# System Description (BDD, Sekvens etc)

# System architecture and user interface

## Tekniske overvejelser – individuelt

## Fordele og ulemper ved forskellige implementeringer – individuelt

## Topdesign indepth (guldkorn)

### EMULATED EEPROM – Jonathan

Der var I projektet brug for noget non-volatile hukommelse ikke meget men bare nok til at gemme high score. Dog var der den ulempe at PSoC4 ikke har indbygget nogen EEPROM som ellers ville have været optimal for opgaven.

For at løse dette problem har Cypress lavet en emulator der giver en mulighed for at tilskrive til flash området hvor koden ligger. Dette område er som standart kun et ”read only” område, og PSoC4 skal være i en rigtig tilstand for at tilskrive til flash området.

Eksampel: udsnit fra “GameEngine.c”

static const uint8 CYCODE HighScore**[**4**]** **=** **{**0**};** // fortæl at dette skal gemmes på flash ved at bruge postfix CYCODE der fortæller hvor variablen skal gemmes.

**...**

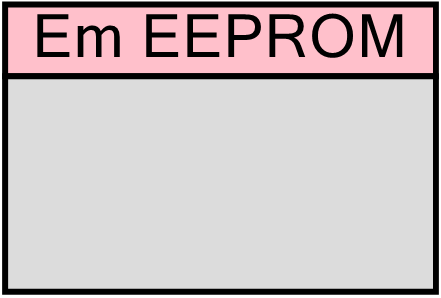
uint8 newHighScore**[**4**];**

uint32 HighScore\_ **=** **(\*(**volatile uint8 **\*)&**HighScore**[**0**]);** // aflæsning fra flash område.

**...**

status **=** EEPROM\_1\_Write**(**newHighScore**,** HighScore**,** 4**);** // Hjælpe funktion fra Cypress der sørger for at tilskrivning forgår korrekt.

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

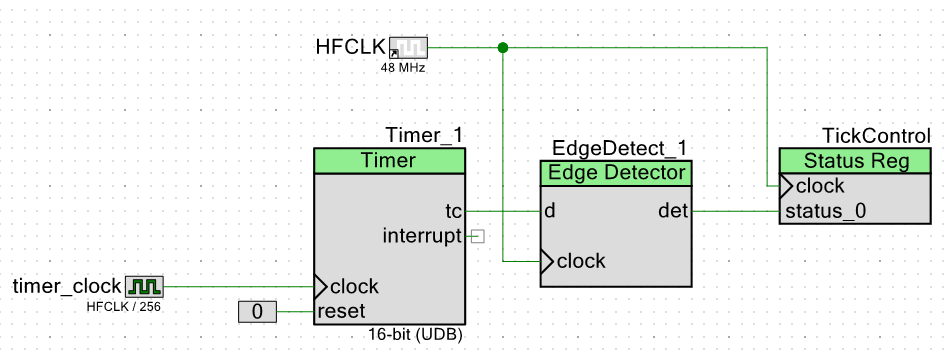


Figur 1 - Cypress EMULATED EEPROM, som tilføjer et API til at kunne sikret tilskrive til FLASH området hvor Koden ligger.

### TICK Controller – Jonathan

På grund af den måde der er valgt at bygge spillet på, er der påkrævet at spillet skal fixet tick interval, for at sikre at spillet ikke lige pludseligt køre hurtigere eller langsommer fra tick til tick.

“*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*** “[[1]](#footnote-1)

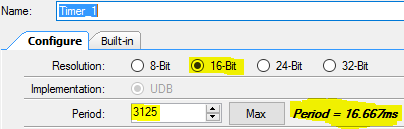


Figur 2 – Hardware opsætning af TICK Controller. Der er brugt 4 komponenter til at bygge TICK Controlleren. En Timer blok der er sat op til at genere et trig hvergang den rammer en ønsket værdi hvor efter den vil resette sig selv til 0 igen og tælle op igen. Til at trigge Timer’en er der brugt en nedskaleret clock af HFCLK. På grund af der er brugt en nedskalleret clock af HFCLK behøves der ikke nogen Hardware til at synkronisere signalerne, men fordi vi kun er interesseret i trigget skal der bruges en EdgeDetect komponent, for ellers vil det ligne at den halv delen af tiden genere en masse trig’s. Som det sidste trin er der brugt et Status register, der er sat op til at være et sticky register (Vil blive holdt høj, hver gang der kommer et trig, indtil en aflæsning sker hvor den så bliver kørt lav igen).

