BPTI: Bomberman in VHDL

3. Februar 2018

Teil I.
Beschreibung der Entities

1. bomberman_ent

2. game_mechanic_ent

2.1. game_state_ent

2.2. player_ent

2.2.1. movement_ent

2.2.2. clk_movement_ent

Die Entity clk_movement_ent ist ein Clock Divider, die Architecture wird funktional beschrieben. Die Entity wird genutzt, um die Geschwindigkeit der Bewegung der Spieler zu steuern. Es wird alle 78125 Originaltakte die Ausgabe Ausgabe negiert. Da in unserem Design die VGA-Clock genutzt wird, wird somit alle $\frac{78125*2}{25.175*10^6} s \approx 6.2ms$ ein Takt erzeugt. Dies hat zur Folge, dass bei ständiger Bewegung eines Spielers in die gleiche Richtung, die pro steigender Takt-Flanke des ausgegebenen Takts stattfindet, die Bewegung vom linken bis zum rechten Rand des Spielfelds (480 Pixel) $480*6.2ms \approx 2.98s$ dauert. Diesen Wert haben wir bei BombermanGB über einen Emulator gemessen und „rückwärts“ den nötigen Zähler, also 78125, bestimmt.

2.2.3. bomb_ent

3. graphics_ent

3.1. pixel_gen_ent

3.2. Sprites

In den beiden Entities PlayerROM und SpriteROM befinden sich Sprites, die in Abhängigkeit des aktuellen Zustands der Arena, der Spieler und der Bomben RGB-Werte an den VGA-Controller ausgeben. Dazu befinden sich in den beiden Entities PlayerROM und SpriteROM pro Sprite (es existieren Sprites für: Spieler 1, Spieler 2, leerer Block, zerstörbarer Block, unzerstörbarer Block, Bombe, Explosion) ein zweidimensionales konstantes Array (wird synthetisiert als ROM). Diese Arrays sind jeweils 32 * 384 Bit groß. Die Größe ergibt sich folgendermaßen: Die Kacheln des Spielfeldes sowie die Quellsprites haben eine Größe von 32 * 32 Pixel. Da für jeden Farbkanal des VGA-Ausgangs 4 Bit zur Verfügung stehen haben wir in dem Array den RGB-Wert jedes Pixels mit 3 * 4 Bit (also 3 Hex-Werte) kodiert. So kommen 32 Zeilen mit jeweils 3 * 4 * 32 = 384 Bit zustande. Die Sprites haben wir aus einem Sprite-Sheet von opengameart (`https://opengameart.org/sites/default/files/DungeonCrawl_ProjectUtumnoTileset.png`) ausgeschnitten und mithilfe eines Java-Programmes, welches im Abgabe-Verzeichnis zu finden ist und im folgenden kurz beschrieben wird, in VHDL-Arrays umgewandelt.

3.2.1. SpriteExtractor.java

Das Java-Programm bekommt als Kommandozeilenargumente die Pfade zu den 32 * 32 Pixel großen Sprites. Über jedes der Sprites wird pixelweise iteriert und für jeden Pixel der RGB-Wert ausgelesen. Daraus werden die Werte der einzelnen Farbkanäle bestimmt, welche anschließend normiert werden, da auf dem FPGA nur 4 Bit pro Farbkanal nutzbar sind. Die normierten Werte werden in Hex-Character umgewandelt und auf der Konsole ausgegeben, sodass die RGB-Kanäle jedes Pixels mit 3 Hex-Charactern dargestellt werden. Die Ausgabe erfolgt in der Form von 32 komma-separierten std_logic_vectoren der Länge 384.

3.2.2. SpriteROM

Die Entity SpriteROM wird mit einer Architecture funktional beschrieben. Abhängig von der sprite_id wird mittels sprite_row und sprite_col asynchron auf die verschiedenen

Arrays, die Sprites für den leeren/zerstörbaren/unzerstörbaren Block enthalten, zugegriffen. Die sprite_id orientiert sich hierbei an den Kodierungen des Spielfeldes (siehe game_state, z.B. leerer Block = x0). Ist keine gültige id gesetzt (also id > x1 und id < xD), wird RGB schwarz ausgegeben.

