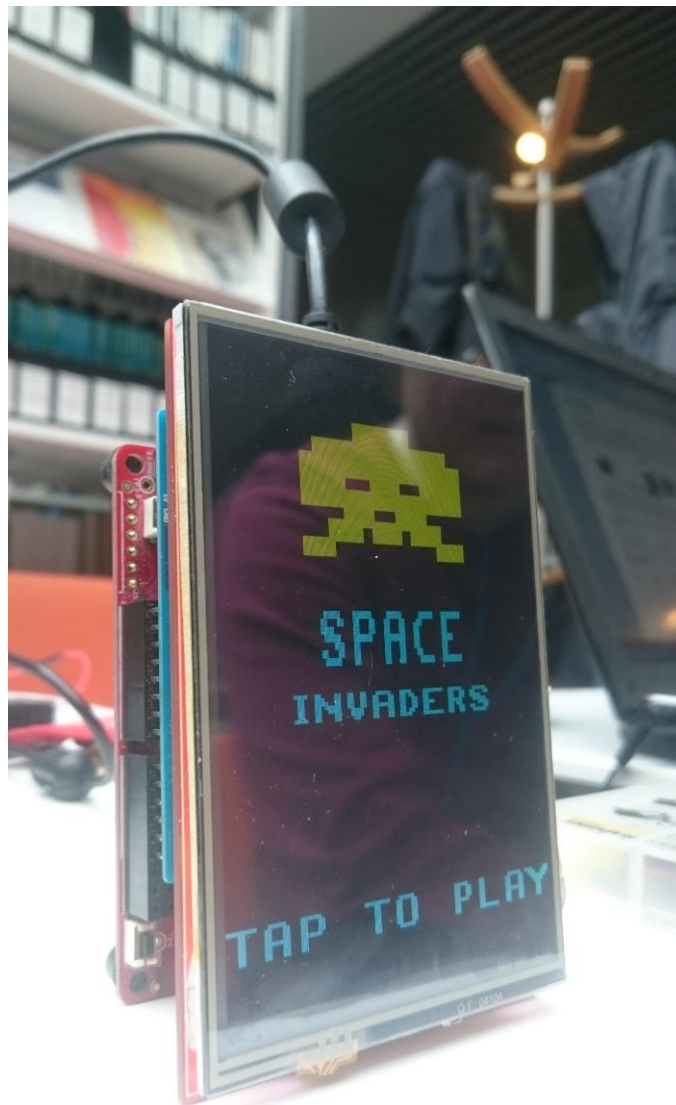


# Space Invaders

## Gruppe 11

Laimonas Ignas Bendikas, 201408723

Jonathan Schougaard, 201407553



## Indholdsfortegnelse

Indledning.....	2
Overordnet projekt beskrivelse.....	3
System beskrivelse .....	5
Blokdiagrammer for Space Invaders .....	5
Udvidet beskrivelse af topdesignet samt drivers .....	6
Emuleret EEPROM .....	6
Tick Kontroller .....	6
TFT_Control - Register kontrol .....	8
ADC_TOUCH .....	9
Touch driver.....	10
TFTLCD driver.....	11
Aktivitetsdiagram – Space Invaders .....	12
Klassediagram – GameEngine .....	13
Sekvensdiagram – GameEngine .....	14
Tilegnelse af viden og problemer undervejs .....	15
Matlab Script – Fra billede til header fil .....	15
In-circuit debugging.....	16
Test resultater .....	16
Konklusion .....	18
Referencer .....	18

**Antal normal sider: 13,3.**

## Indledning

I dette projekt er der tænkt at udarbejde et spil. Specifikt er det Space Invaders, som vil laves. Spillet vil ikke være komplet det samme, som man kender det – Dog en nedkogt og ændret udgave. Til hardware har vi bl.a. besluttet at benytte os af en TFT skærm, touch interfacet på skærmen, EEPROM til at high scores. Projektet vil blive skrevet i objekt orienteret C på en PSoC4 (ARM cortex-m0) og vi bestræber os på selv at udarbejde al kode og ikke låne fra nogen steder. Dvs. også vores GameEngine og de tilhørende elementer udarbejder vi selv.

Man vil kunne være i stand til at styre 'helten' i spillet med touch ved f.eks. bruge sin finger til at trykke på skærmen for at skyde, og ellers vil figuren følge fingerens position på skærmen. Koden vil blive optimeret for hastighed samt vi vil skyde på at have en tick rate på omkring 60 (opdateringer i sekundet).

## Overordnet projekt beskrivelse

I dette afsnit vil der kort blive præsenteret for de forskellige hardware dele som indgår i projektet.

Der er altid nogle fordele og ulemper ved valg af forskellige processorer og platforme. Det umildbare valg stod mellem Mega32 med STK-500 kittet, en Arduino eller en **PSoC 4**. Efter nogle overvejelser om hastighed og bl.a. udvikling i forskellige miljøer, så faldt valget på PSoCen. Den er blevet brugt i tidligere semestre og derfor kendtes dens styrker og hvordan man skulle benytte sig af den. Kort sagt, så befinder der sig en ARM M0 processor på PSoCen med en hastighed af **48MHz**, hvilket er 3 gange hurtigere end en mega32. Dertil foreligger **32kB flash** og **4kB SRAM**. Det er ikke meget at gøre med især ikke, hvis man vil have billeder eller mange elementer kørende på samme tid. Det mentes, at PSoCen var det ideelle valg, eftersom dens arkitektur gør, at man selv kan skræddersy sin løsning ved kun at bruge den nødvendige hardware. Hertil er det også en del nemmere at debugge og optimere på platformen, som er et stort plus.

Nedenfor på Figure 1 og Figure 2 ses et kort overblik over topdesignet på PSoCen. Det væsentligste at bemærke er, at der bl.a. er benyttet kontrol registre, ADC, EEPROM og en timer.

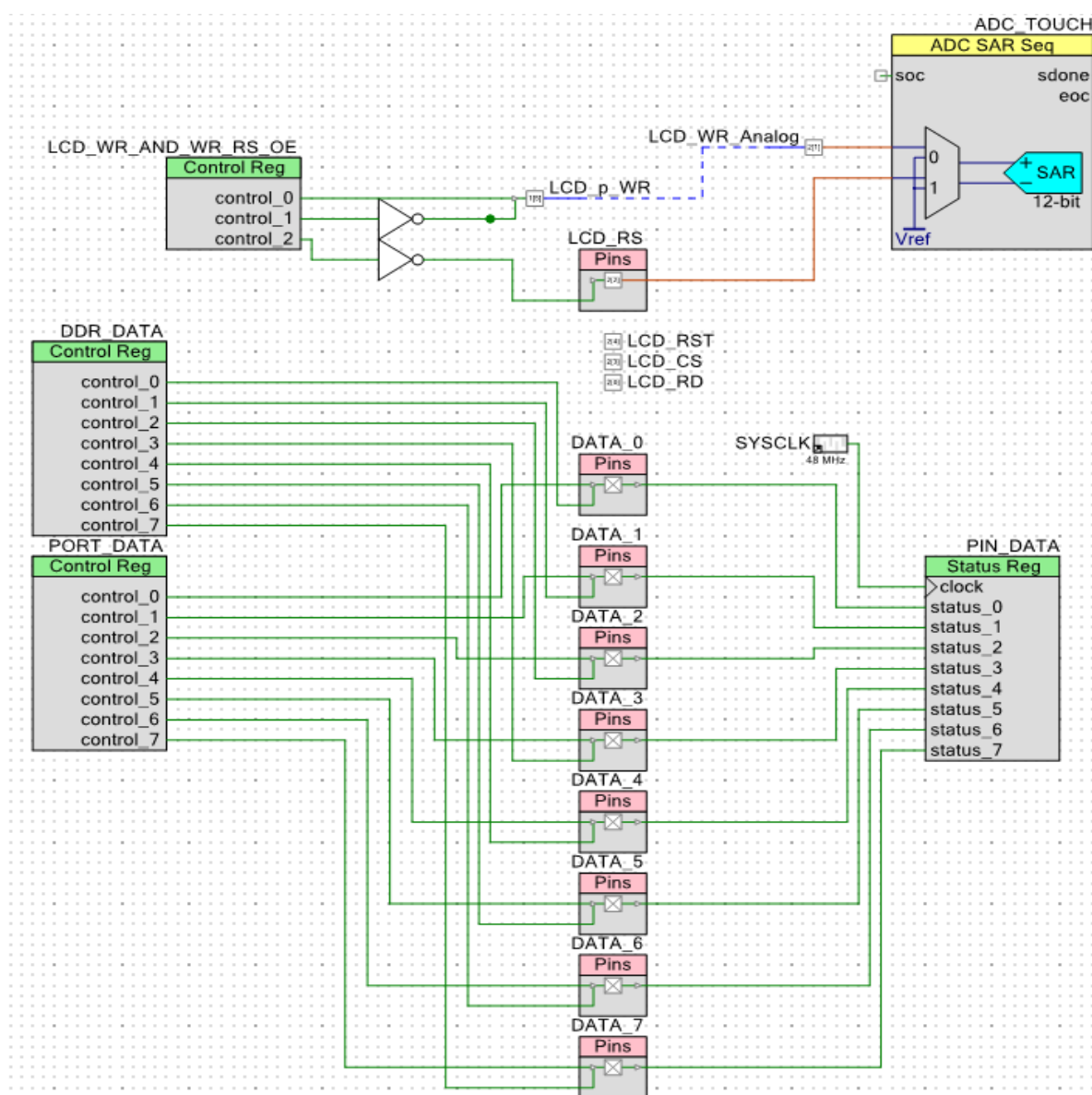


Figure 1 - Overblik over systemet og dets sammenhæng. På billedet ses nogle kontrol registre, en ADC og diverse pins.

