

**Software Dokumentation**

Indholdsfortegnelse

[SWA1 - Applikationsmodel: 3](#_Toc42943864)

[SWA2 - Klassebeskrivelser: 5](#_Toc42943865)

[SWA3 – Software Modulbeskrivelse: 6](#_Toc42943866)

[SWA4 - Testkode for modultest 11](#_Toc42943867)

[Test af count/interrupt: 11](#_Toc42943868)

[Test af loadBits/sendBits 11](#_Toc42943869)

[Test af inputReader() 12](#_Toc42943870)

[Test af binToDec() 12](#_Toc42943871)

[Test af runMode()/interrupt 12](#_Toc42943872)

[Test af enterCode() 13](#_Toc42943873)

[Ændringer af Mode two 13](#_Toc42943874)

[SWA5 - PuTTy UI 14](#_Toc42943875)

[SWA6 – DE2-board 15](#_Toc42943876)

[Baud\_Rate\_Generator 15](#_Toc42943877)

[UART\_transmitter 16](#_Toc42943878)

[Code\_lock\_err: 19](#_Toc42943879)

[Code\_lock\_uart: 23](#_Toc42943880)

[SWA7 - Test af DE2-Board 25](#_Toc42943881)

[Test for code\_lock 25](#_Toc42943882)

[Test for UART\_transmitter 27](#_Toc42943883)

[Test for code\_lock\_uart 28](#_Toc42943884)

## SWA1 - Applikationsmodel:

#### **Use Case 3:**

I Use Case 3 vælger brugeren ”Definer mode 2”, hvor brugeren bliver bedt om at indtaste en kode på DE2-boardet. Når den korrekte kode er indtastet, kan man ændre indstillingerne på mode 2. Nedenstående laves klassediagram med udgangspunkt i Use Case 3, bestående af controller klasse ”HomeProtectionSoftware” og boundry klasserne ”userInterface”, ”XTen” og ”DE2-board”:



Figur 1 – Klassediagram for Use Case 3

Systemet udskriver menuen for brugeren på userInterface, hvoraf brugeren vælger ”Definer mode 2”. HomeProtectionSoftware skal sørge for at de korrekte funktioner bliver initialiseret for Use Case 3, og vil derfor bede brugeren om at indtaste en kode på DE2-boardet. DE2-boardet består af nogle switches og keys, som skal benyttes for at få adgang til at ændre indstillingerne i mode 2. Superbrugeren indtaster kode for at få adgang. Hvis koden indtastes forkert 3 gange låses systemet. Og der skal herefter trykkes på reset knappen på de2-boardet, før Superbrugeren kan forsøge igen. Hvis koden indtastes korrekt, kan brugeren hermed ændre indstillingerne. Systemet initialiseres og videregiver informationerne til XTen, der indstiller dimmer og switch

Følgende sekvensdiagram på figur 14 viser funktionskaldene mellem de forskellige moduler i applikationen:



Figur 2 - Sekvensdiagram for Use Case 3

På nedenstående figur vises et opdateret klassediagram med funktioner og attributter. Her ses hvordan funktionerne i de forskellige klasser hører sammen:



Figur 3 - Klassediagram med members for Use Case 3

## SWA2 - Klassebeskrivelser:

main.cpp (arduino\_sender)

* Main.cpp for SA, har til opgave at opstille den serielle forbindelse mellem computer og arduino, samt initialisere interrupt og CTC signaler og opstille objektet startMode, der er af klassen Transmitter. Main kalder da chooseMode() fra Transmitter.cpp. Herefter ventes der på fuldendelse af chooseMode(). Efter fuldendelse vil main.cpp hoppe tilbage, så chooseMode() igen bliver kaldt.

Transmitter.cpp:

* Trasmitter.cpp har til formål at bestemme hvilket mode der skal udføres, via et kald fra UI: Mode 1, Mode 2, Change mode 2, deactivate Home Protection. Efter valg af mode, skal systemet være i stand til at videresende data fra valgte mode, med mindre ”Change Mode 2” er valgt. Her vil UI bede om oplysninger om ændring fra brugeren.

UI.cpp

* UI.cpp har til formål at kommunikere med brugeren, via PuTTy. Den vil give oplysninger, så som menu, aktiv mode og information om transmittering af data. Den vil yderligere bede om inputs fra brugeren, den returner til Transmitter.cpp.

ModeTwoSettings.cpp

* ModeTwoSettings.cpp har til formål at ændre værdierne for Mode 2. Dette gøres ved input fra Transmitter.cpp, der bestemmer hvordan Home Protection skal opføre sig. Den opbygges i tre dele, to arrays der indeholder 2 bits og et der indeholder 4 bits. De to bits fra de to arrays, bestemmer henholdsvis tænd og sluk. Arrayet med 4 bits bestemmer intensiteten (PWM) af lysstyrken på lampen. Bits for tænd og sluk, samles i arrays fra det respektive array fra Mode 2, hvor plads 2 og 3 bliver udskiftet og svare til tænd og plads 4 og 5 bliver udskiftet og svare til sluk. Arrayet for lysstyrken (PWM) ændres, hvor værdierne på plads 2-5 bestemmer lysstyrken for lampen.

main.cpp (arduino\_receiver)

* main.cpp for Arduino receiveren starter med at initialisere interrupt og PWM-signaler. Herefter indgår den i en løkke, der vil bruges til at indsætte værdier i et array, loadBits, fra signaler der modtages fra SA. Efter arrayet er blevet fyldt, vil værdierne blive indsat i sendBits. Dette gøres af 2 grunde. 1. arrayet loadBits vil konstant opdateres, her vil send bits kun blive opdateret når loadBits er klar med nye værdier. Yderligere, er loadBits af typen volatile, hvilket skaber problemer af vidersendelse af arrayet. Arduinoen vidersender arrayet sendBits til inputReader().

Forskellen på main.cpp for dimmer og switch, består af at dimmer skal modtage 14 værdier, mens switch kun skal modtage 7. Dimmer skal modtage værdier, både for tænd og sluk, men også intensitet af lys.

Receiver.cpp

* Receiver.cpp består af fire funktioner. En til at initialisere interrupts (initInterrupt()), en der initialisere PWM (initPWM), en der kan konvertere binære tal til decimal tal (binToDec()), og en der, ud fra decimaltal værdierne, kan bestemme start og sluk tidspunkt for dimmer og switch, samt lysstyrken for lampen (inputReader()). inputReader() vil yderligere analysere startbit, for at tjekke om det er det relevante array der skal læses. Efter bestemmelse, vil arduinoen udføre sit tildelte job, hvor der ventes på start, bestemmes intensitet og ventes på stop.