3.2.3. PlayerROM

Die Entity PlayerROM ist hinter SpriteROM geschaltet und überschreibt asynchron die RGB-Werte, die von SpriteROM ausgegeben werden, wenn sich im aktuellen Pixel des Bildes ein Spieler befindet. In PlayerROM befinden sich zwei konstante Arrays, die in der gleichen Form wie in SpriteROM die Sprites für die Spieler enthalten. PlayerROM bekommt zum Auslesen der Arrays eine id (x2 für Spieler 1, x3 für Spieler 2), sowie die x- und y-Position des Spielers im aktuell betrachteten Pixels. Ist die id gültig, wird der RGB-Wert des entsprechenden Sprites gelesen und ausgegeben. Überlappen sich beide Spieler, so wird stets Spieler 1 gezeichnet. Ist keine gültige id gesetzt (also id > x3 oder id < x2), wird der RGB-Wert von SpriteROM durchgeschaltet. Hat ein Pixel den (in unseren Sprites speziellen) RGB-Wert xEEE wird ebenfalls der RGB-Wert von SpriteROM durchgeschaltet. Dies sorgt dafür, dass die Spielfiguren erkennbare Strukturen haben (Spieler sind keine Kacheln) und sich sichtbar vor einem Hintergrund bewegen.

3.3. rgb_assign_ent

In der Entity rgb_assign_ent werden die Farbausgänge für den Monitor gesetzt. Dazu wird bitweise auf die RGB-Eingangsvektoren zugegriffen und die einzelnen Ausgangsbits gesetzt. Diese Vorgehen am Ende der Bearbeitung erlaubt es uns während den Berechnungen mit drei Vektoren für die Farben zu arbeiten, anstatt immer 12 einzelne Bits zu benutzen.

4. sync_gen_ent

Die Entity sync_gen_ent wird mit einer Architecture strukturell beschrieben durch die Unterkomponenten hsync und vsync. In dieser Entity werden die Signale für den VGA-Controller generiert (also hsync und vsync). Desweiteren werden zum Setzen der Pixel die aktuelle row und column ausgegeben. An dieser Stelle sei auf das VGA-Übungsblatt und den Bericht darüber verwiesen. Wir haben unsere Lösung von dem Übungsblatt für das Projekt übernommen und im Laufe des Projekts nichts an der Entity inklusive Unterkomponenten gerändert.

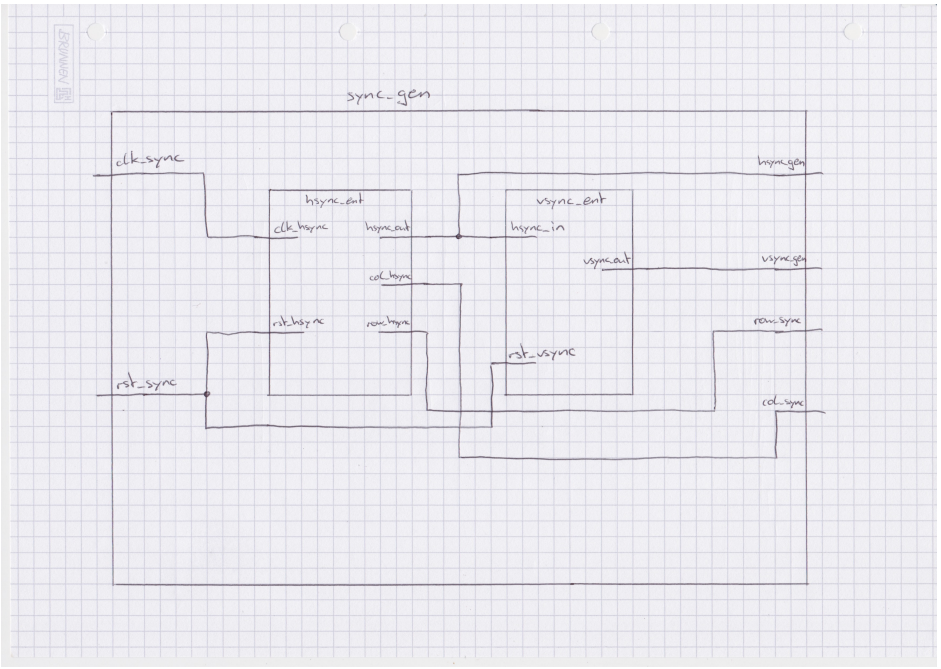

Abbildung 1: Blockdiagramm sync_gen_ent

Teil II.
Probleme

5. bomberman_ent

6. game_mechanic_ent

6.1. game_state_ent

6.2. player_ent

6.2.1. movement_ent

6.2.2. clk_movement_ent

6.2.3. bomb_ent

7. graphics_ent

7.1. pixel_gen_ent

7.2. Sprites

7.2.1. SpriteExtractor.java

7.2.2. SpriteROM

7.2.3. PlayerROM

7.3. rgb_assign_ent

8. sync_gen_ent

Ein Problem, das schon bei der Lösung des Übungsblatts aufgetaucht ist, hat sich auch durch das Projekt gezogen. Die VGA-Ausgabe funktioniert nicht an allen Monitoren

ordnungsgemäß (flackern). An unserem Test-Monitor hat die VGA-Ausgabe nach einigen Einstellungen jedoch funktioniert.