Eftersom TFT skærmen passede som et arduino-shield ned i PSoCen, så skulle de fleste pins mappes internt i PSoCen. Specielt for WR og RS pins skulle de også forbindes til en ADC for at kunne læse Touch data ind. Derfor ses der to WR pins (LCD\_p\_WR og LCD\_WR\_Analog), og den blå forbindelse symboliserer, at de fysisk er den samme pin (forbindelsen er blevet loddet sammen). For så at kunne styre om man ville læse touch data eller sende WR kommandoer til skærmen er de blevet styret af endnu et kontrol register.

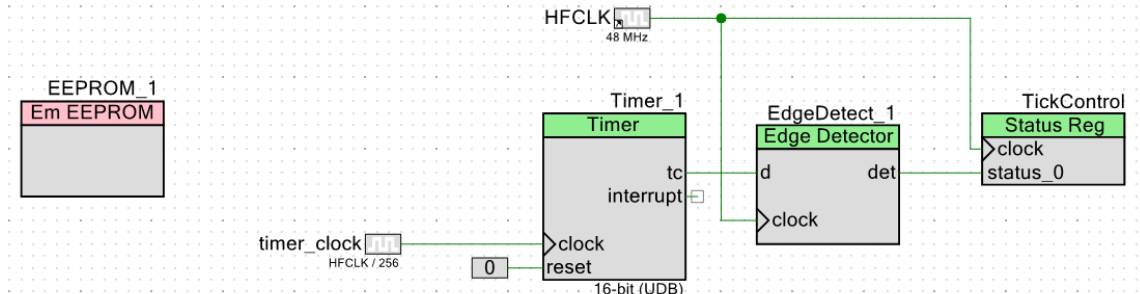


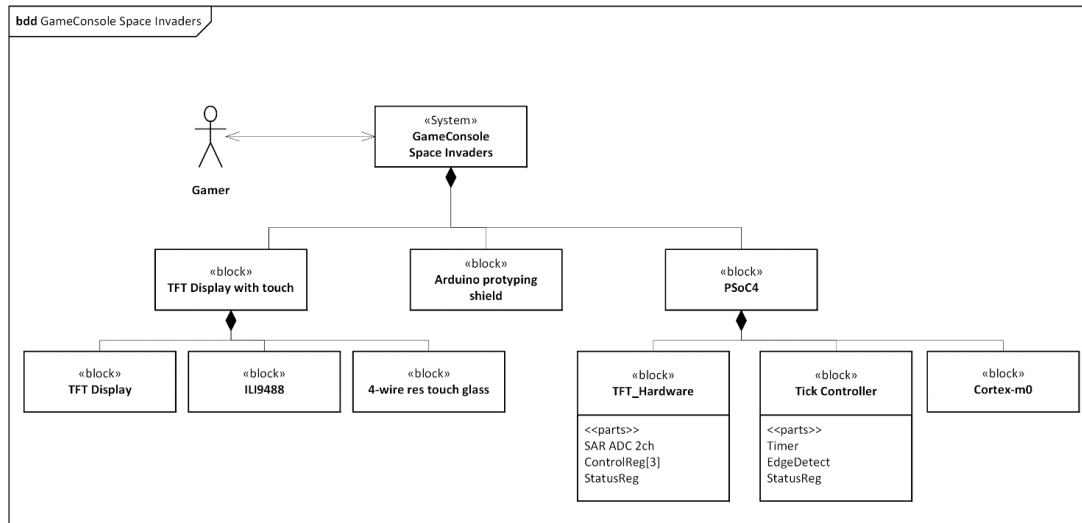
Figure 2 - Overblik over EEPROM og Tick Control – Her ses bl.a. hvad TickControl komponentet består af.

På Figure 2 ses en emuleret EEPROM blok, hvilket vil sige, at den bruger noget af flash hukommelsen sådan så det fungerer som EEPROM. Dette benytter vi senere til at gemme highscoren i spillet. Dertil bruges timeren som en opdateringskontrol for hvor mange ticks per second, som vi gerne vil have.

De blokke der menes essentielle eller især relevante for vores projekt vil blive omtalt individuelt og grundigere i et det næste afsnit.

## System beskrivelse

### Blokdiagrammer for Space Invaders



Figur 1 – BDD for GameConsole – Her ses et overblik over systemets blokke. Overordnet set så består systemet af tre hoveddele, et TFT Display der er beregnet til Arduino uno; et Arduino prototyping shield og en PSoC4, som underlæggende styrer al logik.

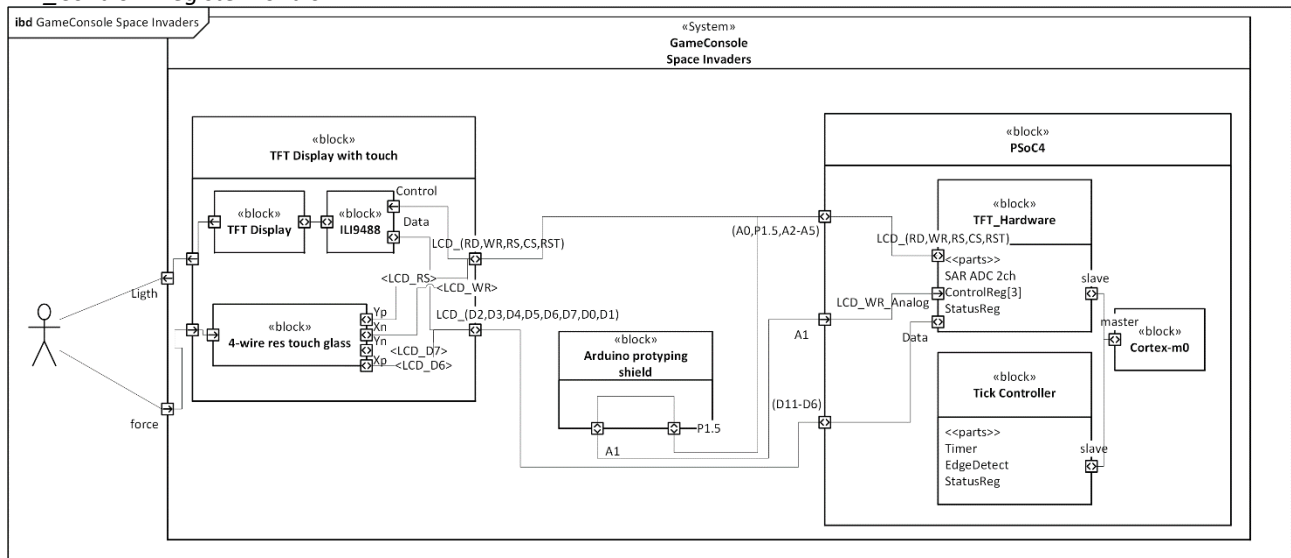
**TFT Display with touch:** Er systemets interface, det bliver brugt af brugeren til at interagere med og systemet bruger displayet til at vise spillet til brugeren. Displayet består af et TFT display, der bliver styret af en ILI9488 skærm kontroller, med indbygget grafisk hukommelse. Touch delen består af et 4-wire resistiv touch glass system, som sidder ovenpå skærmen. Den ændrer sine resistive værdier ud fra hvor henne på skærmen brugeren trykker, og hvor hårdt der bliver trykket.

**Arduino prototyping shield:** Er et ekstra komponent, der var nødvendig at tilføje på grund af pin begrænsning i PSoC4 arkitekturen. Det bruges til at tilkoble to pins sammen på PSoC4 (ekstern rute); En udvidet forklaring kommer i *TFT\_Control - Register kontrol* afsnittet.

**PSoC4:** Denne blok består af en lang række komponenter, eller rettere og sagt en masse digitale blokke og analoge blokke og en enkel Cortex-m0. PSoC4 giver en mulighed for selv at vælge hardware opbygningen.

**TFT\_Hardware:** Der er i dette projekt bygget noget internt hardware til at interagere med ILI9488 enheden og 4-wire res touch glass. Til dette er der brugt en SAR ADC, tre kontrol registre og et status register.

**Tick Controller:** Er en hardware blok der bruges til at styre noget af softwaren på Cortex-m0. Dette vil blive uddybet i *TFT\_Control - Register kontrol*.



Figur 2 - IBD GameConsole Space Invaders – Dette giver et overblik over hvordan blokkene i systemet er koblet sammen.

