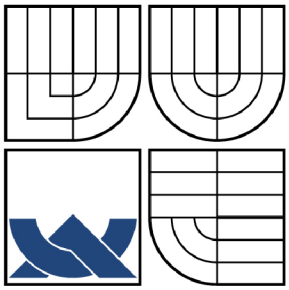
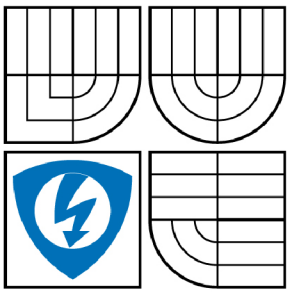


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV MIKROELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF MICROELECTRONICS

APLIKAČNÍ ROZHRANÍ PRO PODPORU GRAFIKY V JAZYCE VHDL

APPLICATION INTERFACE FOR HANDLING GRAPHICS IN VHDL LANGUAGE

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

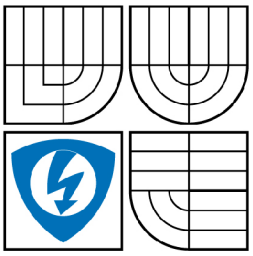
AUTOR PRÁCE — Bc. PETR VLČEK

AUTHOR

VEDOUCÍ PRÁCE — Ing. MAREK BOHRN

SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav mikroelektroniky

Diplomová práce

magisterský navazující studijní obor
Mikroelektronika

Student:	Bc. Petr Vlček	***ID:***	22533
Ročník:	2	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

Aplikační rozhraní pro podporu grafiky v jazyce VHDL

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit rozhraní pro manipulaci s grafickými daty v paměti v jazyce VHDL.
Detailně prostudujte problematiku metod pro práci s grafickými daty v paměti. Vytvořte v jazyce VHDL grafický modul podporující vybrané metody. Vytvořený modul musí být jednoduše připojitelný k řídicím obvodům, například mikrokoprocesoru PicoBlaze nebo externímu rozhraní. Zdrojové kódy optimalizujte pro použití v obvodech řady Spartan-3.

DOPORUČENÁ LITERATURA:

Dle pokynů vedoucího práce.

Termín zadání:	9.2.2009	***Termín odevzdání:***	29.5.2009

Vedoucí práce: Ing. Marek Bohrn

prof. Ing. Vladislav Musil, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt:

Cílem této práce je vytvořit rozhraní pro generátor obrazu. Rozhraní generuje VGA signál s možností až 4 bitů barevné hloubky, ovládá 2 čipy jednoportové SRAM IS61 dodávané spolu s Digilent Spartan-3 Starter Kit Board a komunikuje prostřednictvím FIFO bloků na principu posuvných registrů. Grafické prostředí generuje čáry a odvozené tvary, kružnice a odvozené tvary, vyplňuje oblasti a ovládá 2D transformace obrazu.

Abstract:

The objective of this thesis is creating interface for the picture generator. The interface generates a VGA signal with possibility of 4bit color depth. The interface controls two chips of one port SRAM IS61 witch is supplied with Digilent Spartan-3 Starter Kit Board and comunicates trought FIFO blocks based on the shift register principle. Graphics interface generates lines and secondary forms, circles and secondary forms, fills area up and controles 2D transformations of picture.

Klíčová slova:

grafické prostředí, bresenhamovy algoritmy, digitální diferenciální analyzátor, semínkové vyplňování, VGA, SRAM paměť, posuvné registry

Keywords:

graphics interface, bresenham`s algorithms, digital differential analyzer, seed fill, VGA, SRAM memory, shift registers

Bibliografická citace díla:

VLČEK, P. Aplikační rozhraní pro podporu grafiky v jazyce VHDL. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 66s. Vedoucí diplomové práce Ing. Marek Bohrn.

Prohlášení autora o původnosti díla:

Prohlašuji, že jsem tuto vysokoškolskou kvalifikační práci vypracoval samostatně pod vedením vedoucího diplomové práce, s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 28. 5. 2009

.....
Petr Vlček

Poděkování:

Chtěl bych touto cestou poděkovat vedoucímu diplomové práce Ing. Marku Bohrnovi za ochotu a trpělivost s jakou mne vedl při zpracovávání diplomové práce.

OBSAH

Seznam obrázků

Seznam tabulek

1 Úvod

V mnoha uplatněních mikroelektroniky je žádoucí mít k dispozici nějakou formu grafického výstupu – ať už se jedná o rychlou a přehlednou kontrolu průběžně měřených údajů – např. na výrobní lince – nebo třeba jako jednu z možností diagnostiky. Jednou z možností je použití LED displejů, které budou umístěny přímo na desce plošných spojů – tato varianta v sobě má nebezpečí nekompatibility jednotlivých displejů a následné přizpůsobování zdrojových kódů grafického prostředí konkrétnímu použití. Druhou variantou je pak použití standardního VGA zařízení, jejichž ceny i rozměry v posledních letech výrazně klesají. Výhodou tohoto řešení je pak použití standardů, které každé zobrazovací zařízení musí splňovat.

Tato práce vznikla z potřeby mít k dispozici efektivní nástroje pro zobrazování na zařízeních podporujících standard VGA. Zároveň řešení použitá při zpracování této práce jsou snadno přenositelná a lze tak zvolit čip nižší nebo vyšší třídy dle potřeby, stejně tak lze relativně snadno nahradit SRAM paměti, atp.

V práci jsou popsány povětšinou algoritmy, které byly použity jako základy pro vytváření programového vybavení. Uvádění zdrojových kódů v tištěné podobě by nebylo účelné, lze je snadno dohledat na přiloženém CD.

