Space Invaders

Gruppe 11

NICE BILLEDE -----

Indholdsfortegnelse

[Indledning 2](#_Toc477128005)

[Tilegnelse af viden og problemer undervejs 2](#_Toc477128006)

[Overordnet projekt beskrivelse 3](#_Toc477128007)

[System beskrivelse 4](#_Toc477128008)

[Udvidet beskrivelse af topdesignet 4](#_Toc477128009)

[ADC\_TOUCH 5](#_Toc477128010)

[Sekvens diagrammer 6](#_Toc477128011)

[Flowchart / aktivitetsdiagram 6](#_Toc477128012)

[SpaceInvaders – temp billede 6](#_Toc477128013)

[Generel beskrivelse 7](#_Toc477128014)

[Touch driver 8](#_Toc477128015)

[TFTLCD driver 9](#_Toc477128016)

[Referencer 10](#_Toc477128017)

[Konklusion 10](#_Toc477128018)

# Indledning

Snakke om brug af psoc og lign…

Introducere projektet – space invaders spil

Indhold – hvad har vi tænkt at bruge/bruger

# 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.

# 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 arkitek-tur 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.

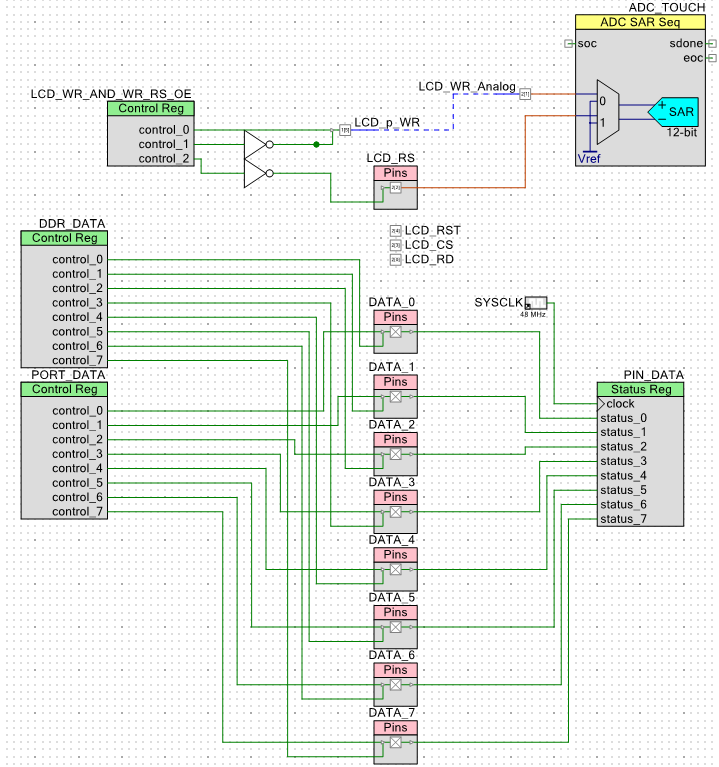


Figure 1 - Overblik over systemet og dets sammenhæng

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 – Forklaringen følger i **TOUCH** **AFSNITTET**). 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.

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 frames per second (fps), som vi gerne vil have.

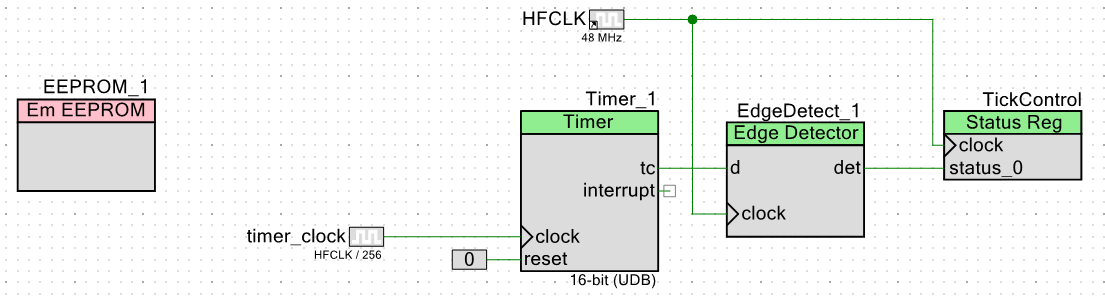


Figure 2 - Overblik over EEPROM og Tick Control

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

# System beskrivelse

I dette afsnit vil system arkitekturen blive beskrevet og dets funktionaliteter præsenteret på en overskuelig måde. Eksempelvis med aktivitets- og sekvensdiagrammer.

## Udvidet beskrivelse af topdesignet

### 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 størelse, så de endelige værdier ikke kommer til at overskride pixel opløsningen på de 320x480.