## Udvidet beskrivelse af topdesignet samt drivers

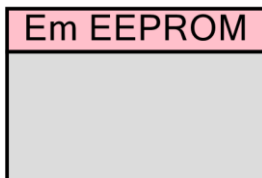
### Emuleret EEPROM

Der var brug for noget non-volatile hukommelse til at gemme ens High-score i spillet. Dog stod vi over for den problematik at PSoC4 ikke har indbygget EEPROM, som ellers ville have været det optimale valg. For at løse dette problem har Cypress lavet en emulator, der giver mulighed for tilskrivelse i flash området, hvor programkoden normalt ligger. Dette område er som standard kun et "read only" område, og PSoC4 skal være i en rigtig tilstand for at kunne tilskrive flash området.

Eksempel: udsnit fra "GameEngine.c"

```
static const uint8 CYCODE HighScore[4] = {0}; // Fortæl at dette skal gemmes på
flash ved at bruge postfix CYCODE.
...
uint8 newHighScore[4];
uint32 HighScore_ = (*(volatile uint8 *)&HighScore[0]); // Aflæsning fra flash
området.
...
status = EEPROM_1_Write(newHighScore, HighScore, 4); // Hjælpe funktion fra
Cypress der sørger for at tilskrivningen foregår korrekt.
```

For at få adgang til funktionen skal man først inkludere EM\_EEPROM i topdesignet, som er en ren software blok, der blot indeholder en brugbar funktion (de andre er tomme), nemlig "EEPROM\_1\_Write."

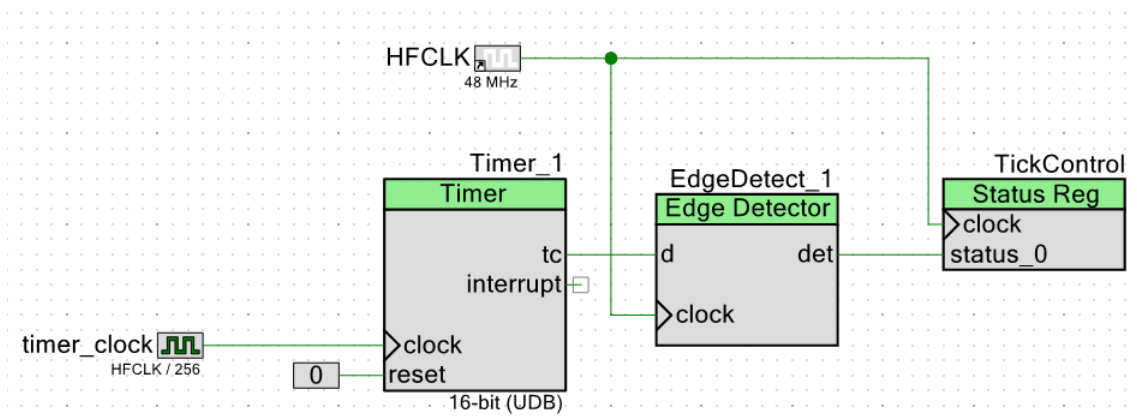


Figur 3 - Cypress EMULATED EEPROM - Tilføjer et API, som gør det muligt og sikkert at tilskrive et bestemt sted i FLASH området.

### Tick Kontroller

På baggrund af den måde, som spillet er opbygget på, så er det nødvendigt at spillet skal have en fixed tick rate. På denne måde vil man kunne sikre, at spillet ikke pludseligt eksekverer hurtigere eller langsommere fra det ene tick til det andet.

*"Nearly all video games (including Minecraft) are driven by one big program loop. Just as every gear in a clock is synchronized with the pendulum, every task involved in advancing a game simulation is synchronized with the game loop. Appropriately, one cycle of the game loop is called a **tick**"<sup>1</sup>*



Figur 4 – Hardware opsætning af TICK kontrolleren. Der er brugt 4 komponenter til at bygge TICK Kontrolleren.

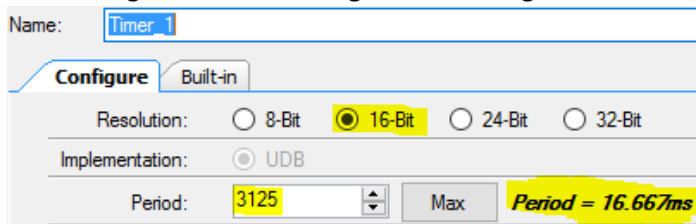
<sup>1</sup> <http://minecraft.gamepedia.com/Tick> (Det eneste sted som havde en fin beskrivelse af hvad et tick var.)

En Timer blok der er sat op til at generere et trig hver gang den rammer en ønsket værdi, hvorefter den vil resette sig selv til 0 og derefter tælle op igen. Til at trigge Timeren er der brugt en nedskaleret clock af HFCLK. På grund af at der er brugt en nedskaleret clock af HFCLK, så behøves der ikke nogle hardware komponenter til at synkronisere signalerne. Men eftersom vi blot er interesseret i trigget skal der bruges et EdgeDetect komponent, for ellers vil det ligne at der bliver genereret mange trigs. Som det sidste trin er der brugt et status register. Det er sat op til at være et sticky register (Dvs. det bliver holdt højt, hver gang der kommer et trig. Og den går ikke lav inden der sker en aflæsning).

### Opsætning

**Timer clock:** Grunden til at der er brugt en nedskaleret clock af HFCLK, skyldes at der ikke kunne implementeres en stor nok timer, på grund af mangel på UDB'er<sup>2</sup>.

**Timer 1:** Timeren er sat up til at generere et trig(tc<sup>3</sup> signalet går høj) hver gang dens værdi er lig med 0. I dette design er den sat til at generere et trig hvert 16.67ms, som svarer til 60 trig i sekundet.



**EdgeDetect 1:** Bruges til at detektere hver gang tc signalet går højt.

**Status Req (TickControl):** Et sticky register som der kan aflæses fra. Når det bliver sat højt vil det holde på dets værdi indtil der sker en aflæsning.

Det eneste nødvendige for at benytte sig af Tick Kontrolleren i software domænet, er at initialisere Timeren. Herefter vil den efter hvert tick forløb vente på at TickControl igen er lig 1.

Udklip fra "SpaceInvader.c" - Implementeringen af TICK Controller på software siden.

```
while(1)
{
    ... //game code
    while(TickControl_Read() == 0);
}
```

Denne løsning har nogle ekstra fordele. F.eks. hvis der er et tick der tager for lang tid, (f.eks. hvis mange skud skal allokeres) vil den komme ud af while løkken med det samme, og få mulighed for at indhente næste tick. Selvfølgelig kan der sket det, at tick hastigheden er for høj. Dette vil medføre, at spillet ikke længere kan følge med, og på den måde ikke være i stand til at opfange alle Trigs fra Tick Kontrolleren. Dette er derfor noget der skal indstilles lidt på, så man sikrer sig at få den helt rigtige værdi. (Dette design kører med 60 ticks i sekundet).

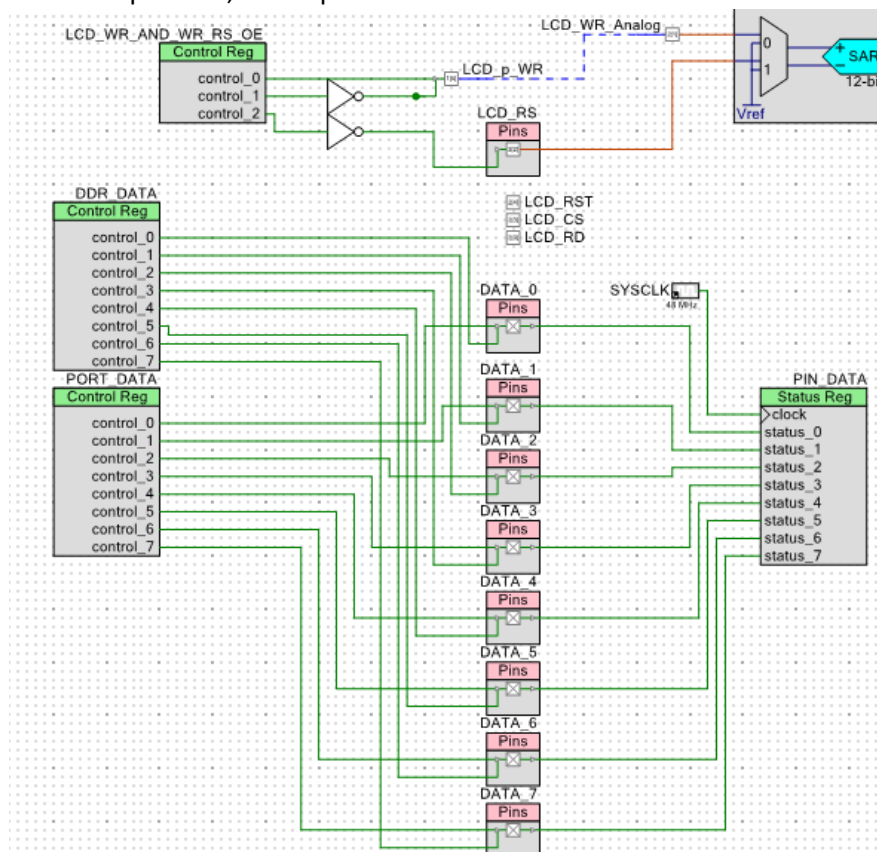
<sup>2</sup> Universal Digital Block : <http://www.cypress.com/video-library/PSoC/intro-udb-components-psoc-5lp/386161>

<sup>3</sup> Terminal Count: <http://www.cypress.com/documentation/component-datasheets/timer>



## TFT\_Control - Register kontrol

TFT Skærmen der blev brugt har pin layout til en Arduino uno, som også passer til den enhed der bliver brugt i dette projekt: PSoc 4 Pioneer Kit<sup>4</sup>. Dog med den lille ulempe, at port layoutet ikke var helt ens. Og hvis der ikke blev gjort noget ved det, så ville det resultere i at koste mange software operationer blot at skifte værdi på DATA pins. For at løse dette problem, blev der brugt et par kontrol registre og et status register, til at lave en løsning der ikke ville koste for mange clock perioder. Man kan godt sige at portene er blevet omplaceret, så det passer med Arduino uno.