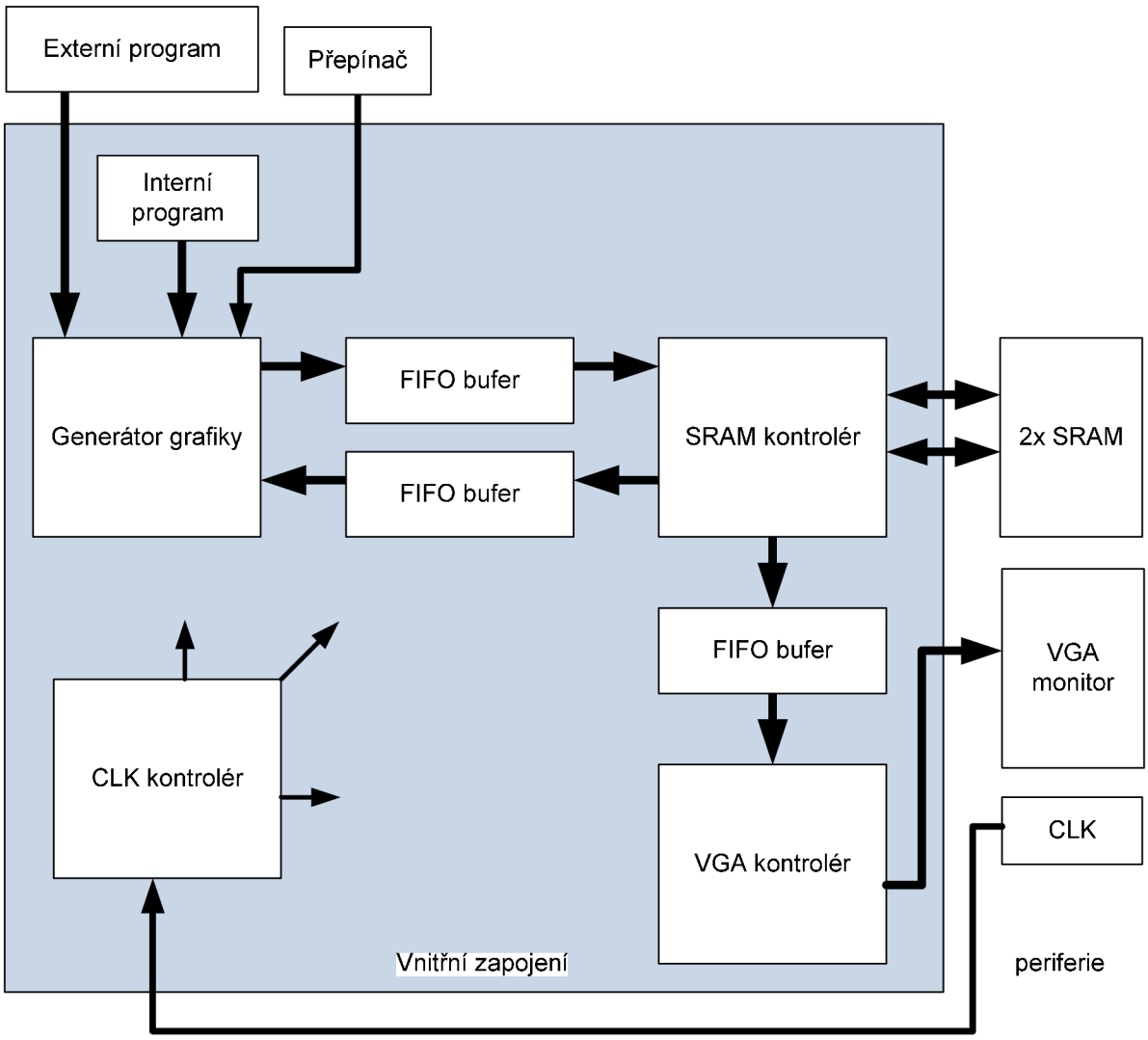
2 Vývojový kit

Pro účely této práce se jako nejvhodnější ukázalo použít vývojový kit od společnosti DIGILENT s označením SPARTAN-3 Starter Board. Tato vývojová deska je osazena čipem SPARTAN-3 XC3S200 a dvěma čipy SRAM pamětí ISSI IS61LV25616AL, které mají kapacitu 1MB. Na desce je k dispozici jak VGA výstup, který je ovšem 1-bitový pro každou z barev, tak i rozšiřovací konektory, jejichž prostřednictvím lze vlastnosti VGA výstupu výrazně zlepšit a připojit k desce vnější zdroje signálu – v našem případě řídící signály pro ovládání grafického rozhraní. Podrobné parametry vývojového kitu a jeho další možnosti jsou dostupné v příslušné literatuře [1].

Vzhledem k tomu, že tato práce je zaměřena na programové vybavení, nikoliv vlastní implementaci do systému, zvolil jsem použití již hotového vývojového kitu. Jak již bylo napsáno v úvodu, implementace do jiné sestavy je poměrně jednoduchá. Použití vývojového kitu je vhodné také vzhledem k ceně kitu a integraci všech potřebných periferií s výjimkou více bitového převodníku pro VGA.

3 Vnitřní zapojení

Na základě zadání a zvoleného obvodu spolu s vývojovou deskou jsem zvolil následující vnitřní zapojení jednotlivých funkčních bloků:



Obrázek 1 Vnitřní zapojení obvodu

Popis jednotlivých částí:

- Generátor grafiky – generuje na základě vnějších příkazů výsledný obraz, jeho výstup směrem k SRAM paměti, může mít požadavek na čtení informací o daném pixelu nebo zápis daného pixelu do paměti, rozlišuje pomocí externího přepínače zda jsou informace