Forskellen på Receiver i dimmer samt i switch, består af at dimmer indeholder værdier for både tænd, sluk og intensitet, hvilket giver den 8 værdier der skal processeres, i henholdsvis 2 arrays, med størrelsen 7. Switch indeholder kun 4 værdier, der alle er samlet i et array af størrelsen 7. Dimmeren vil skulle omsætte 8 binære værdier (tænd (2 bits), sluk (2 bits), intensitet (4 bits)), mens switch kun skal omsætte 4 (tænd (2 bits), sluk (2 bits)).

## SWA3 – Software Modulbeskrivelse:

I tabellen nedenfor findes modulbeskrivelser af hver klasse med member functions og parametre:

|  |  |
| --- | --- |
| Klasse: Transmitter | |
| Funktion: | **Indeholder/Benytter:** |
| Transmitter(): Transmitter | modeType: char  settings: char  inputCheck: bool |
| Constructur for klassen Transmitter. Sætter variablerne modeType, settings og inputCheck. | |
| Funktion: | **Indeholder/Benytter:** |
| chooseMode(): void | -Settings: char  -modeType: char  -inputCheck: bool  -modeOneDimmer: char[]  -dimmerPWM: char[]  -modeOneSwitch: char[]  -modeTwoDimmer: char[]  -modeTwoSwitch: char[]  -printMenu(): char  -printInit(): bool  -runMode(char[]) : void  -printSettigs(): void  -setDimmerOn(): void  -setDimmerOff():void  -setDimmerPWM(): void  -setSwitchOn(): void  -setSwitchOff(): void  -collectMode(): void  -printActive(): void |
| chooseMode() har som funktion, at vælge hvilken mode der skal køres. Dette gør den ved hjælp af printMenu() funktionen, der fremviser en menu, hvor brugeren kan vælge et tal mellem 1-4. Herefter vil printInit() blive kørt, der tjekker om værdien er korrekt, og herefter printer, hvilket mode man har ønsket. Ved forkert input, vil printMenu() kaldes igen, og der vælges ny værdi. Ved tast af 1, vil Mode 1 blive kørt. Her vil runMode(char[]) blive kaldt 3 gange, en med vært array tilknyttet Mode 1. Samme vil ske ved tast 2, hvor Mode 2 vil kører på samme måde som mode et, dog med de arrays der er tilknyttet Mode 2. Ved tast af 3, vil der blive kaldt printSettings(). PrintSettings() vil fremvise fire menuer for indstilling af mode 2. Dette gøres ved kald af funktionerne setDimmerOn(), setDimmerOff(), setDimmerPWM(), setSwitchOn() og setSwitchOff(). Herefter samles arraysne for mode 2, ved hjælp af funktionen collectMode(). Sidst vil der blive kaldt funktionen printActive(), der fremviser hvilket mode er blevet aktiveret. Herefter slutter chooseMode(). | |
| Funktion: | **Indeholder/Benytter:** |
| runMode(char[] ): void | sendBits: char[]  printWaiting(): void  printBitSent(): void |
| runMode() står for at sende et binært tal indeholdt i et array som machester-kode. Funktionen benytter en for-løkke til først at checke om en plads i arrayet indeholde et 1 eller et 0. Derudfra indstilles interrupt til enten rising eller falling edge. En while-løkke, hvori printWaiting() køres, benyttes til at vente på at et interrupt er indtruffet, hvorefter for-løkken foretager samme procedure med det næste element. | |

|  |  |
| --- | --- |
| Klasse: initArduino | |
| Funktion: | **Indeholder/Benytter:** |
| initInterrupt(): void |  |
| Her initialiseres interrupt registrene. Vi benytter her INT4, indstillet til rising edge som default. | |
| Funktion: | **Indeholder/Benytter:** |
| initCTC(): void |  |
| Her initialiseres CTC. Vi benytter OC3A, indstillet til mode 4, CTC. Clocken er sat til Match Down Counting, med en prescaler på 1. | |
| Funktion: | **Indeholder/Benytter:** |
| initPortB(): void |  |
| Initialisere Port B LED’er på Arduino shield til output. | |
| Funktion: | **Indeholder:** |
| initUART(): void |  |
| Initialiserer UART. Der indstilles til normal ,asynchronious mode med 1 stop bit. | |

|  |  |
| --- | --- |
| Klasse: modeTwoSettings | |
| Funktion: | **Indeholder/Benytter:** |
| setDimmerOn(char ): void | Settings: char  dimmerOn: char[2] |
| SetDimmerOn(char) modtager fra chooseMode() variablen settings, med en værdi mellem 1 – 4. SetDimmerOn består af en switch, der styres af settings, der vær har en værdi på given plads i dimmerOn[]. Arrayet dimmerOn[] sættes til værdier tilsvarende given setting værdi. | |
| Funktion: | **Indeholder/Benytter:** |
| setDimmerOff(char ): void | Settings: char  dimmerOff: char[2] |
| serDimmerOff(char) har samme funktion som setDimmerOn(). Det er dog i arrayet dimmerOff der ændres på. | |
| Funktion: | **Indeholder/Benytter:** |
| setDimmerPWM(char[], char ): void | Settings: char  dimmerPWM: char[4] |
| setDimmerPWM(char[], char ) modtager en pointer til arrayet dimmerPWM og variablen settings der ligger mellem 1-10. Her er der, ligesom for setdimmerOn en switch, der bestemmes af settings. Forskellen ligger på at der nu er 10 muligheder, og dimmerPWM ændres direkte via en pointer og at der er 4 værdier der ændres på: plads 2 til 5. | |
| Funktion: | **Indeholder/Benytter:** |
| setSwitchOn(char ): void | Settings: char  switchOn: char[2] |
| setSwitchOn(char ) har samme funktion som setDimmerOn(). Det er dog i arrayet switchOn der ændres på. | |
| Funktion: | **Indeholder/Benytter:** |
| setSwitchOff(char ): void | Settings: char  switchOff: char[2] |
| setSwitchOff(char ) har samme funktion som setDimmerOn(). Det er dog i arrayet switchOff der ændres på. | |
| Funktion: | **Indeholder/Benytter:** |
| collectMode(char[] ,char[] ): void | modeTwoDimmer: char[]  modeTwoSwitch: char[]  dimmerOn: char[2]  dimmerOff: char[2]  switchOff: char[2]  switchOn: char[2] |
| collectMode(char[] ,char[]) sørger for at modtage 2 pointers til modeTwoDimmer og modeTwoSwitch. Her vil collectMode(char[] ,char[]) indsætte værdierne der befinder sig i arraysne dimmerOn, dimmerOff, switchOn og switchOff. Værdierne for ”on” arraysne, vil indsættes på plads 2-3 og værdierne for ”off” arraysne vil indsættes på plads 4-5, på deres respektive pladser i modeTwoDimmer og modeTwoSwitch. | |

