Dokumentation SW

# SW1:

## 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, fra signaler der modtages fra SA. Efter arrayet er blevet fyldt, vil arduinoen indlæse værdierne, hvilket vil blive videresendt 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, en der initialisere PWM, en der kan konverterer binære tal til decimal tal, og en der, ud fra decimaltal værdierne, kan bestemme start og sluk tidspunkt, samt lysstyrken for lampen. 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 3 binære værdier (tænd, sluk, intensitet), mens switch kun skal omsætte 2 (tænd, sluk).

## System arkitektur - 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 11 – 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 12 - Sekvensdiagram for Use Case 3

Ændr wait():void til pressReset

På Figur 15 vises et opdateret klassediagram med funktioner og attributter. Her ses hvordan funktionerne i de forskellige klasser høres sammen:



Figur 13 - Klassediagram med members for Use Case 3

## 9.1.4 Test af arduino programmer

Ved brug af simuleringer, kan vi eftervise om vores forventninger er opfyldt. Dette gøres ved at simulere de givende scenarier.

Ved test af ændring af modeTwo, altså modeThree scenatiet, vil der udskrives standart værdier og dernæst ændrede værdier for modeTwo dimmer arrayet. Vi ser altså på, om modeThree har ændret på modeTwo.

Yderligere ved test af ændring af modeTwo, er der blevet testet om der bliver skrevet korrekt kode. Selve koden er her modderet en del. Dette er grundet mangel på tilgang til DE2-board. Der er altså blevet skrevet om på koden, så værdien kommer via PuTTy konsollen. Koden acceptere kun værdien ‘5’. Ved alle andre indtastninger, vil systemet give fejl og bede om koden igen.

Ved test af transmitering af bit værdier (HIGH og LOW) testes det om interrupt ISR(INT2\_vect) kan aktiveres. Dette gøres da ISR(INT2\_vect) kan aktiveres direkte på arduino sheildet. Der vises en success, da interruptet gør som forventet.

Der henvises til DOCUMENTATION for yderligere information.

## SW-Design Modulbeskrivelse:

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

TILFØJ codeLock() MENU

|  |  |
| --- | --- |
| 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:

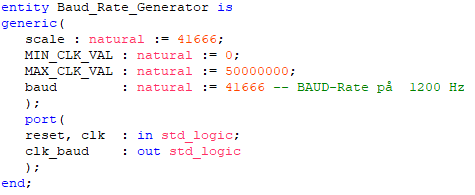
|  |  |
| --- | --- |
| Klasse: INT4 | |
| Funktion: | **Indeholder/Benytter:** |
| ISR(INT4\_vect) |  |
| Interruptet står for at indstile CTC til det 120kHz burst vi ønsker. Registret OCR3A gives værdien 65 for at starte outputtet på 120kHz. EICRB sættes til low-input, for at forberede regsitret til 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. | |

## Baud\_Rate\_Generator

Baud.vhd

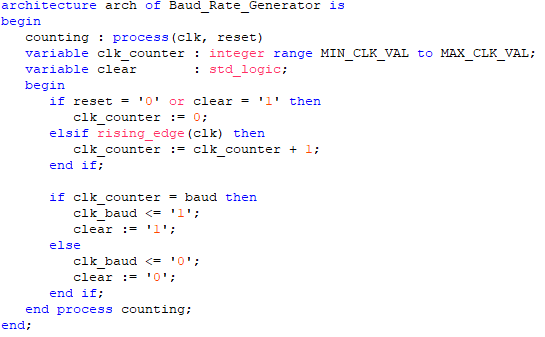
* Baud.vhd består af entiteten og arkitekturen for Baud\_Rate\_Generator. Formålet med denne implementering er at man selv kan indstille en passende hastighed, for hvor hurtigt data skal overføres fra vores transmitter.

Baud\_Rate\_Generatorens implementering starter med nedenstående kodeafsnit, hvor entiten er opbygget.