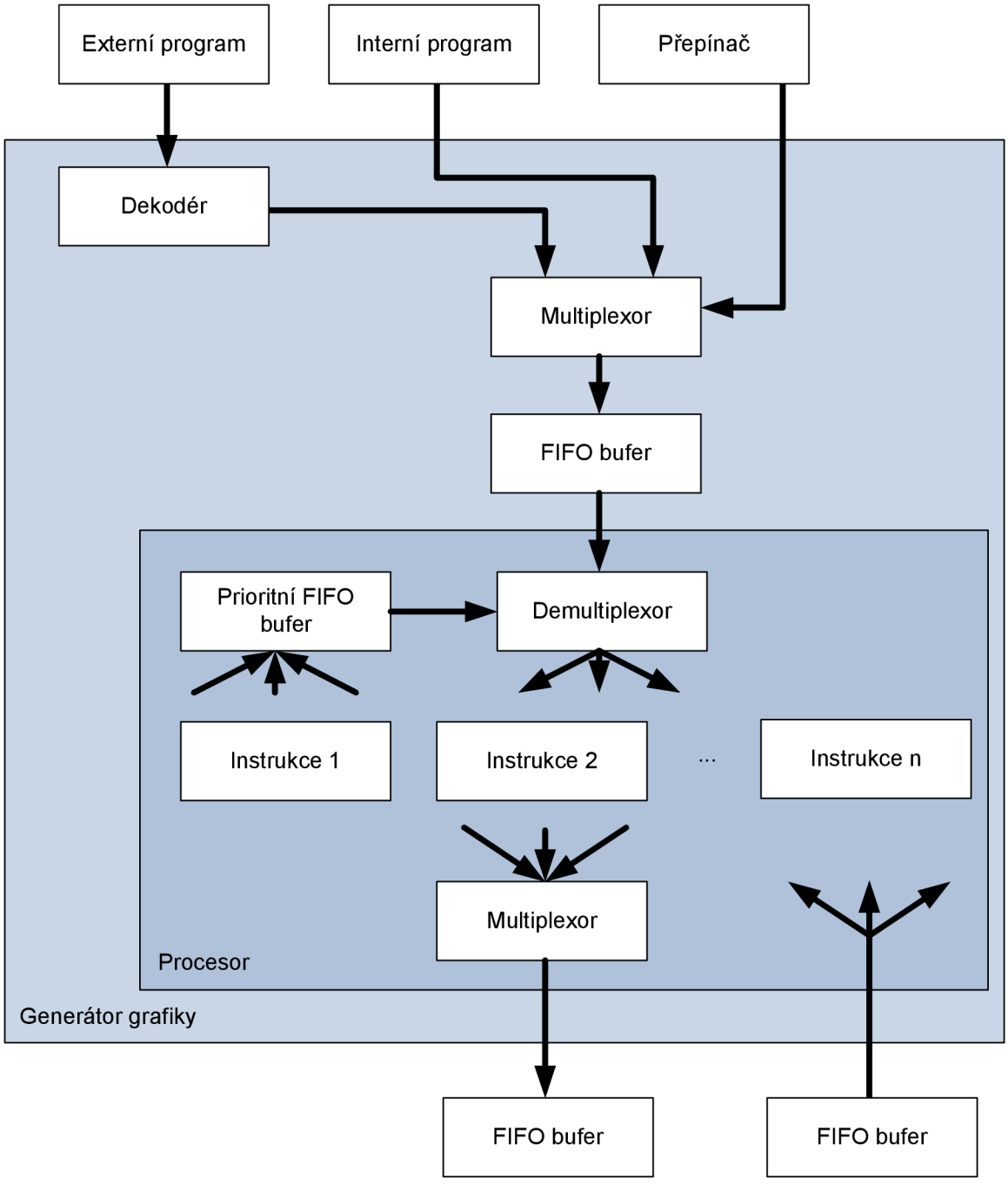
získávány z vnějšího nebo vnitřního programu, vnitřní zapojení tohoto bloku je na obrázku 2

- FIFO bufery mezi Generátorem grafiky a SRAM kontrolérem – mají za úlohu vyrovnávat komunikaci mezi oběma bloky, jejich zapojení do obvodu řeší rozdílné frekvence hodinového signálu u obou bloků a tím i situace, kdy z paměti odchází větší množství dat než je schopen generátor zpracovat a na druhou stranu situace, kdy probíhá čtení dat z SRAM pro zobrazení na VGA rozhraní
- SRAM kontrolér – komunikuje s SRAM pamětmi, emuluje dvou portovou VRAM, která je požadovaná ve specifikaci VGA, rozlišuje několik stupňů priorit požadovaného přístupu k pamětem
- FIFO bufer mezi SRAM kontrolérem a VGA kontrolérem – zajišťuje dostatečný počet následujících pixelů pro zpracování VGA kontrolérem, opět pracuje s dvěma hodinovými signály
- VGA kontrolérem – postupně zpracovává pixely a zároveň generuje synchronizační signály pro správné zobrazení na VGA monitoru
- CLK kontrolérem – je zařazen pro přehlednější administraci jednotlivých kmitočtů – přístupu do paměti, VGA pixel clocku a globálního vnitřního clocku

Na obrázku 2 je zakresleno navržené vnitřní zapojení bloku Generátor grafiky.

- Externí vstupy a jejich zpracování – Do bloku vstupují instrukce jak od externího, tak i interního programu, informace z externího programu se musí převést ze sériově-paralelní na paralelní informaci. K tomuto účelu je zařazen blok Dekodér.
- Multiplexor – na základě informací z Přepínače rozhoduje, který zdroj instrukcí (externí resp. interní program) zapisuje informace do FIFO buferu, který zpracovává instrukční frontu
- Procesor – jednotka zpracovávající příchozí instrukce, druhý vstup je FIFO bufer pro vstup vyžádaných dat z paměti, který je využíván jen některými instrukcemi, výstupem jsou pak informace o barvě a adrese daného pixelu v paměti
    - Demultiplexor – určuje, která instrukce se bude vykonávat, upřednostňuje instrukce z Prioritního FIFO buferu, aktivuje příslušný instrukční modul, užití Prioritního FIFO buferu je nezbytné u některých funkcí

- Instrukce – tyto bloky fungují jako zásuvné moduly, každá z instrukcí vykonává příslušnou funkci – např. vykreslení čáry, pixelu atd.
- Multiplexor – umožňuje výstup dat z jednotlivých instrukčních bloků do FIFO buferu mezi Generátorem grafiky a SRAM kontrolérem



Obrázek 2 Vniřní zapojení bloku Generátor grafiky

4 VGA

VGA rozhraním označujeme dnes rozhraní počítač-monitor. Název byl zaveden firmou IBM a označuje Video graphics array. IBM tento standard zavedlo v roce 1987 a nahrazoval starší protokoly EGA (Enhanced graphics adapter z roku 1984) a CGA (Color graphics adapter z roku 1981). V základní nejstarší verzi je možné dosáhnout maximálního rozlišení 720x480 pixelů, maximální obnovovací frekvence 70Hz a 256ti barev. Základem je dvou portová 256kB Video RAM. Nejčastějším využitím původního VGA byl textový režim v rozlišení 80x40 pixelů.