|  |  |
| --- | --- |
| Klasse: UI | |
| Funktion: | **Indeholder/Benytter:** |
| printMenu(): char | inputOne: char  serialFlush(): void |
| Funktionen skal printe en menu ud i vores konsol, og tage en input som den retunere. Funktionen serialFlush() køres for at forberede et input. En menu prints vha. Serial.println() funktionen, hvorefter Serial.read() bruges til at indlæse et input ind i variablen inputOne. Hvis inputOne ikke er lig '0', bliver konsolen med Serial.write(). | |
| Funktion: | **Indeholder/Benytter:** |
| printSettings(): char | Module: char  inputTwo: char |
| printSettings(): char skal printe en undermenu i konsolen. Variablen module bliver indsat i en switch() | |
| Funktion: | **Indeholder/Benytter:** |
| printInit(): bool | inputOne: char  printWrongInput(): void |
| Benyttes til at printe en besked i UI’en ud fra hvilken mode der er blevet valgt i menuen. Sætter desuden PortB, så LED’erne på Arduino shield passer til den valgte mode. | |
| Funktion: | **Indeholder/Benytter:** |
| enterCode(): void | timer: int  serialFlush(): void |
| Her bedes brugeren indtaste koden på DE2-boardet, så systemet kan låse op for ”mode 3”. | |
| Funktion: | **Indeholder/Benytter:** |
| printWaiting(): void | waitValue: char |
| Sender en besked ud til konsol vindue, der viser at systemet venter på at kunne sende bits videre til arduino modtageren. | |
| Funktion: | **Indeholder/Benytter:** |
| printActive(): void | inputOne: char  inputTwo: char  waitValue: char  module: char |
| Resetter konsollen I Putty. Printer desuden i UI konsol afhængig af hvilken mode der er blevet aktiveret. Slukker for LED’er på Arduino shield. | |
| Funktion: | **Indeholder/Benytter:** |
| printBitSend(): void | waitValue: char |
| Printer en besked I UI konsolen hver gang en bit bliver sendt. | |
| Funktion: | **Indeholder/Benytter:** |
| printWrongInput(): void |  |
| Printer en besked I UI konsolen der indformere om ugyldigt input. | |
| Funktion: | **Indeholder/Benytter:** |
| serialFlush(): void |  |
| Renser systemet for inputs via PuTTy. Ved få tilfælde, vil systemet læse ekstra indtastninger, som f.eks. ”enter”. serialFlush() sørger for at disse inputs ikke går videre, men bliver ” skyllet” ud. Herved er der gjort klar til næste ønskede input. | |

Idet en stor del af funktionaliteten af vores system afhænger af brugen af avr interrupts, finder vi det relevant også at indkludere en modulbeskrivelse af interrupt. Det er her klart for os at INT4 ikke er en klasse og ISR(INT4\_vect) ikke er en funktion:

|  |  |
| --- | --- |
| Klasse: INT4 | |
| Funktion: | **Indeholder/Benytter:** |
| ISR(INT4\_vect) |  |
| Interruptet står for at indstile CTC til det 120kHz burst vi ønsker. Registret OCR3A gives værdien 65 for at starte outputtet på 120kHz. EICRB sættes til low-input, for at forberede regsitret til enten at sende ved rising edge eller falling edge. Et delay på 1ms køres for at sikre at burst-timeren passer overens med zeroCrossing, hvorefter global interrupt bliver disabled. | |

## SWA4 - Testkode for modultest

### Test af count/interrupt:

Ved brug ISR(INT2\_vec) interrupt, vil vi kunne teste om interruptet er aktivt og virker. Ved vært interrupt, vil count stige med 1, hvilket kan ses på arduino sheildets LED’er.

1. while (count < 7) {
2. switch (count) {
3. case 1: PORTB = 0b00000001;
4. break;
5. **case 2: PORTB = 0b00000010;**
6. break;
7. case 3: PORTB = 0b00000100;
8. break;
9. case 4: PORTB = 0b00001000;
10. **break;**
11. case 5: PORTB = 0b00010000;
12. break;
13. case 6: PORTB = 0b00100000;
14. break;
15. **case 7: PORTB = 0b01000000;**
16. break;
17. case 8: PORTB = 0b10000000;
18. break;
19. }
20. **}**

### Test af loadBits/sendBits

For at teste om der er kommet nogle værdier ind i loadBits (og derfor også sendBits) sættes der et for loop til at køre igennem alle værdierne. Siden værdierne for loadBits og sendBits skal være ens, kan de begge testes på samme tid. De værdier der sættes ind, kommer fra count – plads 0 = count = 0, plads 1 = count = 1 osv.

1. for (int n = 0; n < 7; n++) {
2. switch (sendBits[n]) {
3. case 0: PORTB = 0b00000000;
4. break;
5. **case 1: PORTB = 0b00000001;**
6. break;
7. case 2: PORTB = 0b00000010;
8. break;
9. case 3: PORTB = 0b00000100;
10. **break;**
12. }
14. switch (loadBits[n]) {
15. **case 4: PORTB = 0b00001000;**
16. break;
17. case 5: PORTB = 0b00010000;
18. break;
19. case 6: PORTB = 0b00100000;
20. **break;**
21. case 7: PORTB = 0b01000000;
22. break;
24. }
26. }

### Test af inputReader()

For at teste om inputReaderen er aktiv, vil alle LED’erne på arduinoen lyse. Herved ved vi at funktionen bliver kaldt.

1. PORTB = 255;
2. delay(50000);

### Test af binToDec()

Ved brug af et array, kan vi teste om binToDec omskriver de binære værdier om til dec. Decimaltallet vil da vises på arduinoens LED’er

1. char binToDecTest[7] = {0, 1, 1, 0, 1, 1, 0};
2. PORTB = binToDec(loadBits, 2, 3);
3. delay(100000);

### Test af runMode()/interrupt

For at test om runMode() kan køre, kræves det at simulere et interrupt. Dette kan gøres via arduino sheildet, da vi kan gøre brug af ISR(INT2\_vec) der kan aktiveres af SW2 knappen. Herved kan vi se, om koden faktisk for sendt bits via interrupts.

1. void Transmitter::runMode(char sendBits[]) {
3. for (bits = 0; bits < 7; bits++) {
4. sei(); //global interrupt enable
5. **if (sendBits[bits] == 1) {**
6. EICRB = 00000010;
7. }
8. else if (sendBits[bits] == 0) {
9. EICRB = 00000011;
10. **}**
11. while (active == 0) {ui.printWaiting();}
12. active = 0;
13. ui.printBitSent();
14. }
15. **}**

### Test af enterCode()

Test af enterCode() simuleres ved at lave et kode system på arduinoen, da det ikke er muligt at bruge et DE2-board. Koden skal skrives ind af brugeren. Hvis der tastes rigtigt, vil programmet udskrive ”Correct code!” og fortsætte. Tastes der forkert, vil programmet udskrive ”Wrong code! Try again!”, og brugeren skal prøve igen.