Figur 5 – Til DATA bene er der brugt to Control register "DDR\_DATA" og "PORT\_DATA", og et status register PIN\_DATA. DDR bruges til at aktivere output, PORT bruges til at tilskrive værdier til benene, hvor PIN kan aflæse fra benene. Der er dertil opbygget et specielt kontrol register "LCD\_WR\_AND\_WR\_RS\_OE".

### "LCD WR AND WR RS OE":

control\_0 bruges til at bestemme output værdien for LCD\_WR.

control\_1 bruges til at aktivere eller deaktivere outputtet for LCD\_WR, note er aktiv lav.

control\_2 bruges til at aktivere outputtet for LCD\_RS, note aktiv lav.

Grunden til at OE er aktiv lav skyldes at TFT Displayet allerede var implementeret før dette kontrol register blev lavet. Og dette blev gjort for at sikre os, at der ikke skulle opstå uventede resultater efter implementeringen, da "LCD\_WR\_AND\_WR\_RS\_OE" også kunne bruges som et kontrol register for "LCD\_WR".

"LCD\_p\_WR" og "LCD\_WR\_Analog" er vores "LCD\_WR".

"LCD p WR": "p'et" er en reminder om, at den skal styres fra "LCD\_WR\_AND\_WR\_RS\_OE";

"LCD WR Analog": er en analog pin der er loddet til "LCD\_p\_WR" så det er muligt at måle spændingen (Nødvendigt for at kunne læse touch data fra skærmen – Se Touch driver).

<sup>4</sup> <http://www.cypress.com/documentation/development-kitsboards/cy8ckit-042-psoc-4-pioneer-kit>



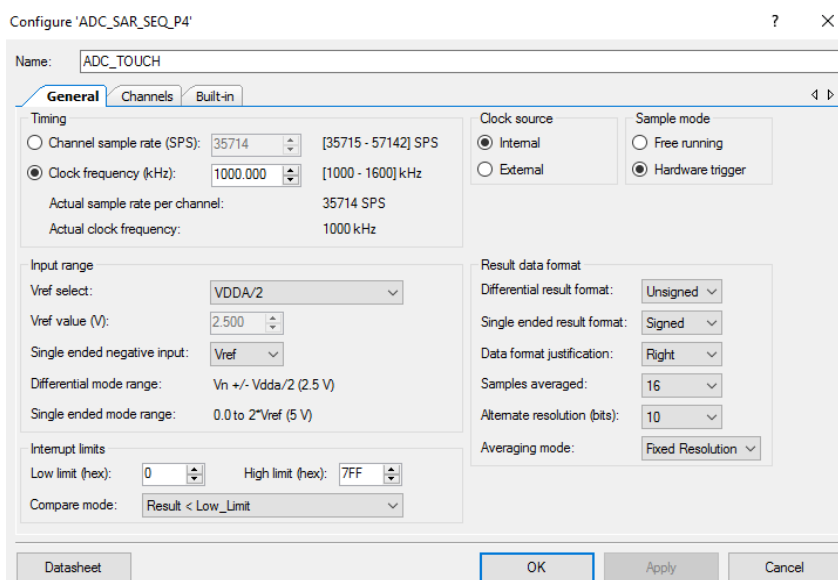
"LCD\_p\_WR" og "LCD\_WR\_Analog" kan virke som en underlig løsning, især når man kigger på "LCD\_RS" hvor alle forbindelser er samlet i et ben/pin. Grunden til dette skyldes PSoC4 arkitekturen. Man kan simpelthen ikke have en hardware et digital output og OE(output enable) samtidig med et analogt input ben.

Dette kunne løses ved at undlade at have hardware digital outputtet, dog så ville det være på bekostning af performance. Med denne løsning ville det ske, at det ville komme til at koste mere end 4 ekstra clockcycles på grund af PSoCs arkitektur (Kode eksempel kan findes i TFTLCD driver). Man skulle tro, at det ikke er meget, men dette ben skal toggles over 4 gange for hver pixel tilskrivning. Hvis man så vil skrive mange pixels ad gangen så bliver koden i princippet mere end 4 gange langsommere til at tegne skærmen. Den endelige løsning, som er blevet omtalt her tager kun en enkel register tilskrivning for at lave en pixel tilskrivning. Dette er en væsentlig bedre og hurtigere løsning.

## ADC\_TOUCH

Der er blevet valgt at bruge en ADC, da der er nogle analoge pins der skal aflæses i forbindelse med når der bliver trykket på skærmen. Hertil har PSoC 4 en SAR ADC til rådighed. Her er der valgt at køre 'single ended mode', dvs. at vi får en unsigned værdi tilbage.

Opløsningen er 10 bit, man kunne vælge mere, men med en opløsning der svarer til 4,88mV, synes det mere end rigeligt til formålet. Dertil er der valgt at køre ADCen i hardware trigger mode. Dette forårsager, at selvom man har initieret ADCen, så vil den ikke sidde og sample på inputtene hele tiden og bruge



ressourcer på dette. Den har så en funktion fra ADCens API som gør, at man kan udføre et software trigger – Så dette udføres der, når der er behov for at aflæse fra skærmen og få touch data. Antallet af gange der tjekkes for tryk på skærmen i sekundet er defineret af TickControl, som er sat til at være 60 gange i sekundet. Når værdierne er indlæst omregnes de og skaleres ned til skærmens størrelse, så de endelige værdier ikke kommer til at overskride pixel opløsningen på de 320x480.

Figure 3 - Overblik over ADCens opsætning i PSoC Creator

De fordele eller ulemper der kunne være med ADCen, når man snakker tidskritiske systemer, kunne være om ADCen kommer til at bruge for mange ressourcer til at sample og konvertere værdierne den får ind. Hvis man f.eks. havde en lavere clockfrekvens, som f.eks. mega32 har, ville det ikke være så hensigtsmæssigt at benytte sig af en meget høj samplingsrate og overveje om man f.eks. kunne gå ned i kvaliteten af dataet og dertil gå op i performance. Med kvalitet menes der f.eks. opløsningen, antallet af samples der bliver midlet over. Men med PSoC arkitekturen har man her mulighed for kun at sample når man har brug for det, derfor bruger den ikke nær så mange ressourcer, hvis den kørte frit.

## Touch driver

Der var gjort nogle overvejelser om hvordan vi skulle styre 'helten' i spillet med f.eks. en xbox controller der skulle interfaces over noget usb sådan så kunne styre figuren. Men det ville være for besværligt, og måske ikke muligt at hoste fra PSoCen, eftersom der jo findes flere tusinder af controllers. Dog så endte vi med at blive enige om, at det nok var lettest og fedest at benytte sig af touch skærmen som hørte til TFT displayet. Som tidligere nævnt, så har det været meget besværligt at få touch funktionaliteten til at virke. Hvori den største årsag var, at der ingen dokumentation var at finde for touch opsætningen på vores skærm. Så der skulle testes og debugges meget for at få det til at virke i sidste ende. Vi startede med at finde frem til, hvordan der skulle interfaces til en touch skærm. Dernæst skulle der så findes frem til, hvordan lige præcis den skærm vi havde benyttede sig af de pins ved at debugge på dem.

Generelt set, så skal man bruge fire pins for at kunne aflæse fra en resistiv skærm. Minimum to af de fire pins skal være analoge for at kunne aflæse en given værdi. De andre kan være digitale for at enten forbindes til stel eller sættes til forsyningsspændingen. Når man så driver strømmen på en bestemt måde gennem en resistiv skærm, så vil man være i stand til at aflæse de modstande der forekommer.