Od další verze protokolu SVGA (Super VGA) je již protokol vyvíjen asociací VESA (Video Electronics Standards Associations). Jedná se o konsorcium výrobců a vývojářů grafických karet a monitorů. Plné znění dokumentace VESA je dostupné pouze za poplatek. V současnosti je standardizováno minimálně 7 řad obrazových poměrů. Nejpoužívanější jsou 4:3, 16:9 a 16:10.

4.1 Princip

Princip VGA protokolu lze nejlépe vysvětlit na principu klasického CRT monitoru. U CRT monitoru je obraz vytvářen amplitudově modulovaným elektronovým paprskem pohybujícím se po stínítku zleva doprava a odshora dolů. Při horizontálním vychylování paprsek svým pohybem zleva doprava vykresluje jednotlivé body v přímých liniích jednotlivých řádků. Tento děj je označován jako aktivní řádkový běh. Jakmile paprsek dospěje za pravý obraz obrazu, vrací se zpět na začátek řádku. Tento děj se označuje jako zpětný řádkový běh. Při zpětném řádkovém běhu se již nezobrazuje žádná informace. Paprsek je zároveň i vertikálně vychylován. Toto vychylování je ale mnohem menší než horizontální. Důsledkem tohoto vychylování je, že řádky jsou ve skutečnosti mírně nakloněny. Vrací-li se paprsek zpět nahoru, jedná se o aktivní snímkový běh.

Výše popsaným způsobem vytvořený obraz je tedy ve skutečnosti obraz časově rozvinutý v body. Barva každého jednotlivého bodu je pak daná třemi signály označovanými dle jim příslušných barev – RGB (Red Green Blue). Signály RGB jsou při zpětných bězích neaktivní a mají v době aktivity těchto dějů úroveň, při které se bod na obrazovce nevykreslí – resp. nevysvítí. Kvalitního obrazu lze dosáhnout pouze správným nastavením časování u všech signálů. U takto vykresleného obrazu jsou přesně dány pozice jednotlivých bodů.

4.2 Signály

4.2.1 Vertikální synchronizace

Signál vertikální synchronizace (dále označovaný také jako VSYNC) určuje frekvenci, s jakou jsou obnovovány snímky (snímková frekvence). Frekvence tohoto signálu tedy určuje dobu, za kterou je celý jeden snímek vykreslen. Náběžná (v případě pozitivního synchronizačního impulzu) hrana tohoto signálu označuje začátek nového snímku. Existuje značné množství variant možných rozlišení a snímkových frekvencí.

V současnosti se běžně pohybuje snímková frekvence v rozsahu 56 až 120Hz. Optimální snímková frekvence je minimálně 72Hz – od této frekvence lze bezpečně tvrdit, že vykreslování je nerozeznatelné lidským okem a není vidět blikání obrazu. Na tento vjem má vliv ovšem mnohem více faktorů – např. doba zatemnění, plocha obrazu nebo jas.

Je možné vykreslovat celý snímek nebo půlsnímek – toto záleží na tom, zda se jedná o neprokládané (standardně) nebo prokládané řádkování.

4.2.2 Horizontální synchronizace

Signál horizontální synchronizace (dále označován také jako HSYNC) určuje frekvenci, se kterou jsou vykreslovány jednotlivé řádky. Tento signál má podobná pravidla jako VSYNC. Synchronizační signál určuje začátek vykreslovaného řádku. Jeho frekvence je o několik řádů vyšší než v případě VSYNC a pohybuje se v rozsahu 31,5 až 100kHz.

4.2.3 RGB signály

Každá barvu je možné v případě VGA vyjádřit kombinací signálů RGB (Red Green Blue) – výsledná barva je dána jejich optickým součtem. Jedná se o tzv. aditivní míchání barev, kdy při současném dopadu paprsků vzniká nová barva. Intenzita barevného vjemu se sčítá a jas se zvětšuje. Při plné intenzitě všech složek RGB vznikne bílá barva.

Základní možnosti míchání barev jsou uvedeny v následující tabulce:

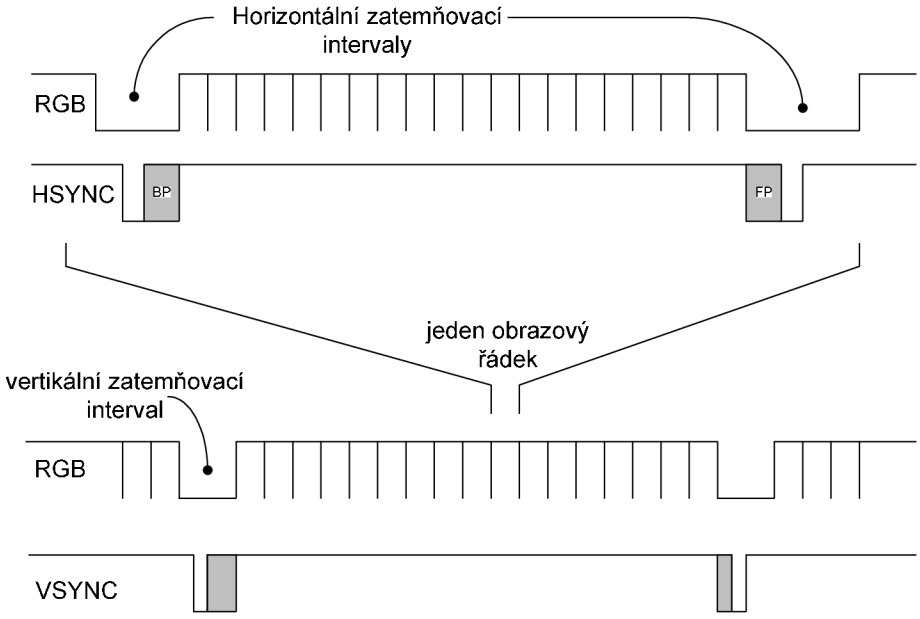
červená (red)	zelená (green)	modrá (blue)	Výsledná barva
0	0	0	černá
0	0	1	modrá
0	1	0	zelená
0	1	1	azurová
1	0	0	červená
1	0	1	purpurová
1	1	0	žlutá
1	1	1	bílá