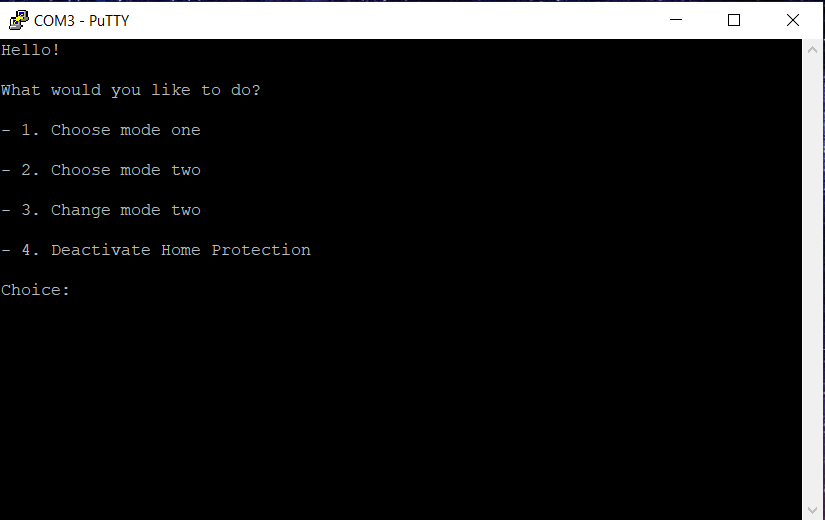
1. void UI::enterCode() {
2. this->serialFlush();
4. int lockValue = 0;
6. Serial.print("Please enter your test code: ");
8. while (lockValue == 0) {
9. if (Serial.available() > 0) {
11. lockValue = Serial.read();
13. }
14. }
16. if (lockValue == '5') {
17. Serial.println("Correct code!");
18. lockValue = 0;
19. }
21. else {
22. Serial.println("Wrong code! Try again!");
23. this->enterCode();
24. }
26. }

### Ændringer af Mode two

Ved at se på ændringerne af Mode two, vil vi printe de værdier der vil ligge i arrayet for Mode two. Arrayet har som standart de samme værdier som Mode one. Det der sker er, at der bruges Mode three, til at omskrive Mode two. Efter omskrivning, vil vi printe værdierne for Mode one (De værdier Mode two startede med) og værdierne for Mode two. Hvis værdierne ikke er ens, er der altså blevet ændret på Mode two.

1. */\**
2. *modeOne dimmer: 0, 0, 0, 1, 1, 1, 1*
3. *modeTwo dimmer: 0, 0, 0, 0, 0, 0, 1*
4. *\*/*
5. for (int test = 0; test < 7; test++) {
6. if (modeOneDimmer[test] == 1)
7. **Serial.print("1, ");**
9. else if (modeOneDimmer[test] == 0)
10. Serial.print("0, ");
11. }
13. Serial.println("**\n**");
15. for (int test = 0; test < 7; test++) {
16. if (modeTwoDimmer[test] == 1)
17. **Serial.print("1, ");**
19. else if (modeTwoDimmer[test] == 0)
20. Serial.print("0, ");
21. }
23. }

## SWA5 - PuTTy UI



## SWA6 – DE2-board

Systemet er forbundet til et Altera DE2-board, som anvendes i forbindelse med Use Case 3, hvor brugeren indstiller mode 2. Når brugeren vælger ”Definer Mode 2”, beder systemet om at få en kode fra superbrugeren. Som tidligere nævnt er koden anvendt for at, det ikke er alle der har adgang til denne funktion. Det er dermed superbrugeren der skal indtaste en kodesekvens på DE2-Boardet for at låse ”Definer Mode 2” op. DE2-boardet skal heraf bestå af en code\_lock, UART og baudrate generator. Da der tidligere i DSD-kurset er blevet lavet journalopgaver med UART og code\_lock, kan disse anvendes som skabelon i dette tilfælde. Koden skrives i VHDL, da det er det sprog, der er blevet anvendt til at arbejde med i forbindelse med FPGA-boardet.

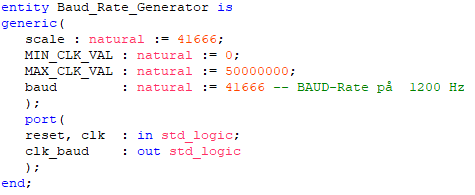
## 

### Baud\_Rate\_Generator

Baud.vhd

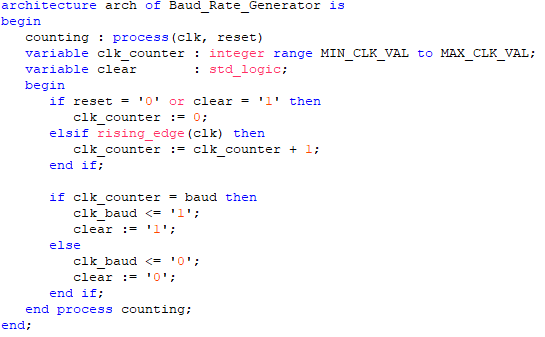
* Baud.vhd består af entiteten og arkitekturen for Baud\_Rate\_Generator. Formålet med denne implementering er at man selv kan indstille en passende hastighed, for hvor hurtigt data skal overføres fra vores transmitter.

Baud\_Rate\_Generator implementeres i de to følgende kodeafsnit, hvor entiten og arkitekturen er opbygget.



Figur 4 - Entity for Baud\_Rate\_Generator

Ved brug af generic, kan baudraten indstilles til en værdi som ønsket. Denne værdi skal hænge sammen med reciever enhedens læserate, så der ikke opstår malplacerede bits. Under test af implementeringen af code\_lock er der benyttet en baudrate på 1200 Hz, der gør det nemmere og mere overskueligt at læse de bit der udsendes.



Figur 5 - Architecture for Baud\_Rate\_Generator

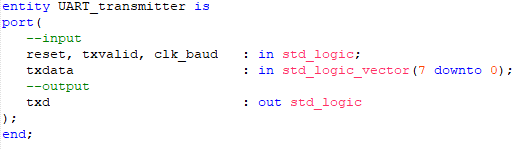
I arkitekturen anvendes der to yderligere variable, clk\_counter og clear. Disse benyttes i forb. med if-else og elsif-sætninger der sammen har til formål at sætte værdien af clk\_baud, som er output og bliver benyttet af UART\_transmitteren. Det kan ses, at hvis der opstår rising\_edge(clk) gentagne gange, så bliver clk\_counter = baud hvorefter clk\_baud sættes ’1’, og kort efter cleares den og clk\_counter samt clk\_baud sættes tilbage til ’0’. clk\_baud benyttes i efterfølgende afsnit om UART\_transmitter.