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

I næste kodeafsnit er architecturen for Baud\_Rate\_Generatoren implementeret

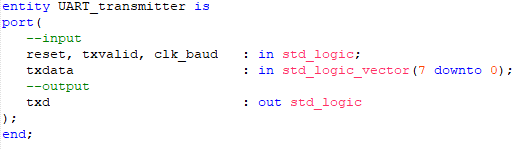


Her anvendes der to yderligere variable, clk\_counter og clear. Disse benyttes i forb. med if-else og elsif-sætninger der sammen har til formål at sætte værdien af clk\_baud, som er output og bliver benyttet af UART\_transmitteren. Det kan ses at hvis der efter gentagne rising\_edge(clk), så bliver clk\_counter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

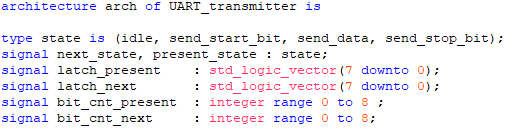
## UART\_transmitter

UART\_transmitter.vhd

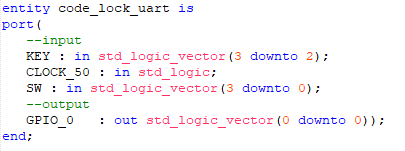
UART\_Transmitteren entitet ses i følgende kodeafsnit

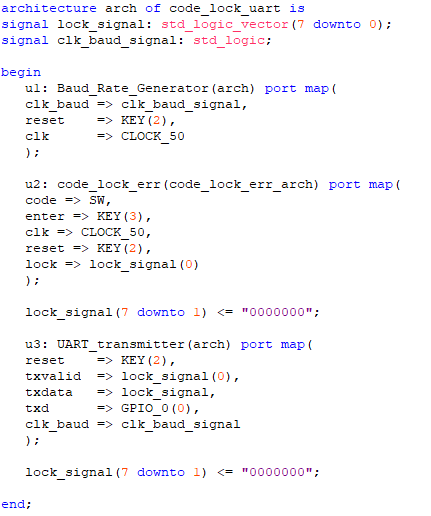


Her kan vi se de porte, inputs og outputs, der anvendes i arkitekturen nedenfor

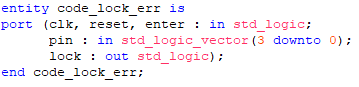


## Code\_lock\_uart:



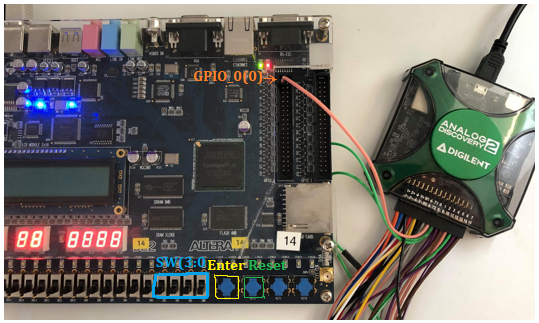


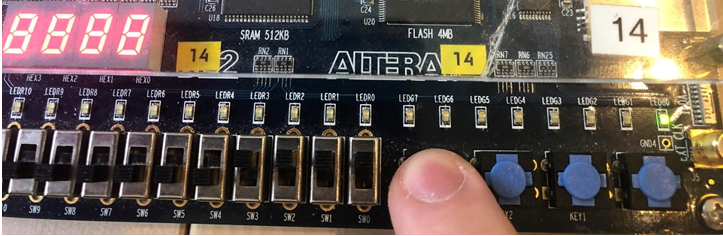
## Code\_lock\_err:



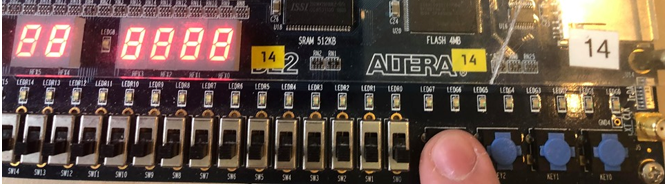
## Opstilling af test for kodelås

Opstillingen for at teste kodelåsen består af DE2-boardet og Analog Discovery 2. Koden indtastes på switchene, der er markeret med blåt i billedet. Herefter trykkes der på enter-knappen. Hvis koden er korrekt sendes 0´ere ud, og hvis koden er ukorrekt, fortsættes der med at sende 1´ere ud. For at teste om dette virker anvendes Analog Discovery og Waveforms. Analog Discovery forbindes til GPIO\_0 pinen og Waveform åbnes.