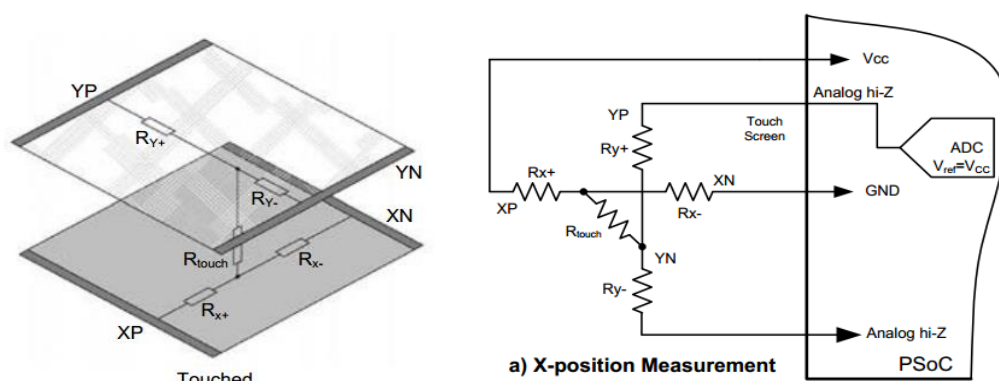


Figure 4 - Oversigt over en resistiv skærm samt hvordan man kan aflæse X positionen fra den

På Figure 4<sup>5</sup> ses det, hvordan man kan kategorisere modstandene som forekommer i en resistiv touchskærm. Og det er dette princip som vi har benyttet os af, når vi skulle foretag aflæsningen for X og Y koordinaterne. Efter mange debugging sessioner var vi i stand til at kunne aflæse X koordinaten og få en

værdi om der var blevet trykket på skærmen eller ikke. Disse værdier kunne vi så senere bruge til at hhv. styre figuren og skyde med.

Et udsnit af "touch.c" ses til venstre. Her udføres den opsætningen for at læse X-koordinaten som er vist på Figure 4. Når det er gjort, forekommer der et lille delay på 1ms, så man er sikker på, at alle pins er nået at blive indstillet. Derefter trigges ADCen til at læse data ind, også ventes der på at konversionen er færdig. Herefter læses dataet ud i en værdi. Alle registre sættes derefter til deres default værdier og den aflæste værdi returneres.

```
uint16 ReadTouchX()
{
    //DATA 7 hiZ, DATA 6 VCC, RS GND, WR hiZ
    DDR_DATA = 0b01000000; //input DATA_7, output DATA_6
    PORT_DATA = 0b01000000; //DATA_7=0, DATA_6=1

    LCD_RS_Write(0); //xn gnd
    WR_AND_WR_RS_OE = 0b011; //yn høj impedant

    CyDelay(1);

    ADC_TOUCH_StartConvert();
    ADC_TOUCH_IsEndConversion(ADC_TOUCH_WAIT_FOR_RESULT);
    uint16 yp_temp = ADC_TOUCH_CountsTo_mVolts(0u,
    ADC_TOUCH_GetResult16(0));

    DDR_DATA = 0;
    PORT_DATA = 0;
    WR_AND_WR_RS_OE = 0b001;
    LCD_RS_Write(1);
    return (yp_temp);
}
```

<sup>5</sup> <http://www.cypress.com/file/134996/download> d. 7/3-2017

## TFTLCD driver

Skærmen til vores projekt er en TFT skærm som er baseret på en ILI9488<sup>6</sup> controller. Det væsentligste at nævne i dette afsnit er hvordan vi har formået at optimere vores TFT\_LCD driver. Først blev der udregnet at en cycle blot var 20,8ns med 48MHz som clockfrekvensen på PSoCen. Hvilket er væsentligt lavere end en Mega32 lavest cycle tid på ca 62,5ns. Når der så blev taget hul på implementeringen begyndte vi at se, at vi kunne gøre det endnu hurtigere. Vi havde før udregnet at et delay på minimum 30ns ( $t_{wc}^7$ ) skulle bruge 2 clockcycles viste sig at bruge mere end 2 clockcycles i sig selv pga. PSoC arkitekturen. Hvis vi f.eks. ville skrive 1, til en pin, så skulle den bruge mere end 4 clockcycles internt på at udføre handlingen. Et udklip af PSoCs egen genererede API ses nedenfor hvor man enten kan sende 0 eller 1 som parameter til funktionen.

```
void LCD_RD_Write(uint8 value)
{
    uint8 drVal = (uint8)(LCD_RD_DR & (uint8)(~LCD_RD_MASK));
    drVal = (drVal | ((uint8)(value << LCD_RD_SHIFT) & LCD_RD_MASK));
    LCD_RD_DR = (uint32)drVal;
}
```

Dette gjorde at vores kode fungerede meget langsomt, hvis vi f.eks. ville sende meget data ad gangen hen til skærmen. Pin-opsætningen ville være ens i starten og slutningen af hver transmission, hvorpå det blot var WR man skulle sætte enten høj eller lav. Derfor inkluderede vi registre direkte ind i vores topdesign, som også kan ses på Figure 1. Hertil kunne vi så definere kontrol registret og sende 1 eller 0 direkte til WR på blot 1 clockcycle.

```
#define PORT_DATA PORT_DATA_Control
#define DDR_DATA DDR_DATA_Control
#define PIN_DATA PIN_DATA_Status
#define PORT_WR LCD_WR_AND_WR_RS_OE_Control
#define _NOP() asm("NOP")

void send_data(uint8 data)
{
    PORT_DATA = data;

    PORT_WR = 0; //data sent
    _NOP();

    PORT_WR = 1;
}

void TFT_write_data(uint8 cmd, uint8 *data,
size_t datasize)
{
    size_t i;
    LCD_CS_Write(0);
    execute_cmd(cmd);
    DDR_DATA = 0xFF;
    for (i = 0; i < datasize; i++)
    {
        send_data(data[i]);
    }
    DDR_DATA = 0x00;
    LCD_CS_Write(1);
}
```

Her til venstre ses et kode udsnit af vores

TFT\_LCD driver om hvordan vi sender data. Da alle pins undtagen CS, default er høje antages det også at de er det inden funktionskaldet, for at udføre så få unødvendige handlinger som muligt. Dertil skal CS trækkes lavt, for at vælge chippen for hver gang man ønsker at komme i kontakt med skærmen.

Ved TFT\_write\_data sendes en kommando (kan findes i TFT\_LCD\_cmd.h) man ønsker at eksekvere. Samt det data man ønsker at sende. Først trækker man CS lav, derefter sætter man alle DATA pins til outputs hvorefter at man kalder send\_data(...). Hvilket tager 4 cycles for PSoCen at udføre.

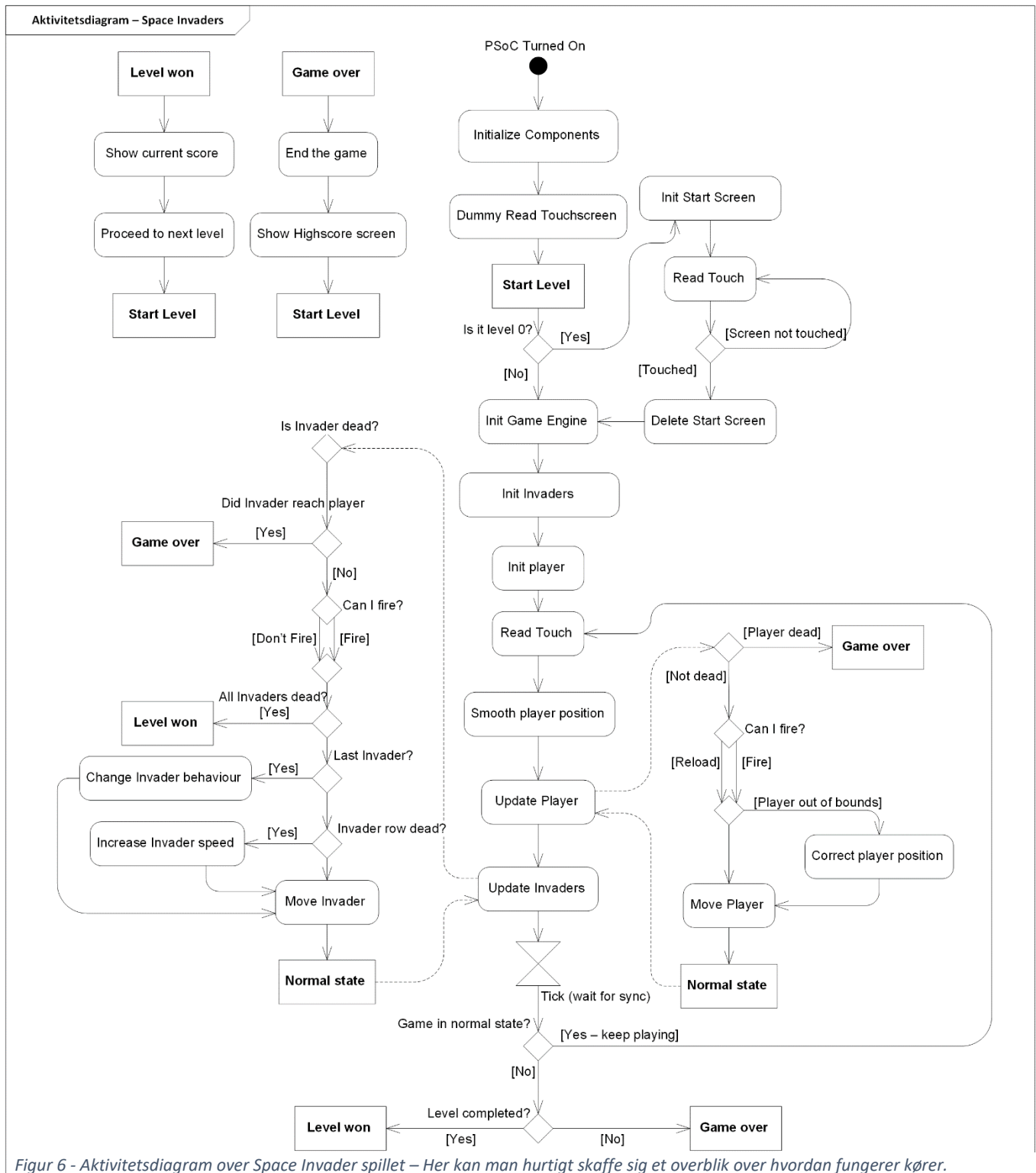
Den tilskriver data til PORT\_DATA registeret, så trækker den WR lav, for at vise skærmen at der bliver sendt data, så ventes der i 1 cycle (min 30ns). Der ventes kun i 1 cycle, fordi man lige før brugte 1 cycle for at sætte registrene, så derfra har man 2 cycles. Til sidst sættes WR til 1 for evt. at sende mere data.