### UART\_transmitter

UART\_transmitter.vhd

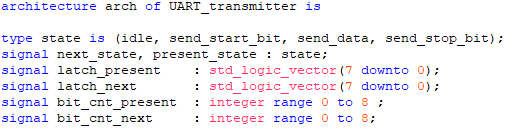
* Formålet med UART\_transmitteren er at der kan oprettes UART forbindelse mellem DE2-Board og Arduino, hvorved der kan sendes et antal databit, der skal initiere en ønsket mode på arduino’en.

UART\_transmitteren implementeres med en entitet og arkitektur for denne, det kan ses i de følgende kodeafsnit



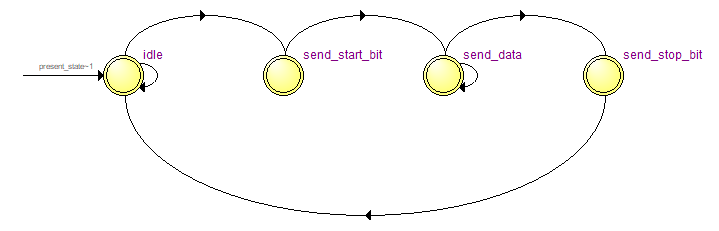
Figur - Entity for UART\_transmitter

Ovenfor ses entiteten UART\_transmitter, denne indeholder in-/output-porte, som vi specificere i arkitekturafsnittene nedenfor samt bruges disse porte i vores samlede .vhd fil ”code\_lock\_uart”



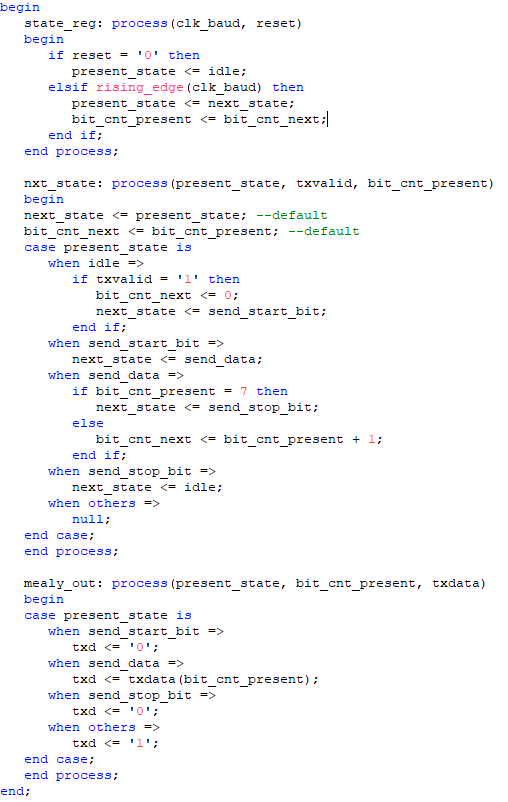
Figur 7 - Architecture for UART\_transmitter (del 1)

Første arkitekturafsnit ses ovenfor, hvor der implementeres 4 states, der udgør en samlet state machine, kan ses på Figur 8.



Figur 8 – State Machine View for UART\_transmitter

I ovenstående state machine illustreres funktionaliteten af UART\_transmitteren. Der består af de fire states (idle, send\_start\_bit, send\_data og send\_stop\_bit) disse initieres vha. koblingen mellem input og signalerne (latch\_present, latch\_next, bit\_cnt\_present, bit\_cnt\_next samt state signalerne next\_state og present\_state).



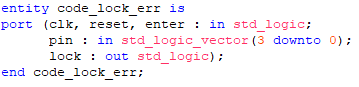
Figur 9 - Architecture for UART\_transmitter (del 2)

I ovenstående kodeafsnit ses den anden del af arkitekturen for UART\_transmitteren, her ses 3 processer (state\_reg, nxt\_state og mealy\_out).   
Den første proces tjekker hvornår der opstår rising\_edge fra baud\_rate clocken(clk\_baud) som blev implementeret i afsnittet om Baud\_Rate\_Generatoren, ved rising edge initieres process ”nxt\_state”. Derudover tjekker state\_reg også om der trykkes på reset-knappen, hvorefter present\_state så sættes til idle   
Den anden proces ”nxt\_state” indeholder funktionaliteten af de 4 states vist på figur 8. Når present\_state er ”idle”, så skal txvalid gå høj = ’1’ før den næste state, send\_start\_bit, initieres og hele processen for transmitteren aktiveres. Send\_start\_bit aktiveres og efter startbit er sendt bliver værdien for txdata transmitteret under send\_data som kan ses i 3. process ”mealy\_out”. Som default vil transmitteren sende et ’1’ signal ud, initieres sekvensen af state machinen korrekt vil den sende ’0’ ud.

### Code\_lock\_err:

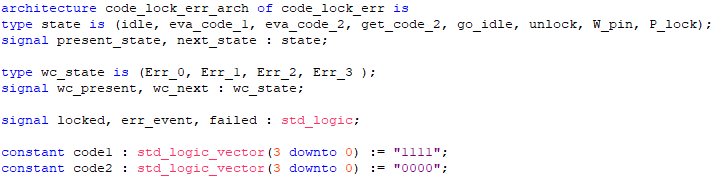
Code\_lock\_err.vhd

* Code\_lock\_err.vhd er essensen bag vores code\_lock\_uart. Formålet med dette afsnit er, at man kan bruge DE2-Boardet som kodelås. Code\_lock\_err er implementeret således, at der skal indtastes en kodesekvens på DE2-Boardet, som skal ved korrekt kode låse op for kodelåsen for at give superbrugeren adgang til at rette i indstillingerne på dimmeren og switchen, dette vil blive uddybet i afsnittet om code\_lock\_uart.