Tabulka 1 Základní možnosti míchání barev

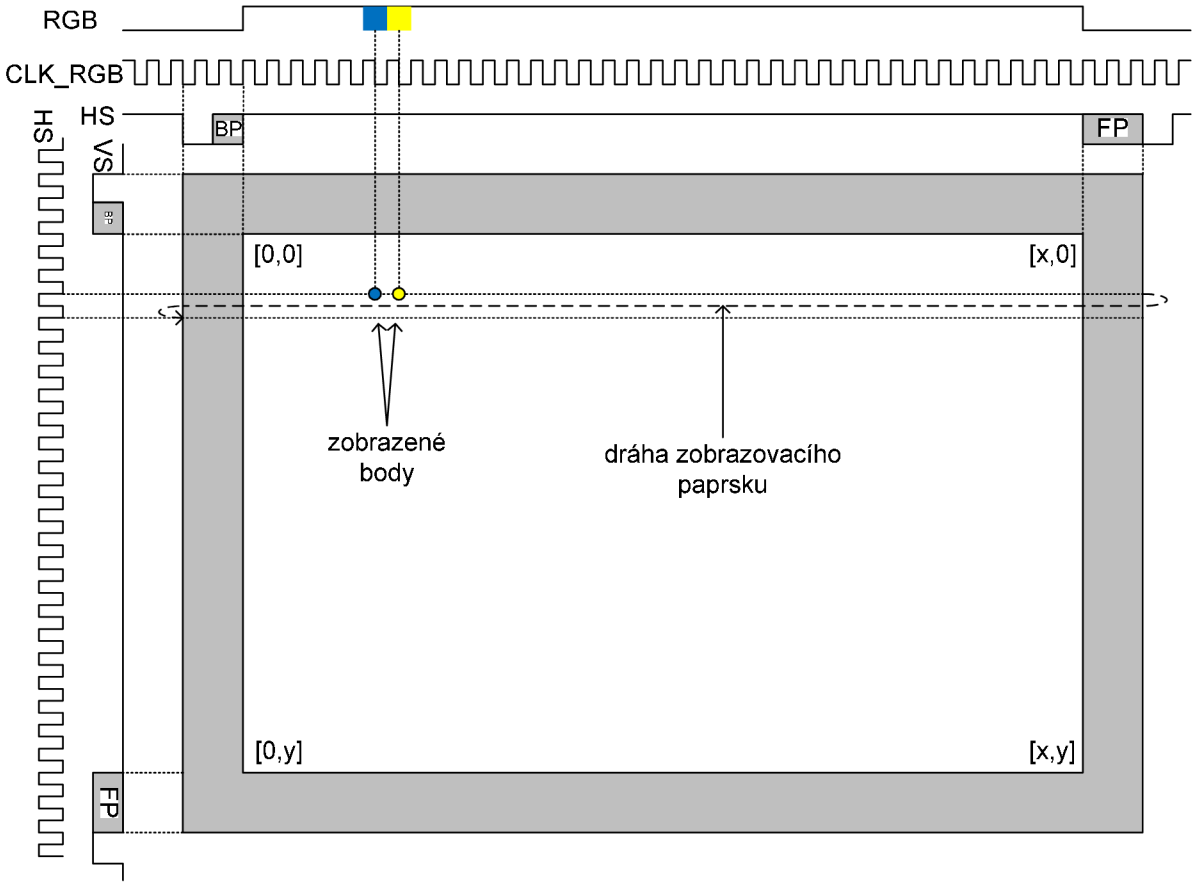
RGB signály přenáší obrazová data pro každý zobrazovaný bod zvlášť a jsou analogové. Frekvence s jakou se jednotlivé body vykreslují se pohybuje od 25 do 315MHz. Signály RGB umožňují přenášet informaci o synchronizaci, tato možnost se ovšem využívá zřídka a vůbec není podporována ze strany klasických adaptérů v PC.

4.3 Vykreslování obrazu a časování signálů

Proces vykreslování obrazu je ilustrován na obrázku 3. Je nutné si představit vykreslování obrazu jako nepřetržitý tok dat a jako děj, který je sériový – neděje se tedy, jak již bylo výše zmíněno, najednou pro celou obrazovku.



Obrázek 3 Proces vykreslování obrazu



Obrázek 4 Tvorba celého obrazu

Na obrázku 4 je situace rozkreslena tak, aby byl mnohem zřetelněji vidět vliv dílčích signálů podílejících se na tvorbě výsledného obrazu. Na obrázku 4 je sice vidět signál CLK_RGB, tento signál je nicméně pouze pomocný pro vytváření a správné časování obrazu a je pouze vnitřní. I přes to se ovšem jedná o hlavní řídící signál, který určuje frekvenci ostatních řídících signálů (VSYNC, HSYNC).

4.3.1 Oblasti zobrazování

Na obrázku 4 je aktivní oblastí bílá plocha ohraničená souřadnicemi. Ty jsou uvedeny z důvodu vysoké variability pouze obecně, lze je ovšem chápat např. jako [639,479]. Šedá plocha je pak plocha zatemnění, ve které je vykreslování neaktivní.

Dobu zatemnění určuje součet FP (Front porch – viz níže), BP (Back porch – viz níže) a synchronizačního intervalu. Z obrázku 4 zároveň plyne, že doba zatemnění bezprostředně odděluje jednotlivé aktivní oblasti v horizontálním směru. Když dojde paprsek na konec obrazovky, nastane zároveň doba zatemnění i pro vertikální směr a paprsek se rychle vrací

z pravého dolního rohu do levého horního rohu. Při době zatemnění jsou signály RGB nulové a nezobrazuje se žádná informace ve stopě paprsku.

