

# **Core Wars**

**TSEA43**

Projektrapport

Jonas Hietala  
Jesper Tingvall  
Jizhi Li

7 Maj, 2012



## **Sammanfattning**

I detta projekt har vi byggt en mikrodator som använder Redcode som assembler. Detta för att kunna spela spelet Core Wars. Vi använder en UART för att skriva in koden till datorns minne och vi kan dumpa ut minnesinnehållet och spelets status på en skärm genom VGA-porten. I rapporten går vi igenom lite Redcode, beskrivning av hårdvara till vår mikrodator och hur vi använder RS232 och VGA standarden. Till sist har vi en del exempel Warriors som visar de vanligaste Core Wars strategierna.

# Innehållsförteckning

<b>1</b>	<b>Inledning</b>	<b>1</b>
<b>2</b>	<b>ICWS88 Redcode</b>	<b>2</b>
2.1	Introduktion . . . . .	2
2.2	Exempel Warriors . . . . .	2
<b>3</b>	<b>Teori</b>	<b>4</b>
3.1	VGA . . . . .	4
3.2	RS232 . . . . .	5

# 1 Inledning

Vårt mål med projektet i denna TSEA43 kurs var att bygga en dator som kunde köra det eminenta spelet Core Wars. Core Wars är ett ointeraktivt spel där spelarna skriver sina program i Redcode assembler. Målet var att bygga en maskin som använde Redcode som sin assembler och som kunde måla ut spelområdet, d.v.s. minnet, till en VGA skärm och ta emot ny kod via en UART. För mer utförlig information om våra designmål rekommenderas en läsning i vår designskiss <sup>1</sup>.

Vi namnger vår dator till M.A.R.C, Memory Array Redcode Computer då simulatorm heter M.A.R.S, Memory Array Redcode Simulator. Värt att nämna är att Core Wars ej refererar till processor kärnan utan till ett gammalt kärnminne.

Vårt mål är att kunna spela Core Wars enligt 1988 standarden<sup>2</sup>, skicka in innehåll till M.A.R.C från en kontroll dator och sätta ut två spelares position. Vi vill även kunna dump ut minnesinnehåll och spelstatus till en VGA skärm. Vår uDator skall kunna utföra alla 10 instruktioner och 4 adresseringsmoder Redcode har samt kunna växla mellan, skapa och ta bort processer. Vi rekommenderar en läsning utav 1988 standarden då vi ej kommer att gå igenom instruktionerna eller adresseringsmoderna i denna rapport.

---

<sup>1</sup>Se bilaga TODO

<sup>2</sup><http://corewars.nihilists.de/redcode-icws-88.pdf>

## 2 ICWS88 Redcode

### 2.1 Introduktion

Vi har programmerat en assembler som kan generera en binärfil ifrån två Warriors skrivna i Redcode. Vi randomiserar också deras startläge. Vi kan sedan skicka den assemblerade koden och startpositionerna till MARC genom UART.

### 2.2 Exempel Warriors

#### Replicator

Replicators skapar kopior av sig själva och förökar sig i minnet. De motverkar bombers då bombers inte kan förstöra replicatorn tillräckligt snabbt.

#### Factory Bomber

Factory bomber (eller bomber factory då den bygger bombers) formaterar hela minnet via att masskopiera en massa 'little bombers' till minnet. Dessa databombar minnet och kommer efter ett tag bomba isär originalkoden. Denna Warrior är därmed en blandning mellan en bombare och en replicator.

#### Carpet Bomber

Carpet bombers är en blandning mellan en bomber och en scanner. De traverserar minnet och lägger in bombers där minnet har ändrats. Denna Warrior är smartare än en vanlig bomber då den inte kommer att bomba ute i tomma minnet. Den kommer också vara lite snabbare än en traditionell bomber som behöver kopiera ut data.

#### Imp Spawner

Denna Warrior är ej offensiv och har som strategi att skapar en massaimps. Imp spawner fungerar ungefär som Factory Bomber fast har en annan payload.

#### Vampire Bomber Gate Replicator

Denna otympliga Warrior startade som ett skämt då vi ville se vad som hände om man inkluderade så många strategier som möjligt i en Warrior. Dock blev den inte så dålig som vi först trodde. Först så skapar Warriorn en kopia av sig själv, denna kopia kan dock ej kopiera sig själv, något som borde kunna lösas med hjälp av lite hjärnverksamhet och en texteditor. Efter kopiatorn så har Warriorn en "bomber cage", dessa två rader databombar minnet bakåt. Efter cagen kommer vampyrkoden. En vampyr JMP bombar minnet i hopp om att fienden skall hoppa in i dess cage. Den kan därmed sno klockcykler ifrån motståndarens kod. Sist finns en gate ifall resten av koden skulle bli överkörd av en Imp.

#### Kopimi

Denna Warrior scannar minnet efter information, kopierar den och börjar sen exekvera den. Den kan därmed härma en fientlig Warrior om den skulle hitta den. Fungerar skapligt trots att den utvecklades mest för att se vad som hände om man skulle tolka Det Missionerande Kopimistsamfundet missionsbudskap<sup>3</sup>; "Kopiera och sprid" i form av Redcode. Denna Warrior använder replicator strategin.