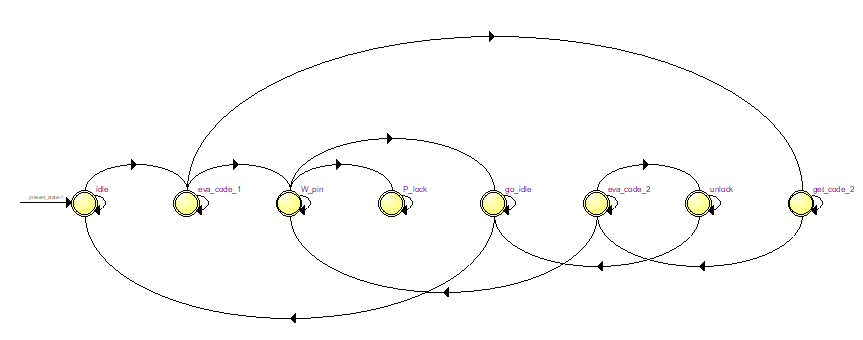
Figur 10 - Entity for code\_lock\_err

Entiteten for code\_lock\_err ses på figur 12, her instantieres in-/output-portene som der bruges i code\_lock\_uart og i arkitekturen nedenfor.



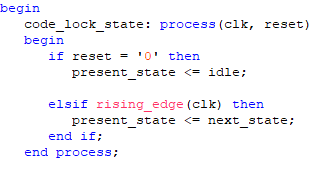
Figur 11 - Architecture for code\_lock\_err (del 1)

På figur 11 ses første afsnit af arkitekturen. Her implementeres der endnu en state machine. Vises på figur 12. Derudover kan der også ses, hvordan de to koder er indstillet, den første kode er indstillet til ’1111’ og den anden kode ’0000’.



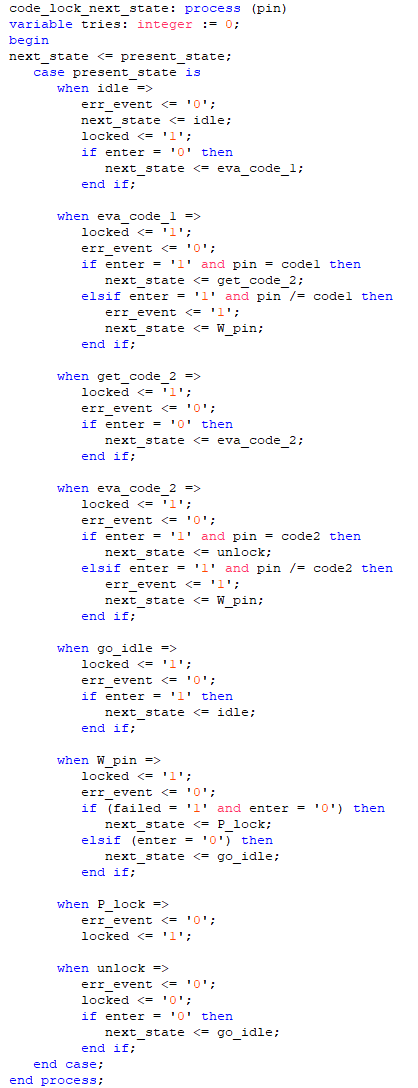
Figur 12 - State Machine View af code\_lock

Code\_lock\_err virker på den måde at den efter korrekt udførsel af kodesekvens fra superbrugeren, vil deaktivere kodelåsen. Derudover er der blevet gjort krav på, at kodelåsen kan blive sat i permanent låst tilstand, hvis en uvedkommende person prøvede på at indtaste koden, eller hvis superbrugeren selv indtaster koden forkert 3 gange. Det kan ses på figur 12. Hvor der efter begge evaluate\_code(eva\_code\_1 og eva\_code\_2) er en pil hen til W\_pin, som betyder, at det er en forkert kode, der er indtastet. Hvis koden indtastes forkert, vil den gå i P\_lock hvorved kodelåsen vil låses indtil der trykkes på reset-knappen, hvor processen startes forfra. Evalueres første kode til at være korrekt, afventes den anden kode og hvis begge er indtastet rigtig, vil den gå over i ”unlock”-state, hvorefter kodelåsen deaktiveres.



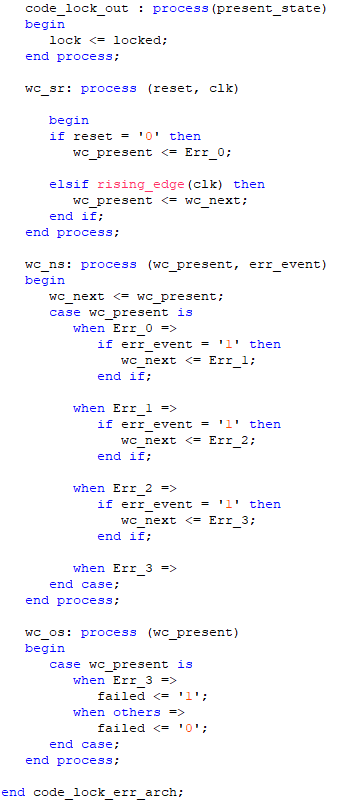
Figur 13 - Architecture for code\_lock\_err (del 2)

På figur 13 vises første process, der tjekker hvornår der opstår rising edge for clk, ved rising edge sættes present\_state til next\_state. Derudover tjekker code\_lock\_state også om der trykkes på reset hvor processen sættes idle/ starter forfra.



Figur 14 - Architecture for code\_lock\_err (del 3)

I arkitekturafsnittet på figur 14, ses hvordan present\_state er stillet op med case-sætninger. Disse case-sætninger omhandler blot, hvad der er skrevet tidligere om code\_lock\_err’s virkemåde. En tilføjelse er, at der ses hvilke signaler der bliver indstillet, for hver state i state machine view, figur 12. F.eks. sættes locked <= ’0’ når programmet går i ”unlock”-state hvor det tidligere er nævnt, at kodelåsen bliver deaktiveret.



Figur 15 - Architecture for code\_lock\_err (del 4)

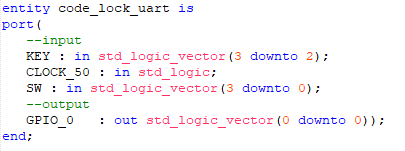
Den 4. del af arkitekturen indeholder code\_lock\_err’s output som sættes lig værdien af locked. Desuden er der også implementeret err\_states heri. Disse 2 processer der omhandler wrongcode, (wc\_ns og wc\_os) skal blot forstås som at koden indtastes forkert 3 gange, og hvis dette sker sættes failed <= ’1’ hvorefter programmet går i staten ”P\_lock”, som låser kodelåsen permanent indtil der trykkes på reset knappen.