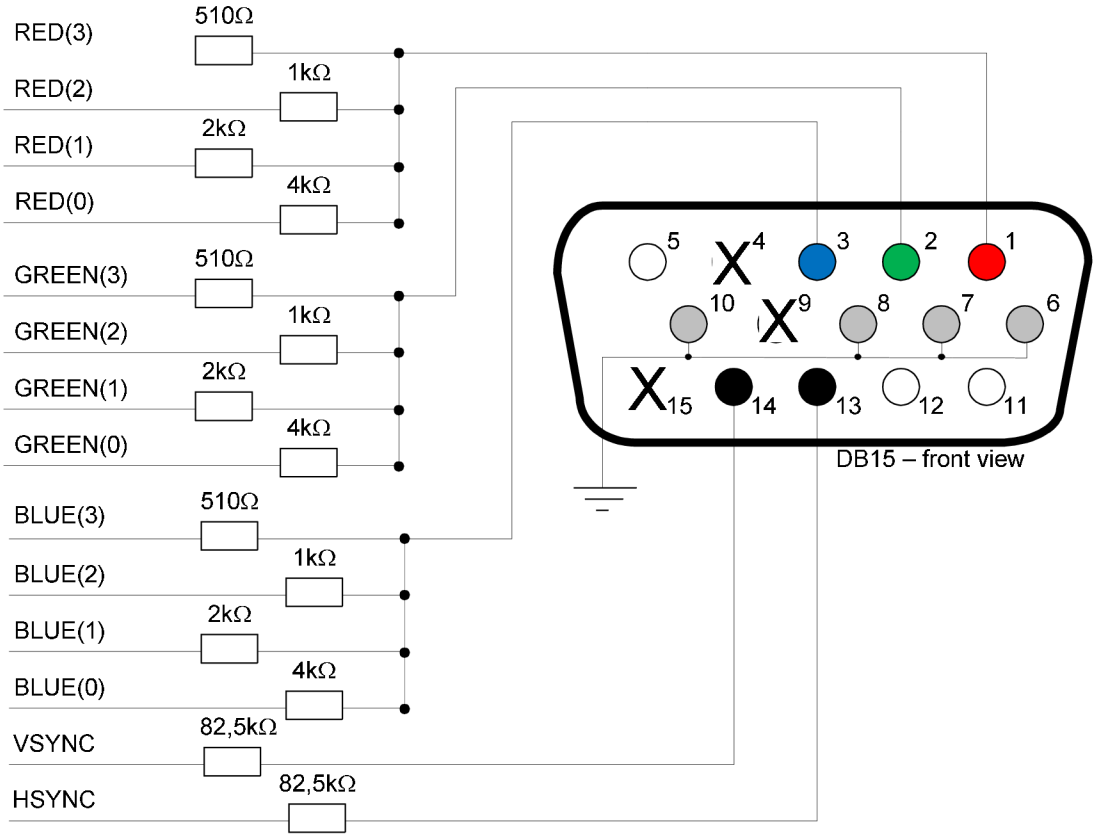
Jako front porch (FP) označujeme časový interval mezi koncem vykreslování obrazu a začátkem synchronizačního impulsu. Jako back porch (BP) pak označujeme časový interval mezi koncem synchronizačního impulsu a začátkem vykreslování obrazu. Back porch bývá zpravidla mnohem delší než front porch.

Přesné hodnoty časovacích signálů a hodnoty orientace synchronizačních signálů lze nalézt v příloze 1.

4.3.2 D/A převodník a zapojení VGA konektoru

Obraz budeme připravovat v digitální formě, komunikace v rámci VGA rozhraní je ovšem analogová. Z tohoto důvodu je nutné na výstup FPGA čipu začlenit D/A převodník. Pro zpracování VGA signálu lze použít i jednoduchý rezistorový převodník. Na obrázku 19 je převodník použitý na DPS XILINX Spartan-3A Starter Kitu. Více viz [4]. Obdobný převodník byl použit pro testování výstupu a byl připojen na jeden z rozšiřujících konektorů.

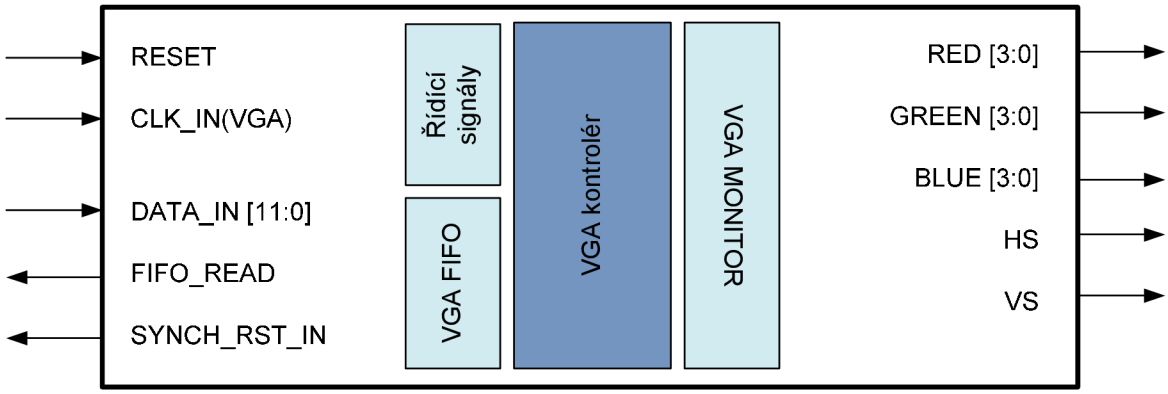
Na obrázku 7 je vyobrazen kromě jednoduchého rezistorového DA převodníku i VGA konektor DB-15 v pohledu zepředu. Piny 5, 11 a 12 jsou standardně uzemněny, případně nejsou zapojeny. V některých speciálních případech je lze použít pro DDC synchronizaci. Piny 6,7 a 8 jsou použity pro zemnění signálů na pinech 1, 2 a 3 (R, G a B).