<sup>6</sup> [https://www.lpcware.com/system/files/ILI9488\\_Preliminary\\_DS\\_V090.pdf](https://www.lpcware.com/system/files/ILI9488_Preliminary_DS_V090.pdf) d. 20/2-2017

<sup>7</sup> Se bilag (timing diagram for skærm kontrollere) - ILI9488\_AC.pdf

## Aktivitetsdiagram – Space Invaders

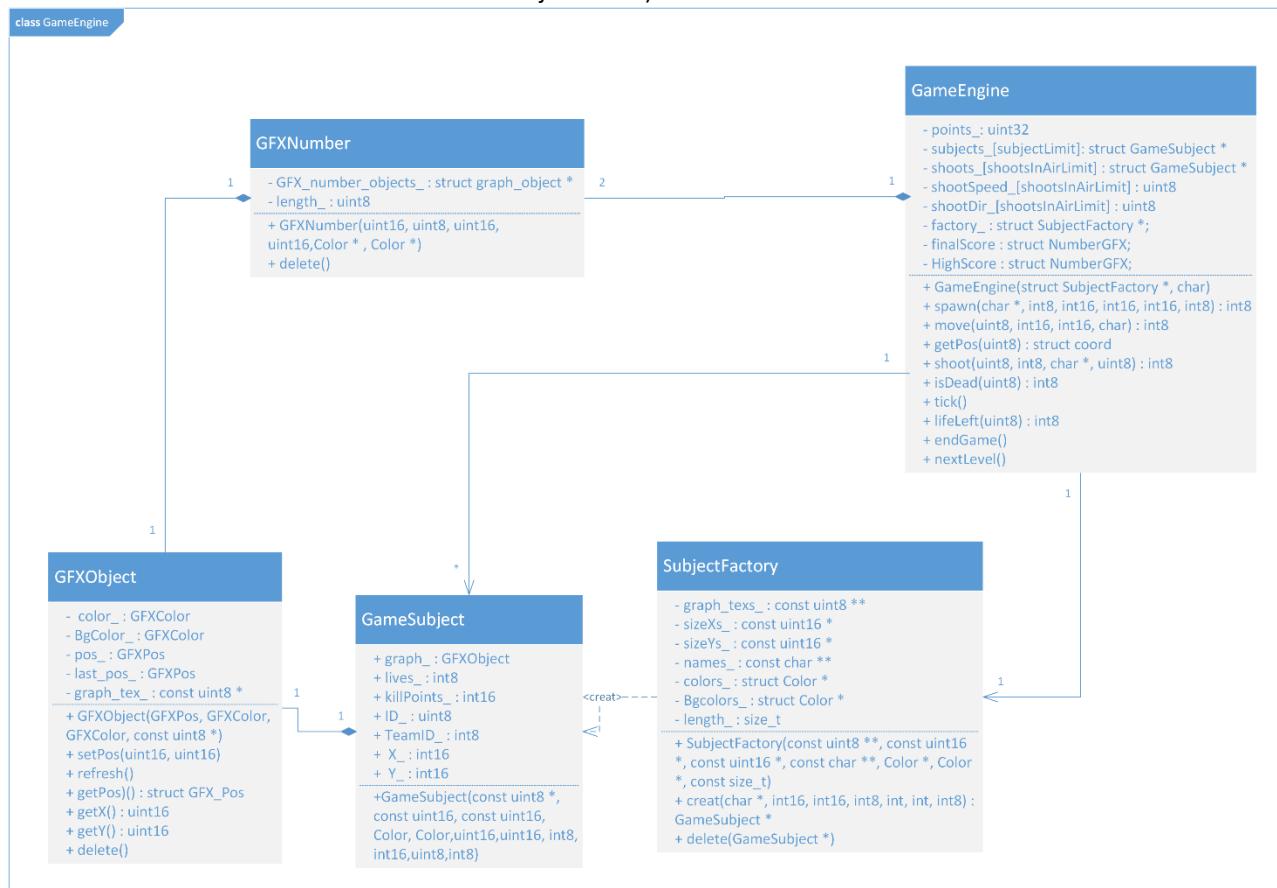
På Figur 6 - Aktivitetsdiagram over Space Invader spillet ses et oversigt over program flowet i selve spillet. Det er forsøgt at gøre det så simpelt og overskueligt ved at vise de forskellige instanser og steps i programmet samt uden at gøre det for meget eller for lidt programnært. Det bedes noteres, at når programmet er i enten "Update Player" eller "Update Invaders" steppet, så kan de returnere nogle værdier.



Disse værdier bliver returneret og brugt i det videre program forløb. Som det kan ses munder de ud i enten "Game Over" eller "Level Won."

## Klassediagram – GameEngine

GameEngine spiller en central rolle i produktet, og den står for håndteringen af subjekter og giver udvikleren mulighed for at bruge et par basale funktioner på dem. Det kunne f.eks. være at flytte dem, tjekke deres status (så som liv) eller affyre et skud. GameEngine gemmen løber ikke alle Subjekter hver gang en metode bliver kaldt. F.eks. hvis der skal flyttes på 16 Subjekter før det næste tick, vil de først opdateres, når metoden tick bliver kaldt i GameEngine. Denne metode opdaterer nye positioner, skud simulering og checker om et subjekt er blevet ramt af et farligt skud (et skud med et andet hold nummer end det nummer subjektet har).



Figur 7 - Klassediagram GameEngine – GameEngine er den klasse der står for håndtering af GameSubjects og den styrer dem ud fra de kommandoer der ønskes hvert tick.

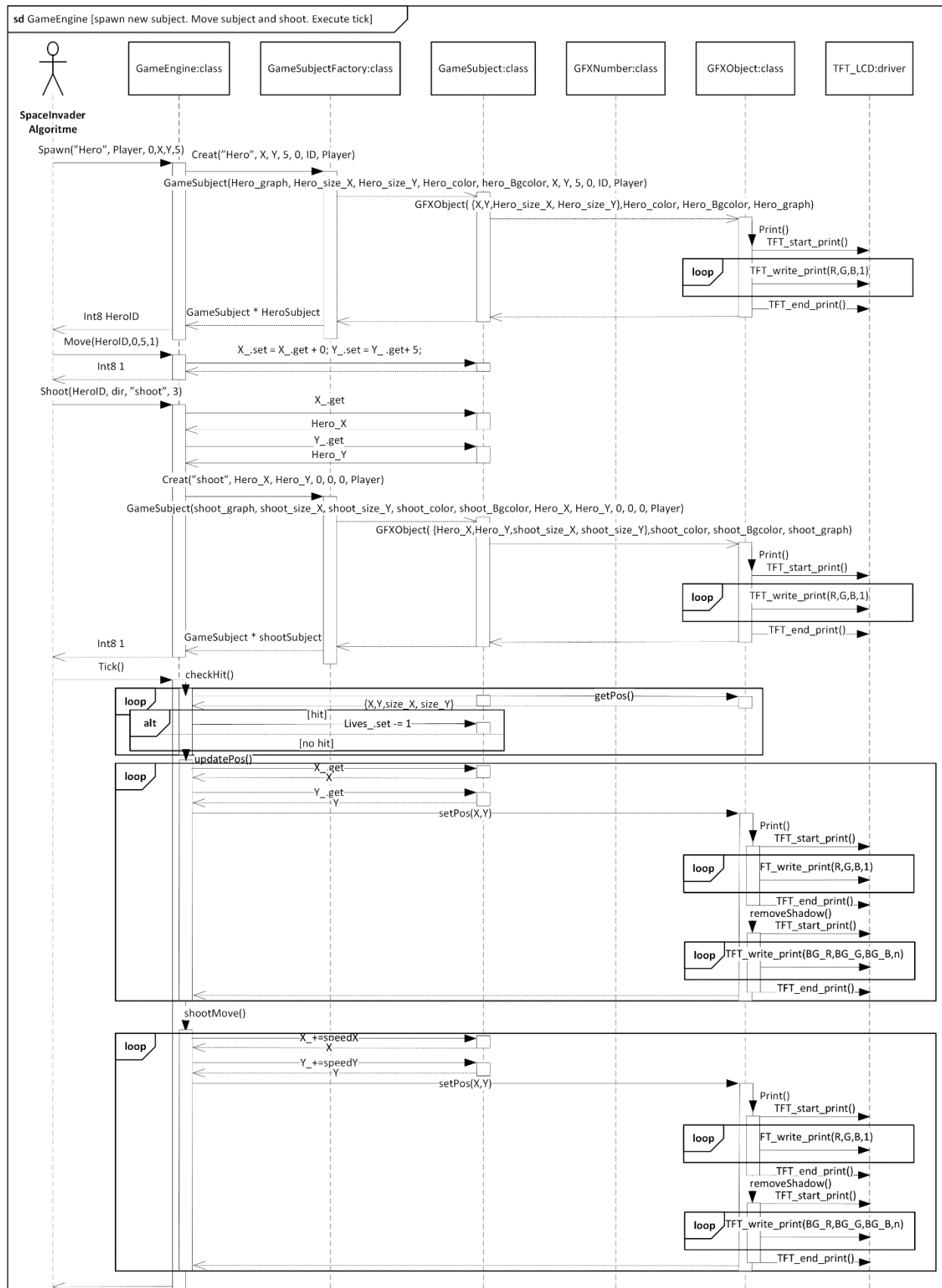
Måden man styre et GameSubject på, er via det unikke ID man får returneret når spawn metoden bliver brugt. Med IDet kan man flytte GameSubjectet, udføre handlinger (kun skyd er implementeret), tjekke hvor mange liv GameSubjectet har eller om det er dødt. Den måde GameEngine spawner et nyt GameSubject på, er ved at bruge GameSubjectFactory til at lokere, oprette og initialisere et GameSubject. Det gør den med metoden "creat", hvor den så returner en pointer til GameSubject'et, hvis den er løbet tør for hukommelse returner den en NULL pointer, som vil medføre at spawn kommandoen vil returnere -1.