Figur 20 Test på De2-Board her indtastes code1 ”1111”, det ses at LEDG[0] lyser så lock = ’1’

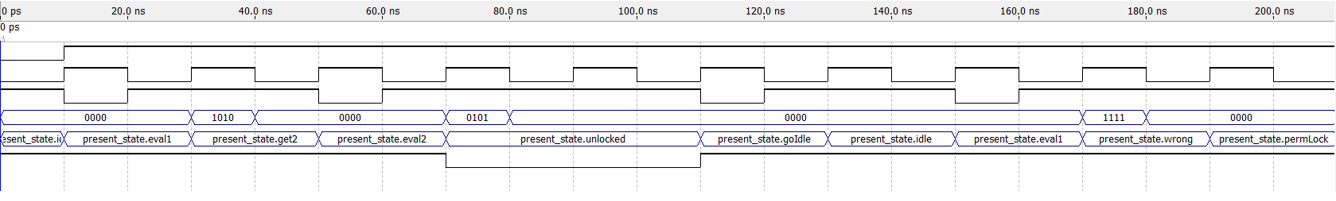


Figur 21 Test på DE2-Board her indtastes code2 ”0000”, hvor LEDG[0] stopper med at lyse så unlock er initieret



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.

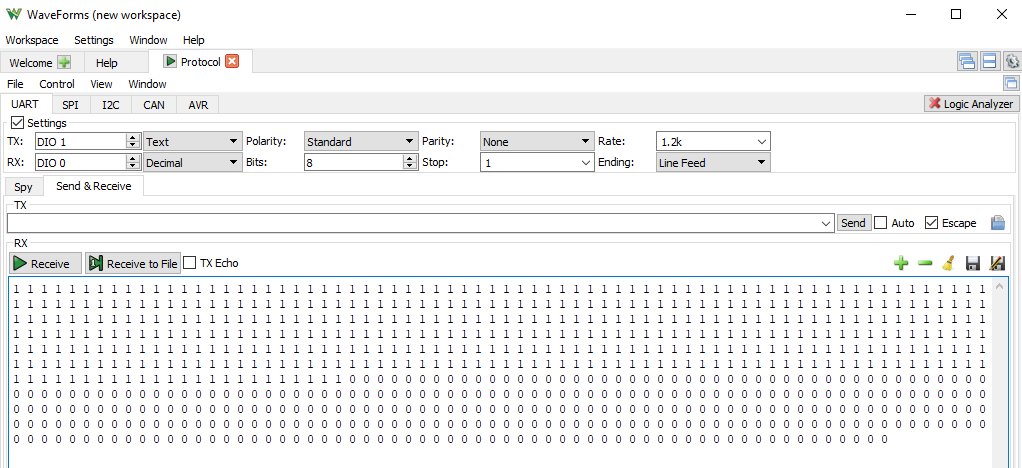
### Funktionel Simulering:



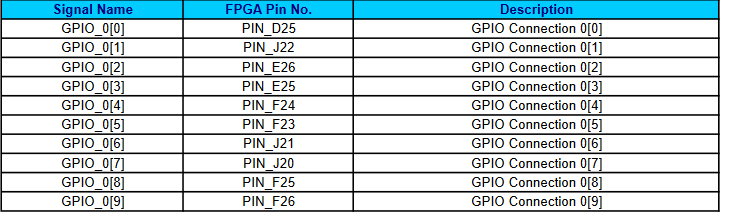
Figur - Funktionel simulering hvor wrong code simuleres, heraf ser vi staten permLock er med

### Resultat i WaveForms

Nedenstående billede viser hvordan kodelåsen fungerer. Indtil der bliver indtastet en korrekt kode, vil programmet forblive i en locked state. Hvilket vil sige, at programmet bliver ved med at sende 1´ere ud. Så snart den korrekte kode bliver indtastet på DE2-boardets switches, vil programmet ændres til en unlocked state og dermed sende 0´ere ud til arduinoen. Waveform viser det ønskede output og der kan derfor konkluderes, at kodelåsen virker.



Reelt 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. Men da gruppen ikke har mulighed for at samles som gruppe og bygge kredsløbet op pga. corona, er der kun blevet testet gennem brug af Analog Discovery.



<http://www.terasic.com.tw/attachment/archive/30/DE2_Pin_Table.pdf>