Opsætning:

Timer\_clock: grunden til der er brugt en nedskalleret clock af HFCLK, skyldes at der ikke kunne implementeres en stor nok timer på grund af mangel på UDB’er[[2]](#footnote-2).

Timer\_1: Timeren er sat up til at genere trig(tc[[3]](#footnote-3) signalet går høj) hver gang dens værdi er lig med 0. I dette design er den sat til at genere et trig hvert 16.67ms, som svare til 60 trig i sekundet.



EdgeDetect\_1: Bruges til at detektere hver gang tc signalet går højt.

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

Det eneste der kræver for at få det hele til køre over i software domænet er at initialisere Timeren. Og efter hvert tick forløb vente på at TickControl er 1.

Udklip af kode implementationen af TICK Controller på software sidden, udklippet er fra ”SpaceInvader.c”

**while(**1**)**

**{**

**...** //game code

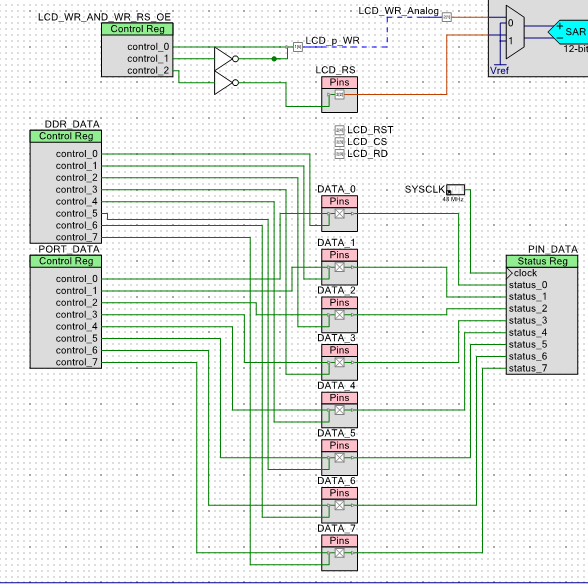
**while(**TickControl\_Read**()** **==** 0**);**

**}**

Denne løsning har et par ekstra fordele, f.eks. hvis der er et tick der tager for lang tid, (f.eks. hvis mange skyd skal allokeres) vil den komme ud vente løkken med det samme, og få mulighed for at indhente næste tick. Selvfølgelig er der mulighed for at hvis tick hastigheden er for høj at spillet ikke kan følge med, og derfor ikke vil opfange alle Trigs fra TICK Controlleren. Dette er selvfølgelig noget der skal indstilles lidt på, for at få den helt rigtige værdi.

### TFT\_Control - Register control – Jonathan

TFT Skærmen der blev brugt havde pin layout til en Arduino uno, som passet fint efter som den enhed der var valgt til dette projekt var en psoc 4 pioneer kit[[4]](#footnote-4), dog var der en lille ulempe port layoutet var ikke ens, og ville koste mange software operationer bare at sætte data pins op til den rigtige værdi, for at løse dette problem, blev der brugt et par control register 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.