GameSubject indeholder en række oplysninger;

- Koordinater (den position den vil have ved næste tick).
- ID (et unik ID for Subjectet).
- TeamID (Hold nummer, bruges til at undgå at spaceinvaders skyder hinanden).
- killPoints (hvor mange points man får, når et GameSubject bliver drabt).
- lives (Antallet af liv et GameSubject har tilbage).
- graph (Er det grafiske object der repræsenterer subjectet).

GFXObject er et grafiske object og står for håndtering af det der skal ske på skærmen via TFT driveren. Det kan f.eks. være hvordan den skal flytte et grafisk object.

## Sekvensdiagram – GameEngine



Figur 8- sd GameEngine – Dette diagram giver en oversigt over de mest basale kommandoer i GameEngine, spawn, move, shoot og tick.



Spawn: er den kommando der bruges til at oprette et nyt GameSubject. Det første skridt er at bruge metoden spawn til at fortælle GameEngine, at der ønskes et ny GameSubject af typen hero, herefter vil GameEngine bruge en Factory til at oprette det ønsket GameSubject. Når et GameSubject bliver oprettet vil der også blive oprette et tilhørende GFXObject, som er et grafisk element på skærmen - Derfor bliver der krævet en tilskrivning til skærm kontrollen. Hvis der ikke er opstået et hukommelse fejl vil factory'en returnere en pointer til det nye GameSubject, som GameEngine vil gemme i et array. Her vil dens index blive brugt som ID, hvilket også vil blive sendt tilbage til aktøren.

Den næste kommando er move som bruges til at flytte et GameSubject, det er muligt at vælge at flytte GameSubjectet ud fra origo, eller den relative position for GameSubjectet selv. Denne metode vil kun ændre nogle variabler i GameSubject, og vil ikke få GameSubjectet til at flytte sig. Dette sker først når et tick er sket.

Shoot kommandoen vil oprette et skud der kommer fra det GameSubject, som der henvises til. Denne metode fungerer på samme måde som spawn, udover at GameSubjectet bliver lagt i et andet array kun for at håndtere skud. Disse shoot GameSubject kan ikke styres af aktøren, men bliver automatisk styret af GameEngine ved hvert tick.

Den sidste metode der bliver kaldt fra aktøren er tick. Det første der sker i et tick er, at den kontrollerer om der er et GameSubject, som er ramt af fjendeligt skud. Hvis ja, så vil GameSubjectet miste et liv, og skuddet vil blive slettet.

Det næste skridt er så at opdatere alle GameSubject'ers positioner; her vil den gennemløbe alle GameSubjects og opdatere deres grafiske element så det passer med GameSubjectet.

Til sidst vil GameEngine opdatere skud positionerne.

## Tilegnelse af viden og problemer undervejs

Projektet indeholder en samling af en lang række erfaringer der er blevet gjort gennem de foregående og nuværende semestre. Derfor har det ikke været nødvendigt at lede efter mange ting. Men alligevel har der været nogle udfordringer. En af dem var, at få skærmen til at komme op at køre i starten – Dog viste det sig blot at være en skærm der var gået i stykker og at vores kode rent faktisk fungerede.

Dokumentationen til selve skærmen var god og fyldig - man var ikke i tvivl om hvorledes timingen skulle overholdes og de redskaber man havde med at gøre eller. Men når der så skulle arbejdes med touch interfacet fandtes der slet ikke nogen dokumentation. Her var det meget frustrerende at arbejde med den enhed, da det eneste man kunne gøre var at prøve sig frem med nogle princip fremgangsmetoder om hvordan en resistiv skærm fungerede.

For at løse de fleste udfordringer, så har en del af tiden været brugt på at søge på nettet og få inspiration eller se hvordan andre har udarbejdet deres drivere. Vi tænkte, at vi f.eks. kunne låne en touch driver, så vi kunne slippe for selv at implementere en og arbejde videre på nogle andre ting, dog endte vi med selv at implementere vores egen.

## Matlab Script – Fra billede til header fil

For at få billeder ned på PSoC4'en blev der lavet et Matlab script<sup>8</sup> til oversætte et billede om til uint8 array. For at få plads til så meget som muligt på den allerede begrænset hukommelse på PSoC4'en (32Kb) har vi valgt kun at bruge mono farvet billede (f.eks. hvid eller sort), på denne måde kunne vi have en pixel opløsning på 1bit/pixel.

---

<sup>8</sup> SpaceInvaders\SpaceInvaders.cydsn\graph\encode\_image\_too\_header.m



## In-circuit debugging

Det smarte ved det PSoC 4 Pioneering Kit som vi har er, at der ligger two ARM processorer på det. Nemlig PSoC4 chippen som vi benytter i projektet, men også en PSoC5LP ARM cortex-m3. Generelt set så er PSoC5LP chippen meget kraftigere end PSoC4, hvor vi har benyttet den i tidligere projekter – Der kunne den køre med en clock på ca. 96MHz, hvilket er dobbelt så hurtigt som den PSoC4 som vi benytter i projektet. Dog er den PSoC5LP chip der ligger på kittet en mindre udgave af den kraftige PSoC5, hvor den kun kan komme op på 67MHz<sup>9</sup>. Hvilket stadigvæk er meget, det var vores ønske at benytte PSoC5LP chippen, eftersom den også har f.eks. 64kB SRAM hvilket er 16 gange mere end PSoC4 – Bare for at sætte det i perspektiv. Men på det kit vi har, har PSoC5LP chippen kun 10 pins, hvilket ikke er nok og derfra ville det heller ikke passe med TFT skærmen shield.

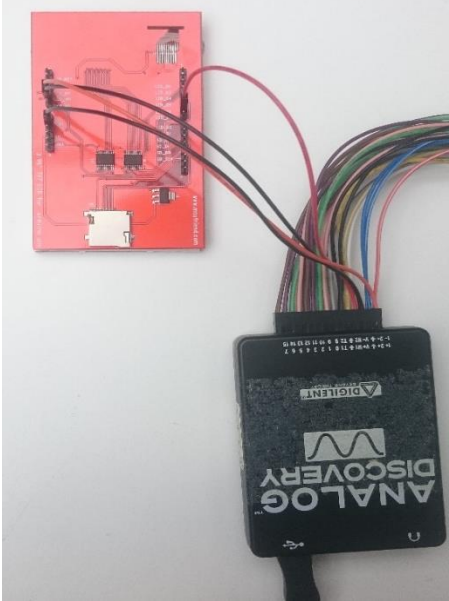
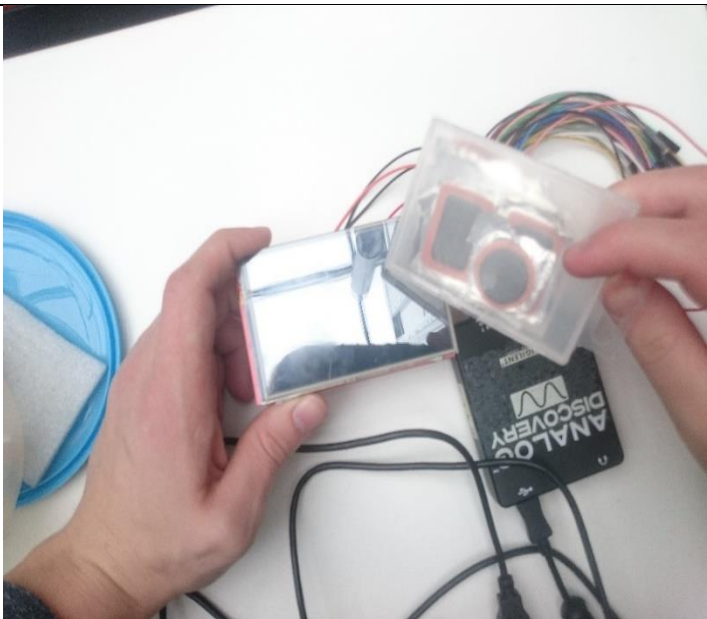
Dog er det smarte ved PSoC kittet at, når man vil debugge så vil PSoC5LP chippen benyttes som debugger på PSoC4 chippen. På den måde kan man få den samme performance, når man kører med debuggeren som uden. Det går en smule langsommere, men det kan ikke mærkes. Så på den måde benytter vi os også af en in-circuit debugger – bogstavelig talt.