Obrázek 5 Zapojení 4 bitového. rezistorového převodníku pro VGA

4.3.3 Zapojení VGA kontroléru do obvodu

Funkcí VGA kontroléru je pouze správné načasování signálů sloužících k synchronizaci – ať už vnější, která je daná VGA specifikací, nebo vnitřní – synchronní reset. Z toho důvodu lze blok připojit do obvodu velice jednoduše:



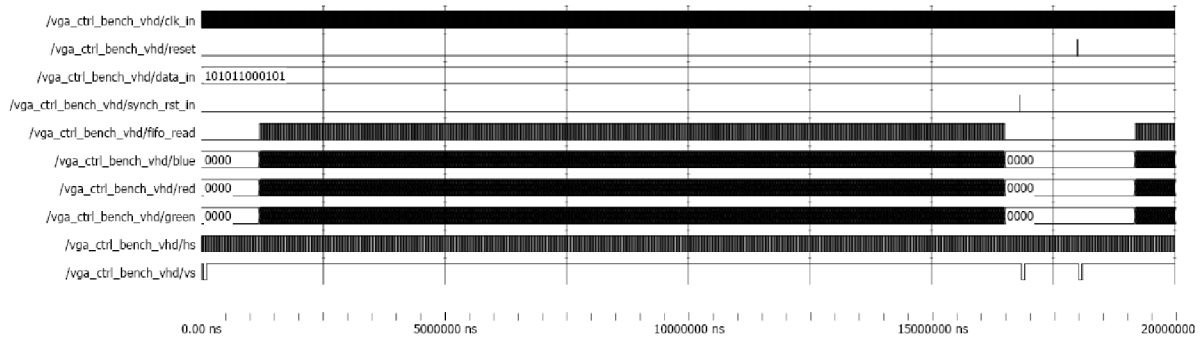
Obrázek 6 Zapojení VGA kontroléru do obvodu

4.4 Ověření vlastností programu

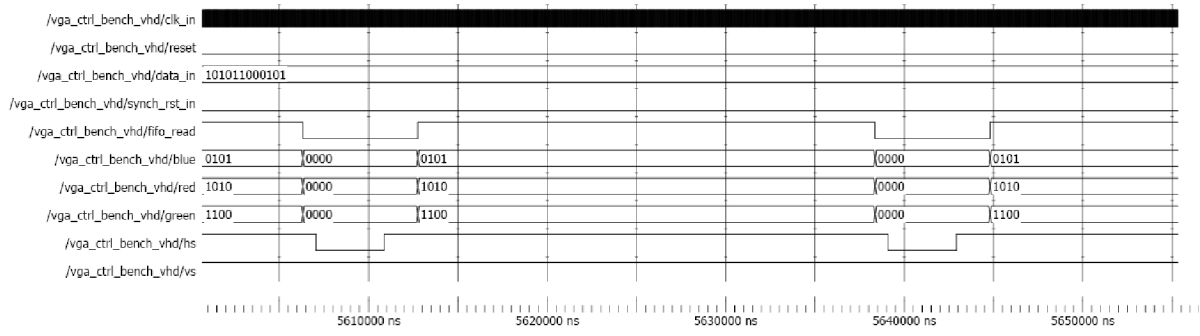
Funkce programu VGA kontroléru byla ověřena časovou simulací. Vlastní program VGA kontroléru je k dispozici spolu se zdrojovým kódem testovacího programu na CD příloze.

Celý průběh pokrývá průběh vykreslování celé jedné stránky při rozlišení 640 x 480 pixelů (obrazových bodů). Na konci stránky je vidět synchronní reset následovaný vertikální synchronizací VS. Ke konci simulace je ještě proveden globální reset a proto jsou v rychlém sledu za sebou dva vertikální synchronizační impulzy.

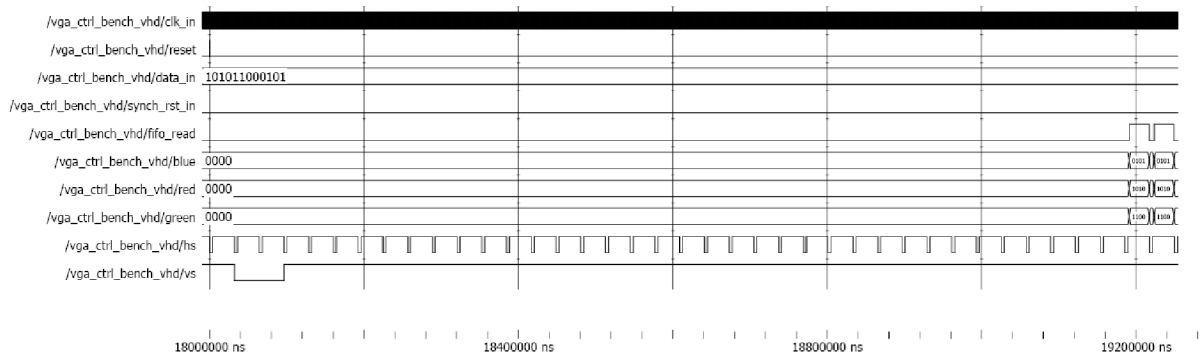
Funkce VGA kontroléru byla prakticky ověřena s reálným VGA monitorem.



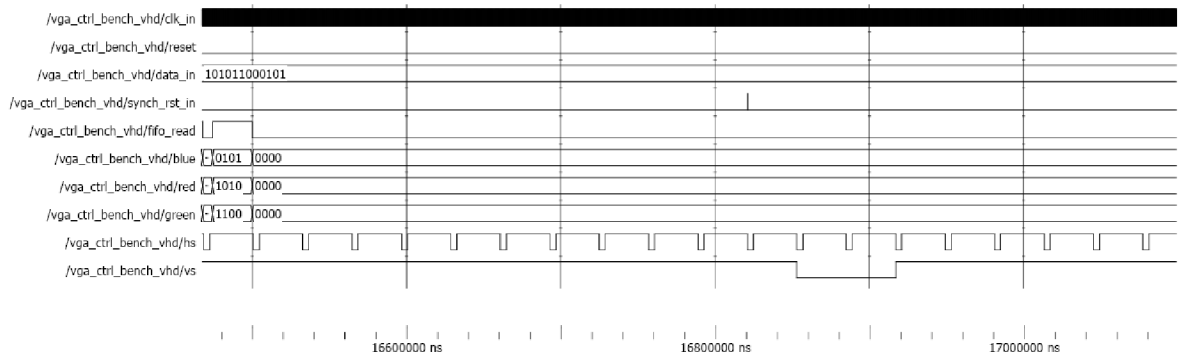
Obrázek 7 Kompletní časový průběh testovacího programu



Obrázek 8 Detailní pohled na okolí horizontální synchronizace HS



Obrázek 9 Detailní pohled na okolí vertikální synchronizace VS



Obrázek 10 Detailní pohled na synchronní reset SYNCH_RST_IN

5 SRAM kontrolér

5.1 Použitý obvod

V použitém Spartan-3 Starter kitu jsou zapojeny 2 čipy s typovým označením IS61LV25616AL-10T. Jedná se o asynchronní SRAM paměti v SMD pouzdru. Jejich kapacita je 256k x 16b, celkem tedy 512kB paměti v každém čipu. Maximální možná frekvence podporovaná tímto obvodem je 100MHz, tzn 10 ns. Čipy jsou jednoportové, jejich použití jako VideoRAM docílíme emulací dvouportové paměti a prioritním přístupem k datům.