### Code\_lock\_uart:

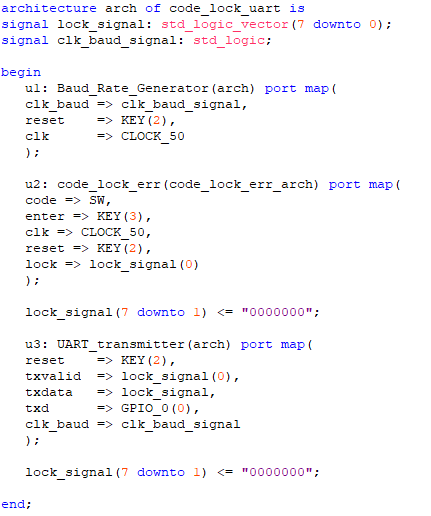
Code\_lock\_uart.vhd

* Code\_lock\_uart.vhd består af de 3 ovenstående .vhd filer, her er de 3 filers funktionalitet blot samlet i én fil, for at vi kan lave en samlet indstilling af in-/output-portene for at de kan blive tildelt en pin på DE2-Boardet, som kan benyttes i praksis.



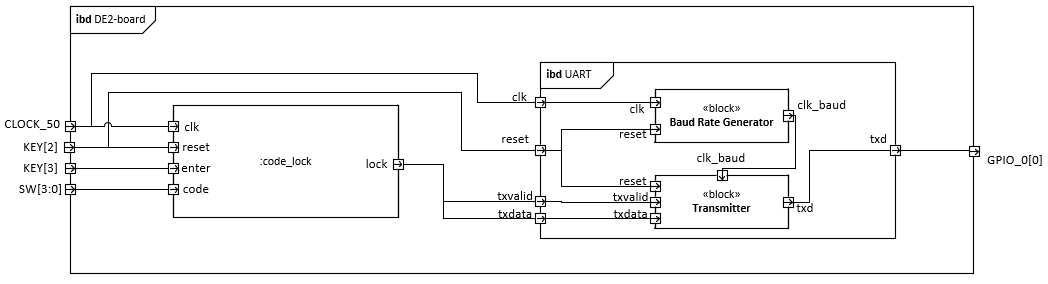
Figur 16 - Entity for code\_lock\_uart

Entiteten for code\_lock\_uart vises på figur 14. Her bemærker vi at KEY[3] og KEY[2] samt SW[0 til og med 3] kan benyttes af superbrugeren som inputs. Derudover sættes output til GPIO\_0 så det er denne pin der skal forbindes med en ledning til Arduino for at etablere UART-kommunikationen.



Figur 17 - Architecture for code\_lock\_uart

I ovenstående kodeafsnit er arkitekturen for den samlede code\_lock\_uart. Her er den samlede arkitektur for Baud\_Rate\_Generator, code\_lock\_err og UART\_transmitteren implementeret. Her vises hvordan de forskellige in-/output-porte er port mappet, altså hvilke pins på DE2-Boardet, som de er blevet tildelt. Derudover tilføjes der to signaler, clk\_baud\_signal og lock\_signal. Clk\_baud\_signalet er oprettet for at få baud\_rate over i transmitterdelen, så der er bestemt en datasignaleringshastighed. Lock\_signalet oprettes for at lave en kobling mellem code\_lock\_err’s output ”lock” og UART\_transmitterens ”txdata og txvalid” inputs, se figur 18.



Figur 18 - IBD for DE2-Board

Desuden skal der gøres opmærksom på at txdata er et 8bit datasignal, men at der samtidig kun skal benyttes 1 bit til at determinere om kodelåsen er aktiveret eller deaktiveret. Dermed er lock\_signalets 7 første bit sat lig ’0’. Så der kun kan aflæses fra den sidste bit, (0).



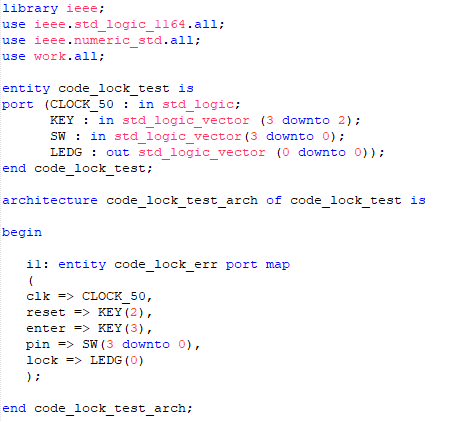
Dermed kan man indstille det sidste bit, til at være lig med outputtet fra code\_lock\_err, nemlig om DE2-Boardet er låst ’1’ eller låst op ’0’



## SWA7 - Test af DE2-Board

### Test for code\_lock

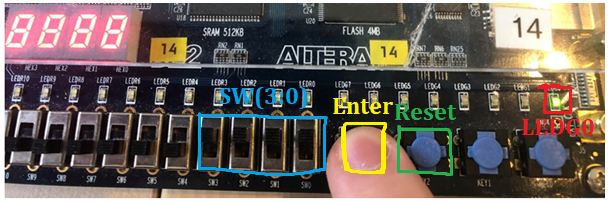
Herunder ses testbenchen af code\_lock\_err.



Figur - Kode for tester code\_lock\_test

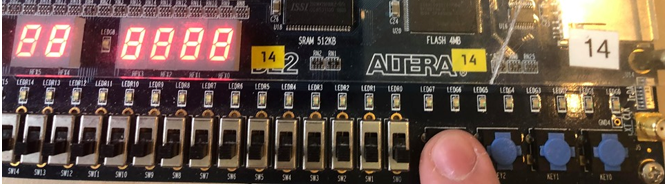
I testbenchen implementers portmapping af entiteten og arkitekturen for code\_lock\_err.

På følgende figurer vises der test for code\_lockens funktionalitet på DE2-boardet:



Figur Test på De2-Board her indtastes code1 ”1111”, det ses at LEDG[0] lyser så lock = ’1’

På figur 22 bliver den første korrekte kode (1111) indtastet, efterfulgt af enter-knappen. LED´et lyser, da programmet er i en locked tilstand.



Figur Test på DE2-Board her indtastes code2 ”0000”, hvor LEDG[0] stopper med at lyse så unlock er initieret

Den anden korrekte kode (0000) bliver nu indtastet efterfulgt af enter-knappen. LED´et stopper med at lyse, hvilket vil sige at programmet ændrer tilstand fra locked til unlocked.

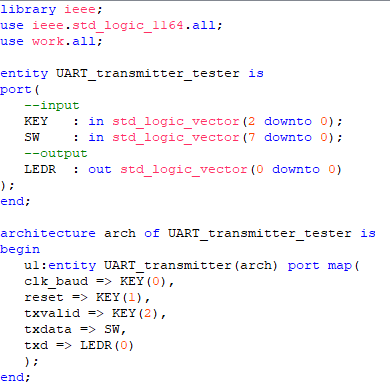


Figur 22 Test på DE2-Board her tester vi om permanently locked virker ved at indtaste rigtig kodesekvens 1&2 hvor inden der er trykket forkert kode 3 gange.