## Test resultater

Her præsenteres nogle få test resultater fra TFT displayet samt touch.

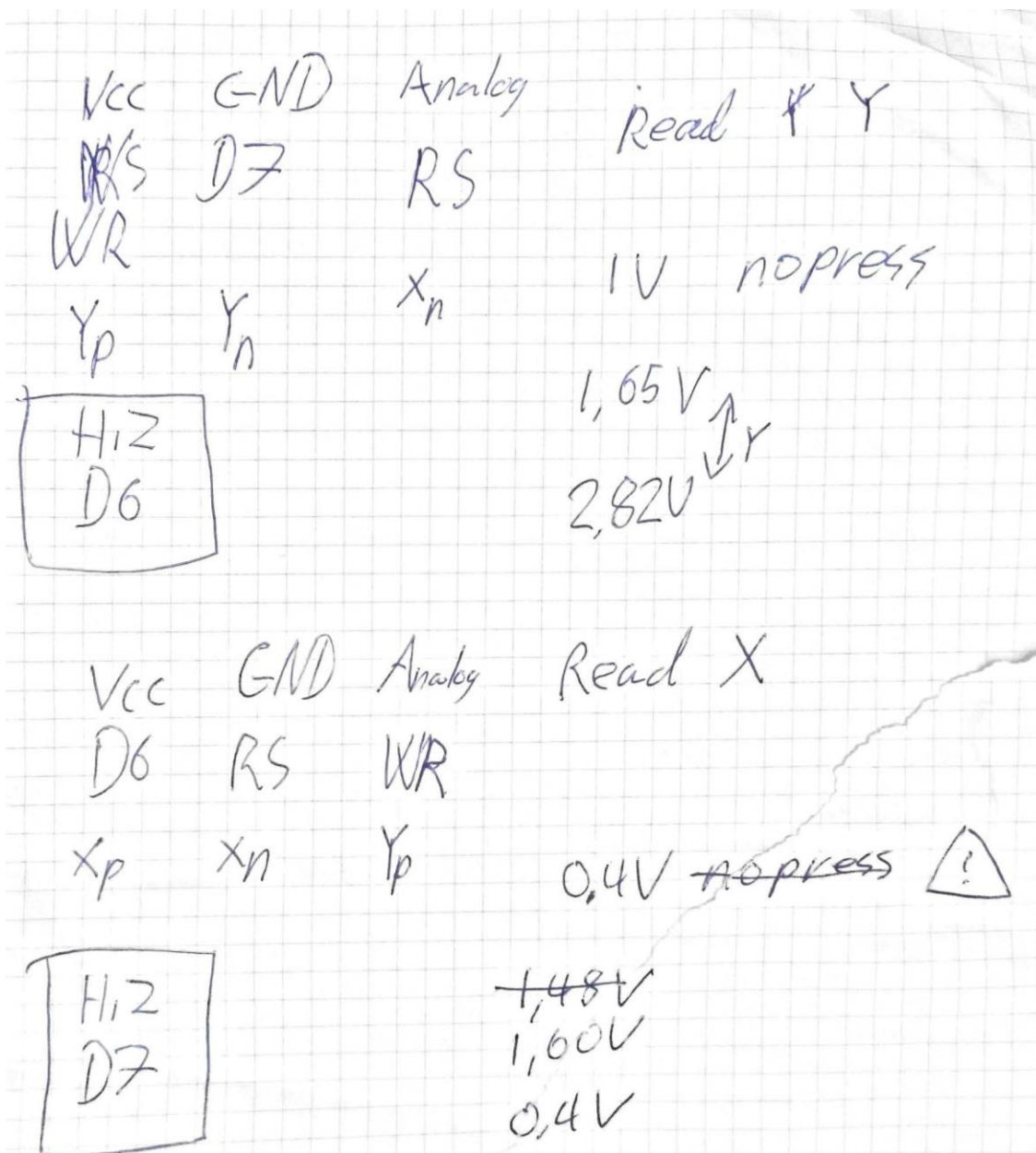
Starten af projektet var der en del problemer med at få TFT displayet op og køre. Som hjælp til at isolere fejlen, brugte vi en Arduino uno, hvor et lille test program er lagt op, der bare aflæse status register på TFT Displayet<sup>10</sup>. Udfra resultat fra aflæsningen kunne det hurtigt bekræfte at der var noget galt med TFT Displayet, og efter en udskiftning begyndte der at komme liv i TFT Displayet.

Touch på TFT Displayet var en del udokumenteret og på grund af dette har vi haft udført en række målinger på Touch delen.

	
<p>Test opstilling af TFT Displayet, måling på Touch.</p>	<p>Udførelse: Der bliver målt max og min værdier for X og Y, og med tryk og uden tryk. Alle disse målinger blev noteret ned, som efterfølgende kunne bruges i Implementation af Touch delen.</p>

<sup>9</sup> PSoC 4 Pioneer Kit Guide - <http://www.cypress.com/file/46056/download> d. 13/3-2017

<sup>10</sup> <http://forum.arduino.cc/index.php?topic=366304.msg2524865#msg2524865>



Figur 9 - Måle resultater fra TFT Displayet (Touch del).

Dette er testresultat fra målingerne fra Touch, der er vist i venstre side hvordan pins er sat op (f.eks. VCC -> WR, GND -> D7, Analog -> RS, HiZ -> D6 høj impedans). I højre side kan målingerne ses for Y kan man se at spændingen er 1V når intet rør skærmen, og at den går fra 1.65V til 2.82V fra min til max Y. Når der skal aflæses fra X i stedet, skal der måles på WR i stedet, og VCC på D6 og GND på RS.

## Konklusion

Projektet har været en stor succesoplevelse. Der har været opture og nedture, med lange fejlfindings timers, men det hele er endt godt. Når projektet blev defineret i starten af semestret var der mange overvejelser og ideer, og hvis der havde været mere tid, ville vi også kunne have udført de ting. Ting som bl.a. blev ekskluderet var en lyd enhed og SD-kort (der var knap nok lige nok plads til at have grafikken på PSoCen). Dog endte vi med et komplet fungerende spil, som kan styres med touch og hvor high scoren også kan gemmes i en emuleret EEPROM. Spillet kan endvidere køre med en tick rate på 60. Alt i alt var det et godt valg at arbejde på PSoCen, da dens arkitektur er smart og brugervenlig samt at den har en kraftig ARM processor til at køre programmet.

## Referencer

I dette afsnit omtales kort hvilke steder eller hvem man har lånt noget kode fra samt referencer til forskellige datasheets og benyttet materiale.

Eftersom der arbejdes på en PSoC og at hele miljøet kan ændre sig i forhold til de hardware elementer man benytter sig af, så bliver der auto genereret nogle API'er for de forskellige elementer. F.eks. hvis man hiver en ADC ind i topdesignet, så auto genereres der et API som man kan benytte sig af og dertil interface eller få ADC'en til at opføre sig på den ønskede måde. Den autogenererede kode fra PSoC Creator forekommer i mappen **'Generated Sources.'**

Herudover har vi også arbejdet videre på Hennings test program, som hurtigt kan vise om der er liv i en TFT skærm eller ikke. Denne funktion kan findes i **'TFT\_LCD.c'** filen ved navn TFT\_test().

- CYPRESS, NOTE: AN2376 – “Interface to Four-Wire Resistive Touchscreen”  
<http://www.cypress.com/file/134996/download>
- TFT LCD Single Chip Driver Datasheet  
[https://www.lpcware.com/system/files/ILI9488\\_Preliminary\\_DS\\_V090.pdf](https://www.lpcware.com/system/files/ILI9488_Preliminary_DS_V090.pdf)
- Tick beskrivelse - <http://minecraft.gamepedia.com/Tick>
- Universal Digital Block : <http://www.cypress.com/video-library/PSoC/intro-udb-components-psoc-5lp/386161>
- Terminal Count: <http://www.cypress.com/documentation/component-datasheets/timer>
- PSoC 4 Pioneer Kit Guide - <http://www.cypress.com/file/46056/download>