---

<sup>3</sup><http://kopimistsamfundet.se>

**Inseminator**

Ännu en Warrior som skapades på skoj men som visade sig vara rätt så effektiv. Den letar upp motståndarens kod och injicerar en massa processer i den i hopp om att motståndaren ej ska förstöra sig egen kod. Detta brukar dock förstöra funktionaliteten i motståndarens kod då den förutsätter oftast att koden exekveras sekventiellt.

**Core Cleaner**

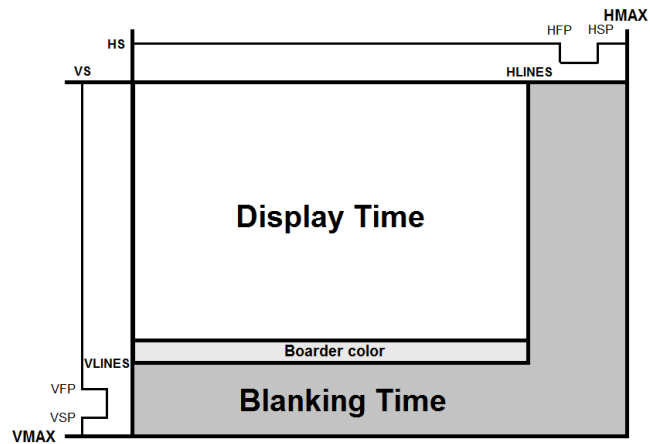
En core cleaner är ett program som databombar hela minnet. Ofta går man igenom minnet två gånger, den första fyller man minnet med split instruktioner för att slöa ner motståndaren och sedan med DAT-instruktioner för att göra slut på honom.

**Dwarf Scout**

En dwarf scout är en enkel bomber som skyddar sig mot andra bombers genom att se om någon ändrar i minnet i dess närhet. Om så är fallet så kommer den att hoppa till en ny plats i minnet och ta med sig sina processer.

### 3 Teori

#### 3.1 VGA



Figur 1: VGA teori

När VGA skickar pixeldata till VGA porten kommer skärmen inte ta emot och visa pixel data under hela tiden. Dessutom finns det en speciell timing till olika upplösningar med olika frekvenser. Upplösning 640x480 med frekvens 60Hz har vi följande timing enligt Digilent®.

- HMAX: 800
- VMAX: 525
- HLINEs: 640
- VLINEs: 480
- HFP: 648
- HSP: 744
- VFP: 482



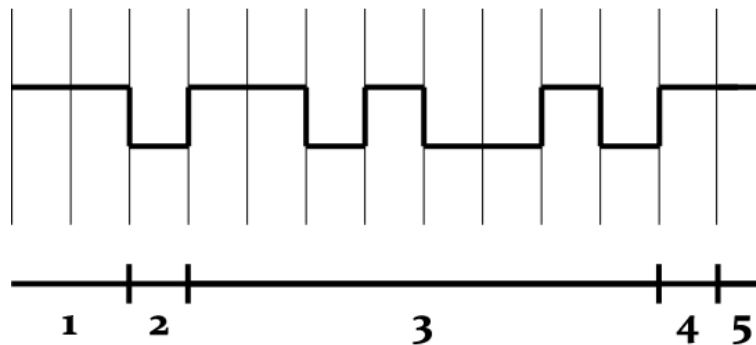
- VSP: 484
- Clk: 25MHz

Vi har blanking time för att en skärm använder en stråle för att visa varje pixel och strålen flyttar sig från vänster till höger och sedan ner på nästa rad och upprepar denna process. Under blanking time kommer strålen flytta sig från höger till vänster och under denna tid ska skärmen inte visa någon pixel. Mellan front porch och back porch går sync signal ner och upp igen på grund av att det är sync signal som uppdaterar och bestämmer frekvens till skärmen.

På display ytan, kommer varje pixel uppdateras enligt den 8 bitars färg som skärmen har fått genom VGA porten och på blank ytan ska vga porten få ingen färg data alls, annars kommer skärmen visa denna färg när de flyttar sig tillbaka över skärmen.

### 3.2 RS232

Vårt FPGA kort har en USB till RS232 port. Vi använde denna för att föra över den assemblerade spelarkoden till kortet. En överföring inleds av en startbit, därefter följer 8 databitar och en stoppbit. Hastigheten mäts i baud, tecken (på 8 bitar) per sekund. I vårt fall var ledningen hög när ingen överföring var igång (1). Överförningen inleds med att ledningen jordas (2), därefter följer 8 databitar i vald hastighet (3). I slutet av överförningen kommer en stoppbit som är hög (4). Se figur 2.



Figur 2: En RS232 överföring. Man har ingen gemensam klocka för sändare och mottagare utan överför endast data, sändaren och mottagaren känner dock till vilken baud rate man överför med. I vårt fall använder vi 115 200 baud.