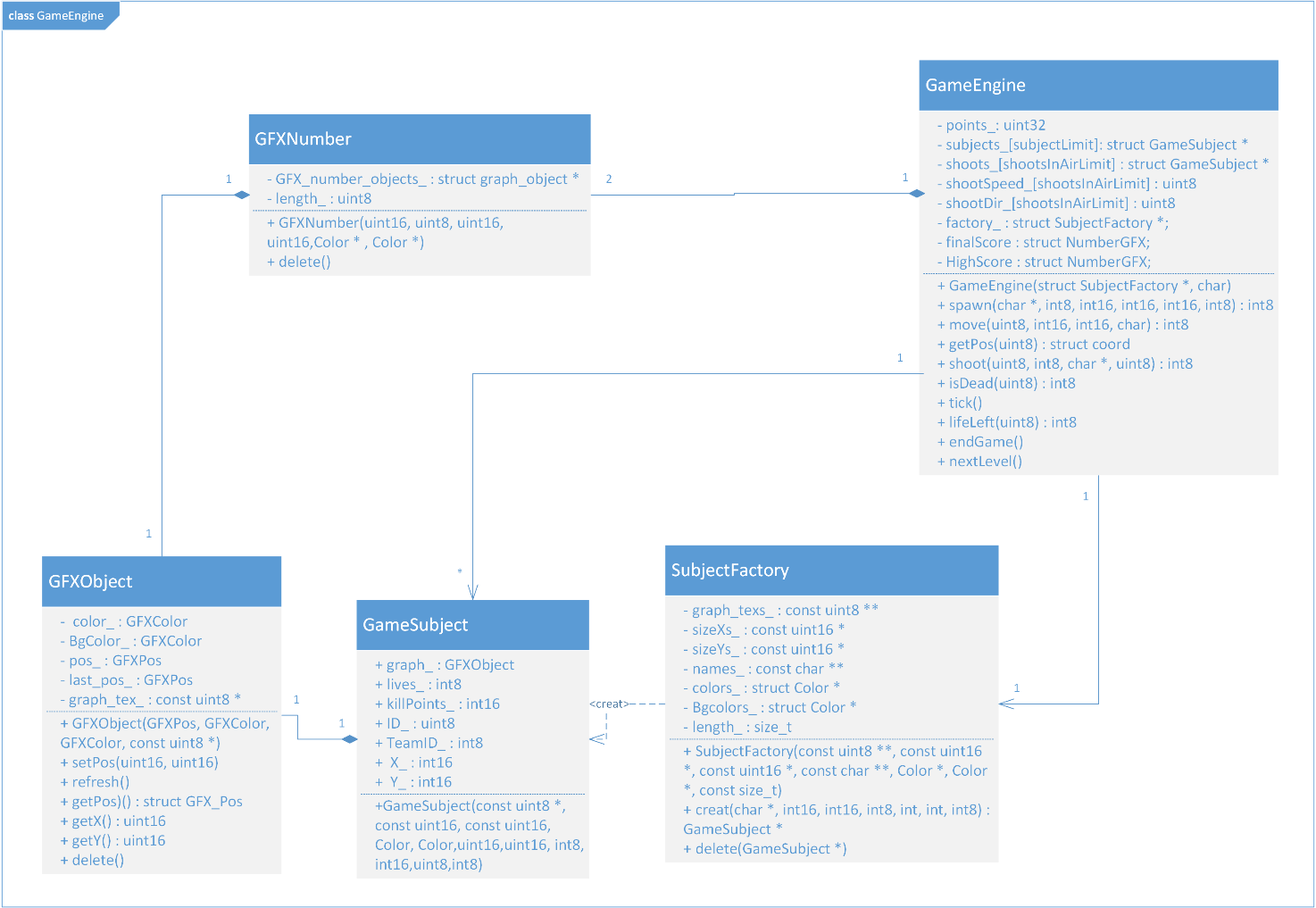
Figur 3 – Til DATA bene er der brugt to control register ”DDR\_DATA” og ”PORT\_DATA”, og et status register PIN\_DATA. DDR bruges til at aktiver output, PORT bruges til at tilskrive værdier til bene, PIN kan aflæse fra bene. Der er bygget et lidt speciel control register ”LCD\_WR\_AND\_WR\_RS\_OE”; control\_0 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\_skærmen allerede var implementeret før dette control register blev lavet, for at sikre at der ikke skulle opstå uventet resultater efter Implementation, da ”LCD\_WR\_AND\_WR\_RS\_OE”, kunne bruges som bare et control register for ”LCD\_WR”. ”LCD\_p\_WR” og ”LCD\_WR\_Analog” er vores ”LCD\_WR”; ”LCD\_p\_WR” ”p’et” er en husker på at den skal styres fra ”LCD\_WR\_AND\_WR\_RS\_OE”; ”LCD\_WR\_Analog” er en analog pin der er lodet til ”LCD\_p\_WR” så det er muligt at måle spændingen, bruges i touch delen.

