PC/Webserver, og en anden terminal kører som Prøvestand. Dette scenarie er gui-klassen lavet til at understøtte, ved at der er en knap der sørger for at hente en JSON template, validere den etc. Hvorefter der kan trykkes på "Start Test", som encoder og sender et test-JSON schema via TCP/IP sockets til prøvestanden (i den anden terminal på localhost), som modtages og decodes, for at sende et svar retur med "GPIO". Dette retursvar kommer til at stå i feltet under "Received Data".



Implementering

Den endelige implementering findes på github [1] under 'testware-1-test-stand-emulator'

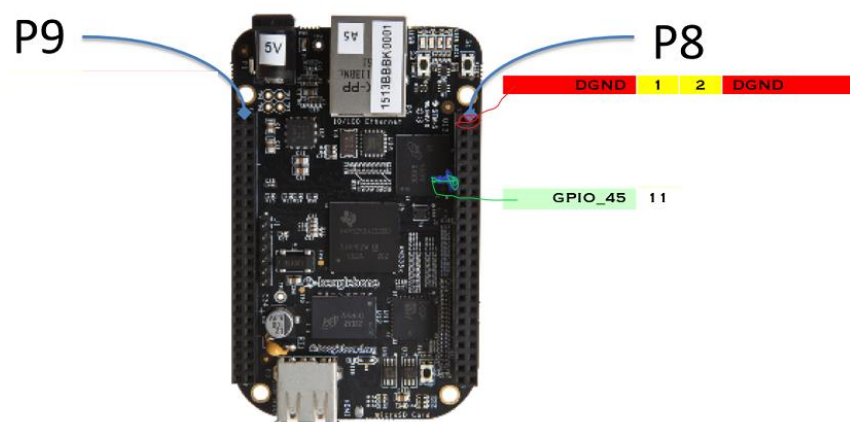
For at starte programmet på kan det køres således:

På PC/Web server køres shell scriptet: `RunThisOnHostPCTestMenu.sh`

På BeagleBone Black køres shell scriptet: `RunThisOnBeagleBone.sh`

Der vil da på PC/Web server fremkomme en test menu i terminalen, som giver mulighed for at sætte GPIO Header P8 pin 11 til Høj eller Lav, med reference til DGND. Der kan også tages Q for at stoppe programmet (stopper dog ikke serveren på prøvestanden, men det er noget som kan implementeres ved senere lejlighed)

Her vises hvor GPIO pin P8_11 er placeret på BeagleBone Black og hvor DGND er placeret, som er de benyttede signaler.



Verifikation + Product Acceptance

Test set-up procedure:

Afhængigheder:

A1: Hent koden med release v1.0 fra [1],

eller `git clone https://github.com/AUTeam2/testware-1-test-stand-emulator.git`

A2: Adafruit_BBIO se evt. [2] (Hentes via pip i forløbet nedenfor)

1 PC Virtual Machine Centos 7 (Host PC) startes op

Åben terminal

`cd til "testware-1-test-stand-emulator"`

`kør # Pip install -r requirements.txt`

2 Tilslut voltmeter/oscilloscope til at måle DC på BBB (BeagleBone Black) P8_11 til DGND.

3 Tilslut BBB Open SDA (usb-mini porten) til en USB port på Host PC (skal give netværksforbindelse på IP: 192.168.7.2) Kan kontrolleres med ping 192.168.7.2 fra host PC. Hvis ikke der skabes denne forbindelse, kan testen ikke forløbe.

Åben terminal

`cd til "testware-1-test-stand-emulator"`

`kør # Pip install -r requirements.txt`

`kør # pip install adafruit_bbio`

4 På Host PC, startes "RunThisOnHostPCTestMenu.sh"

5 På BBB startes "RunThisOnBeagleBone.sh"

6 Test Menu på Host PC fremkommer

Link til video som viser testudførelsen: <https://www.youtube.com/watch?v=Py-jBEhYhZo>

Tabel 1: Tests til verifikation af opgave

Test ¹	Test Steps ²	Pre-requisites ³	Pass-betingelser ⁴	Ref. til data ⁵	Resultat ⁶
TC-1	I testmenu enter "h", vent et kort øjeblik, og aflæs måling.	Test set-up procedure OK	Måles 3,3V / Høj på P8 pin 11	Bilag 1	PASS
TC-2	I testmenu enter "l", vent et kort øjeblik, og aflæs måling.	Test set-up procedure OK	Måles 0V / Lav på P8 pin 11	Bilag 1	PASS