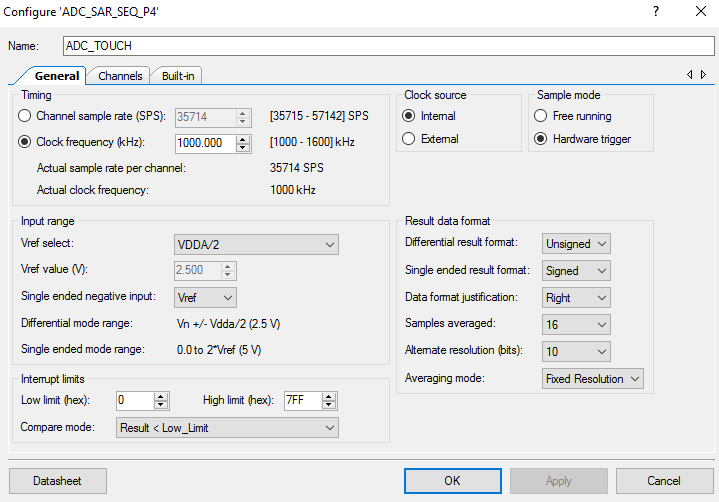


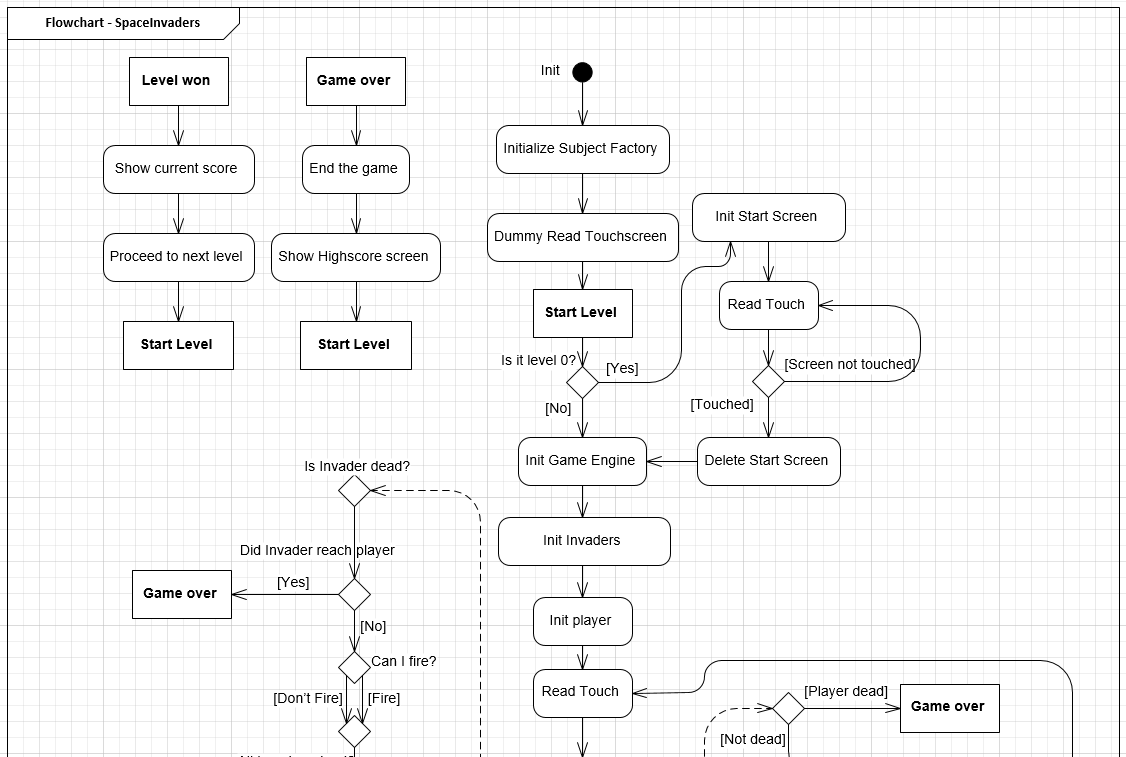
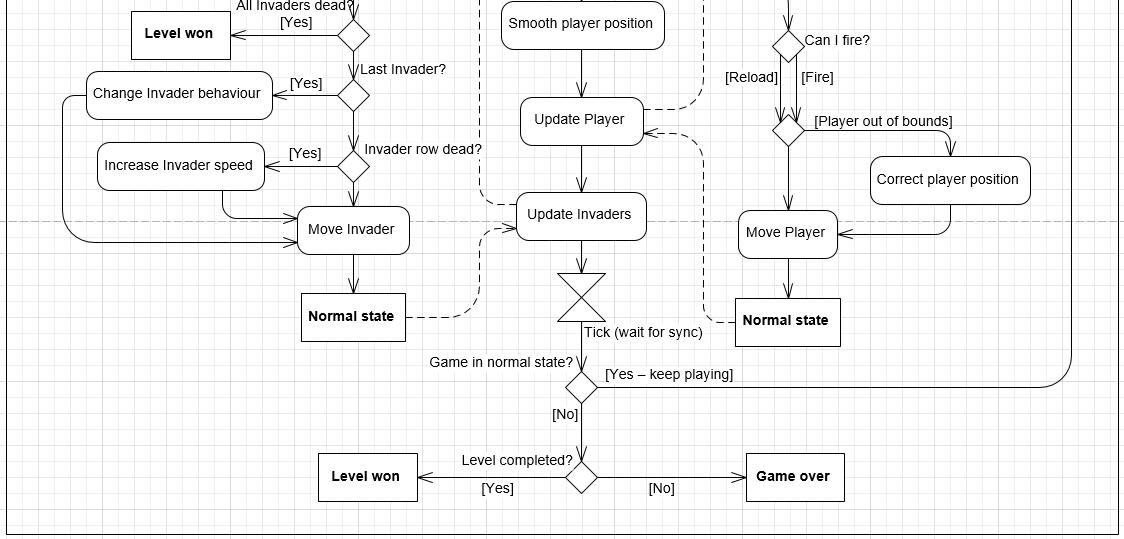
Figure 3 - Overblik over ADCens opsætning i PSoC Creator