”LCD\_p\_WR” og ”LCD\_WR\_Analog” kan virke som en lidt mærkelig løsning især når man kigger på ”LCD\_RS” hvor det hele er samlet i et ben. Grunden til dette skyldes PSoC4 arkitekturen, man kan ikke have en hardware digitalinput og OE sammentid med et analog input ben. Selvfølgelig kunne dette løses ved at undlade at have den hardware digitalinput, dette vil koste omkring 6 ekstra clock perioder, dette ser ikke ud af meget, men eftersom dette ben skal toggels 4 gange for hver pixel tilskrivning, runder det hurtigt op og gøre koden meget langsom. Hvor denne løsning kun kræver en enkel register tilskrivning.

## Klassediagrammer

## GameEngine

GameEngine spiller en central rolle i produktet, og står for håndtering af subjecter og giver brugeren mulighed for at bruge et par basale funktioner på dem, som at flytte dem, tjekke deres status så som liv, eller affyre et skud af, som GameEngine også håndtere. GameEngine gemmenløber ikke alle Subjecter hver gang en metode bliver kaldt. F.eks. hvis der skal flyttes på 16 Subjecter før det næste tick, vil de ikke blive opdateret løbende, men først når metoden tick bliver kaldt i GameEngine, hvor den vil opdatere nye positioner, skyd simulering og check om der er et subject der er ramt af et farligt skyd (et skyd med et andet hold nummer end det nummer subjectet har.)



Figur 4 -Klassediagram GameEngine. – GameEngine er den klasse der står for håndtering af subject og styre dem udfra de kommandoer der ønskes hvert tick, måde man styre et subject på, er via det unikke ID man får returneret når spawn metoden bliver brugt. Med ID’et kan man flytte subjectet, udføre handlinger (kun skyd er implementeret) eller tjekke hvor mange liv subjectet har eller det dødt. Den måde GameEnige spawner et nyt subject på, er ved at bruge SubjectFactory til allokere og oprette og initialiser GameSubject med metoden ”creat” hvor den så returner en pointer til subjectet, hvis den er løbet tør for hukommelse returner den bare en NULL pointer, som vil medføre at spawn kommandoen vil returner -1. GameSubject Indeholder en række oplysninger, koordinater (den position den vil have 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 subject bliver drabt), lives (Antallet af liv et subject), graph (Er det grafiske object der repræsentere subjectet). GFXObject er det 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.

## Sekvensdiagrammer

### GameEngine – Jonathan

### SubjectFactory – Jonathan

### GameSubject – Jonathan

### GFX – Jonathan

# All test results

F.eks. debug ben til touch.

Analogdiscovery dump af analog pins.

## TFT TEST –Jonathan?

## TOUCH TEST – Jonathan?

1. <http://minecraft.gamepedia.com/Tick> (Det eneste sted som havde en fin beskrivelse af hvad et tick var.) [↑](#footnote-ref-1)
2. Universal Digital Block : <http://www.cypress.com/video-library/PSoC/intro-udb-components-psoc-5lp/386161> [↑](#footnote-ref-2)
3. Terminal Count: <http://www.cypress.com/documentation/component-datasheets/timer> [↑](#footnote-ref-3)
4. <http://www.cypress.com/documentation/development-kitsboards/cy8ckit-042-psoc-4-pioneer-kit> [↑](#footnote-ref-4)