Noter til tabellen:

1	Individuel funktionel test til verifikation af delelement af opgave.
2	Udførlig beskrivelse af test steps.
3	Særlige forudsætninger før udførelse af test.
4	Udførlig beskrivelse af systemets tilstand, output eller lignende for at testen erklæres med resultat "pass".
5	Data, screenshots, m.v. fra udført test findes i dette bilag
6	Pass / Fail

Testresultat

Alle tests er forløbet med resultat "Pass"

Konklusion

Der blev ved hjælp af analyse- og design metoder, implementeret en test stand emulator på en BeagleBone Black.

Referencer

[1]	Kilde Github AU Team 2 [https://github.com/AUTeam2/testware-1-test-stand-emulator]
[2]	Adafruit.BBIO til brug for styring af GPIO m.m. [https://adafruit-beaglebone-io-python.readthedocs.io/en/latest/GPIO.html]

Oversigt over forkortelser

Forkortelse	Forklaring
BBB	BeagleBone Black
Open SDA	serial and debug adapter open standard
TCP/IP	Transmission Control Protocol / Internet Protocol

Review

Reviewer signerer for overordnet godkendelse af dokumentet, ved at udfylde "Review udført af" og "Dato" for udført review i "blå boks" i starten af dokumentet. Nedenstående tabel angiver fokusområder som reviewer skal være opmærksom på.