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.

## Sekvens diagrammer

## Flowchart / aktivitetsdiagram

### SpaceInvaders – temp billede



## Generel beskrivelse

### 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 man evt. trådløst kunne styre figuren. Men vi endte 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.

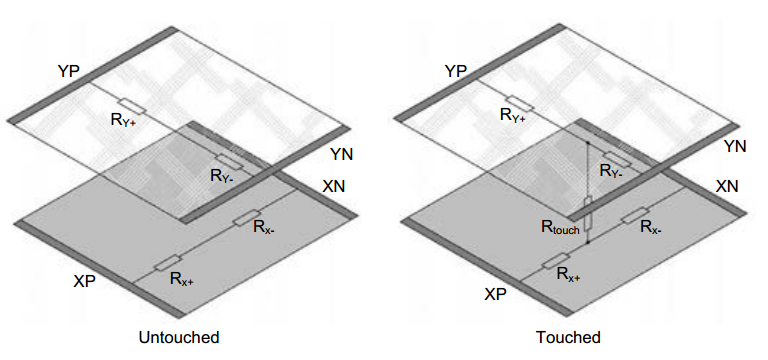
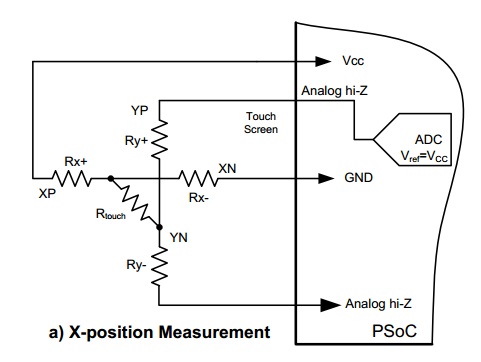


Figure 4 - Oversigt over en resistiv skærm samt hvordan man kan aflæse X positionen fra den

På Figure 4[[1]](#footnote-1) 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.

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**);**

**}**

Et udsnit af touch.c ses til venstre. Her udføres den opsætningen for at læse X-coordinaten 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.

### TFTLCD driver

Skærmen til vores projekt er en TFT skærm som er baseret på en ILI9488[[2]](#footnote-2) 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 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 actionen. 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.

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.

#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);

}

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.

. Herfra arbejdedes der videre med at blive gode til at sende data og teste på den testfunktion som Henning havde lagt op (Nævnes også iReferencerafsnittet).

# 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 APIer for de forskellige elementer. F.eks. hvis man hiver en ADC ind i topdesignet, så auto geneneres der et API som man kan benytte sig af og dertil interface eller få ADCen til at opføre sig på den ønskede måde. Den autogenererede kode fra PSoC Creator forekommer i mappen **’Generated Sources.’**

Herudover har vi også benyttet os af Hennings test program, som hurtigt kan vise om der er liv i en TFT skærm eller ikke. Denne funktion kan forefindes 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>

# Konklusion

1. <http://www.cypress.com/file/134996/download> d. 7/3-2017 [↑](#footnote-ref-1)
2. <https://www.lpcware.com/system/files/ILI9488_Preliminary_DS_V090.pdf> d. 20/2-2017 [↑](#footnote-ref-2)