På figur 24 er der blevet testet for om permenantly locked virker. Hvis koden bliver indtastet forkert 3 gange, bliver programmet låst, og reset-knappen skal anvendes før man kan prøve igen. Hvis der ikke trykkes på reset-knappen, vil man ikke kunne låse op. I ovenstående tilfælde er der blevet indtastet forkert kode 3 gange, og programmer er hermed blevet låst. Den rigtige kode er herefter blevet indtastet, men dog lyser LED´et stadig. Dette er et resultat af, at reset-knappen ikke er blevet brugt. Reset-knappen skal derfor benyttes, hvis programmet bliver låst, før den korrekte kode kan indtastes.

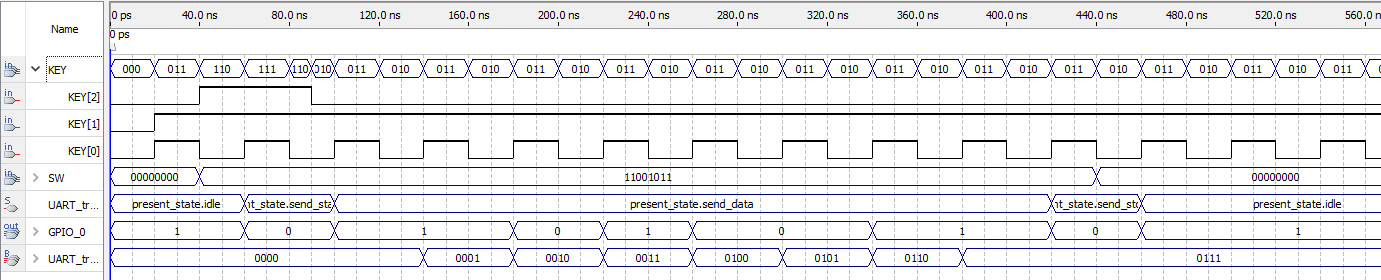
### Test for UART\_transmitter

Testbenchen for UART\_transmitteren ses herunder, her vises hvordan entiteten for UART\_transmitteren portmappes på DE2-Boardet



Figur 23 - Kode for tester UART\_transmitter\_tester

For at teste UART’en er der benyttet Quartus’ simuleringsværktøj til at oprette en funktionel simulering, hvorpå man kan se Transmitterens virkemåde.

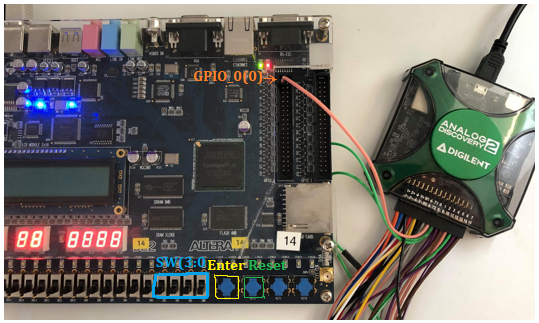


Figur 24 - Funktionel simulering for UART\_transmitter\_tester

Efter tilfredsstillende simulering af transmitteren er der blevet foretaget sidste test for at sikre os at DE2-Boardet virker som det skal, denne test er for code\_lock\_uart.

### Test for code\_lock\_uart

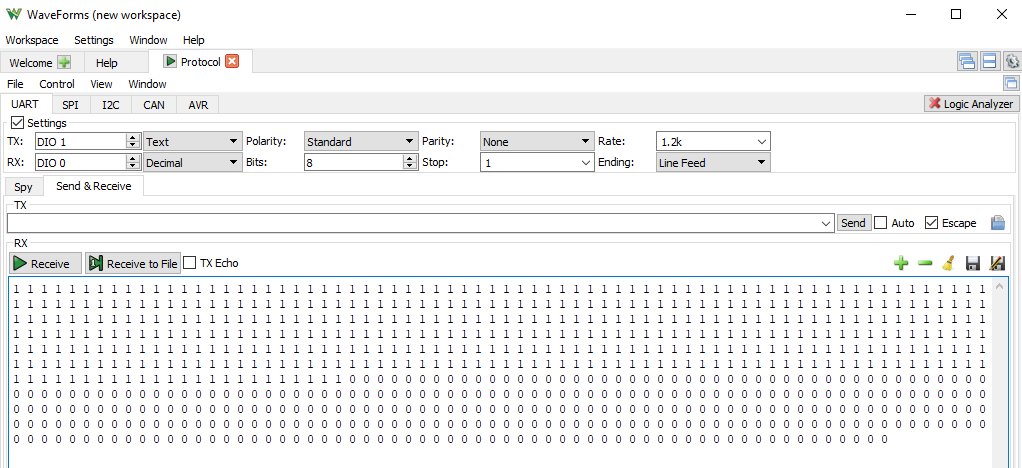
Nedenstående vises et billede af opstillingen af test for DE2-board.



Figur - Opstilling af DE2-board og Analog Discovery

Opstillingen for test af kodelåsen på figur 25 består af DE2-boardet og Analog Discovery 2. Via Analog Discovery kan man benytte protocol-funktionen i Waveforms, her kan man under UART, se hvilke signaler der bliver transmitteret. På opstillingen er de forskellige keys og switches markeret med farver. Koden er skrevet således, at programmet er i en locked-state, og bliver derfor ved med at sende et højt signal ud. Under testen indtastes kodesekvensen på de fire tildelte switches, efterfulgt af at man trykker på enter-knappen. Hvis koden er korrekt, ændres tilstanden fra locked-state til en unlocked-state og sender hermed et lavt signal ud. Hvis koden til gengæld er forkert, vil programmet blive ved med at sende et højt signal ud.

Figur 26 er et screendump af WaveForms, der viser hvordan outputtet ser ud, når den korrekte kodesekvens indtastes. Som man kan se, stater den med at sende et højt signal ud, dvs. at programmet er i locked-state. Der vil altså fortsat blive sendt ’1’-taller ud så længe den rigtige kodesekvens ikke er blevet tastet på DE2-Boardet. Så snart den korrekte kode bliver indtastet på DE2-boardet, vil programmet ændres til unlocked state og dermed sende et lavt signal (0´ere) ud gennem GPIO\_0 pinen. Waveforms viser det ønskede output og der kan derfor konkluderes, at kodelåsen virker.



Figur - Resultater i Waveforms af signal

Ideelt set skal DE2-boardet ikke forbindes til Analog Discovery. Outputtet der kommer ud af GPIO\_0 skal ud til arduinoen, så arduinoen kan give brugeren adgang til at ændre indstillingerne omtalt i Use Case 3. Men da vi grundet omstændighederne, ikke har haft mulighed for at benytte skolens laboratorie, eller kunne samle os som gruppe og opstille kredsløbet. Har vi forsøgt os at teste funktionaliteten vha. Analog Discovery.