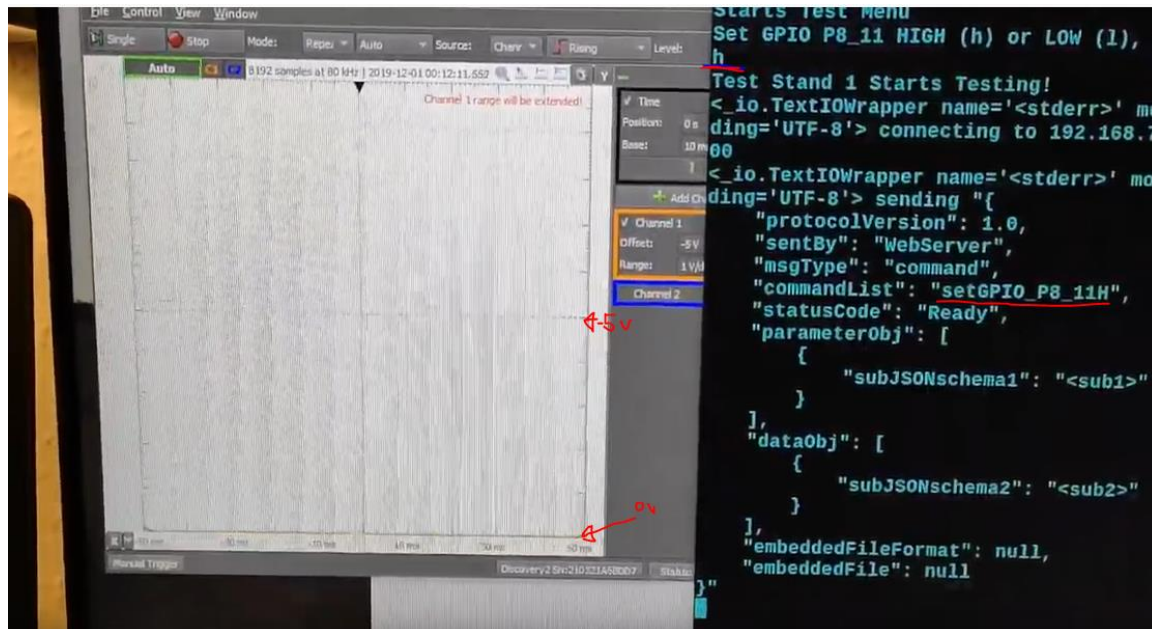
Tabel 2: Reviewtabel

Fokusområde	Check OK (V) eller Mangler (/)
Indhold	
Analyse: Dækker analysen hele opgavens omfang?	
Analyse: Er analysen udført i rimelig dybde?	
Design: Giver designet mening ift. opgaven?	
Design: Er designet baseret på "best practices" inden for fagfeltet?	
Design: Passer designet ind i systemets overordnede arkitektur?	
Implementering: Er implementering udført i overensstemmelse med design?	
Implementering: Overholder implementeringen vores kodestandard og øvrige standarder?	
Tests	
Kan testen reproduceres?	
Er testen nem at reproducere?	
Er testen beskrevet i tilpas omfang (kan den forstås uden særlige forudsætninger)	
Er der beskrevet en test til dokumentet med unikke test ID's?	
Form og layout	
Er definitioner og forkortelser forklaret?	
Er alle refererede dokumenter omtalt i teksten?	
Har alle figurere/tabeller/ligninger et unikt nummer, en forklarende tekst, og er refereret til i en oversigt?	
Formalia	

Er indholdet i dokumentet indenfor dokumentets scope (som beskrevet i introduktion)?	
Er indholdet i dokumentet klart beskrevet og færdigt uden betydelige spring/huller?	
Er indholdet i dokumentet rigtigt?	
Er der kildehenvisninger hvor det er relevant? og er AU's regler for kildehenvisninger i dokumenter efterlevet.	

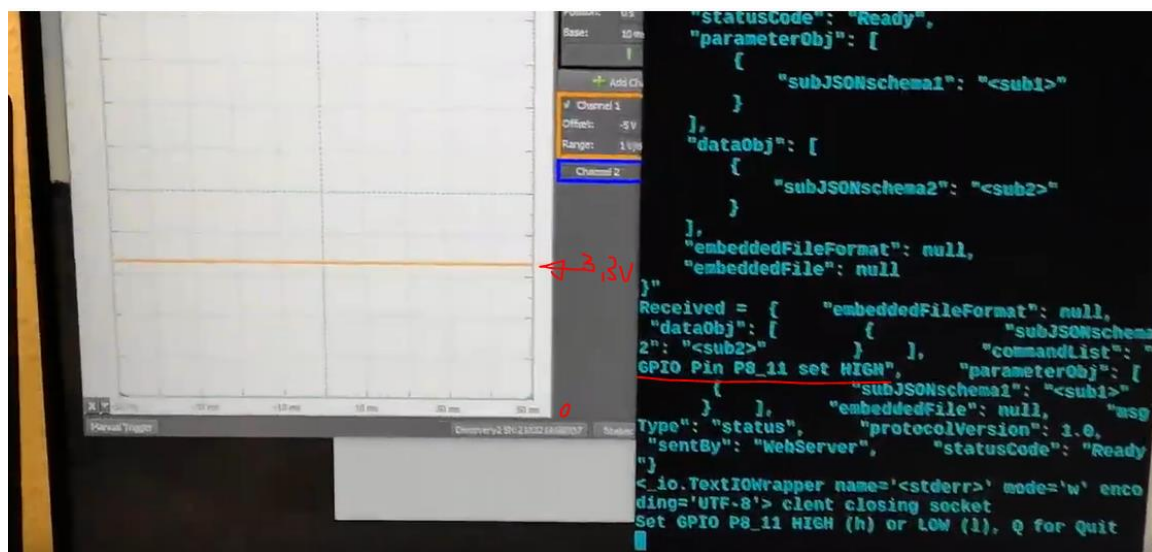
Bilag til verifikation

Der bliver tastet "h" og enter: Processen afvikles, man kan se i JSON skemaet der sendes, at den sender kommando "setGPIO_P8_11H" og at statusCode er "Ready" dvs. test stand er klar.



Efter afvikling, når test menu er klar til input igen:

Det ses på oscilloscop-billedet at den orange streg er ved 3,3V og derfor er GPIO pin blevet høj. Dertil kan man se at JSON skemaet fra prøvestanden skriver "GPIO Pin P8_11 set HIGH", fordi prøvestanden korrekt har tolket at den skulle gøre dette. Det kan man også se, ved at prøvestanden har meddelt at det er en "status" besked, ved at der står "status" i feltet msgType.



På samme måde sætte GPIO pin lav, ved at skrive "l" til testmenuen.

Her er et billede fra testen efter afvikling, hvor GPIO pin er gået fra høj til lav.