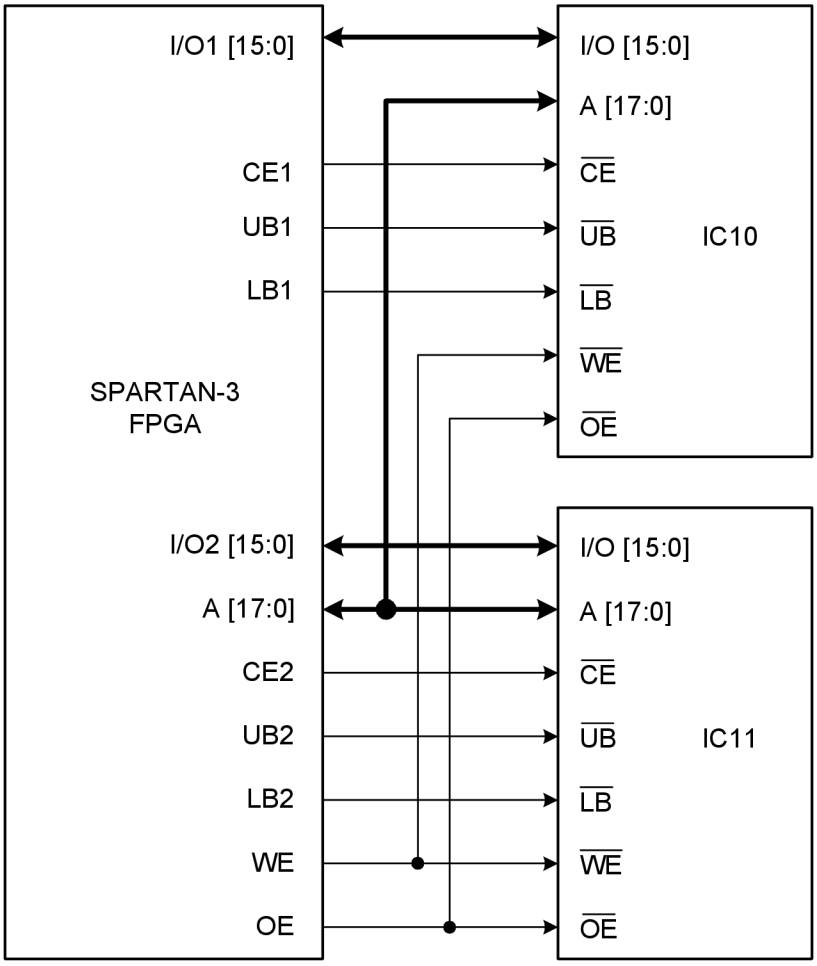
K čipům je možno přistupovat individuálně, mají však společnou adresaci, což znemožňuje dvouportovou emulaci duplikací zápisů a přístup po jednotlivých pixelech. Emulace dvouportové paměti tímto způsobem by byla docílena tak, že obě paměti by nesly stejnou informaci a zápis do jednoho a čtení z druhého čipu by mohly být provedeny najednou, přičemž by se informace musely kopírovat navzájem mezi pamětmi v dobách, kdy není požadavek na čtení ze strany VGA modulu.

Jelikož je uvažováno s čtyř bitovou hloubkou u každé z barev, je nutno k popisu každého z pixelů uložit 12 bitů. Jelikož jsou paměťové čipy 16ti bitové, byly zvažovány různé možnosti zápisu do paměti. Jako nejefektivnější se ukázalo ponechat 4 bity u každého čipu a na každé použité adrese neobsazené. Toto řešení umožňuje rozšíření hloubky barevnosti na 5 bitů u každé z barev, 16tý bit může být využit např. k verifikaci zápisu.

Z výše uvedeného plyne jeden zásadní fakt – pixely jsou na řádcích přístupné v párech a z tohoto důvodu je nutné validovat zápis u jednotlivých pixelů. Problémy a zpomalení to způsobí především u zápisu prostých vertikálních čar a všude tam, kde zapisujeme do paměti nebo z ní čteme po pixelech. Tento přístup je ovšem nutný z principiálních důvodů a nelze bez něj dosáhnout rozlišení 640x480 obrazových bodů při 4 bitové barevné hloubce na použité vývojové desce.

5.1.1 Reálné zapojení pamětí na desce

Na obrázku 11 je zakresleno zapojení jednotlivých signálů tak, jak je realizováno na desce SPARTAN-3 starter kitu. Z tohoto zapojení je samozřejmě nezbytné vycházet při vývoji programu. Zapojení jednotlivých pinů cílového obvodu SPARTAN-3 je součástí příloh na CD a vychází ze SPARTAN-3 FPGA Starter Kit Board User Guide – viz. použitá literatura [1].



Obrázek 11 Zapojení části vývodů FPGA a SRAM čipů

5.2 Adresace paměti

Celkem lze uložit 2.2^{18} pixelů (tj. 2 čipy o 18ti bitové adrese, vždy s informací o jednom pixelu), celkem tedy 2 . 262 144 pixelů = 524 288 pixelů. Maximální možný počet pixelů při standardním rozlišení je tedy 800 x 600 pixelů, při kterém je potřeba uchovat 480 000 pixelů. Jak bude ukázáno dále, adresace rozlišuje zobrazované body a okolní „rám“, který slouží pro

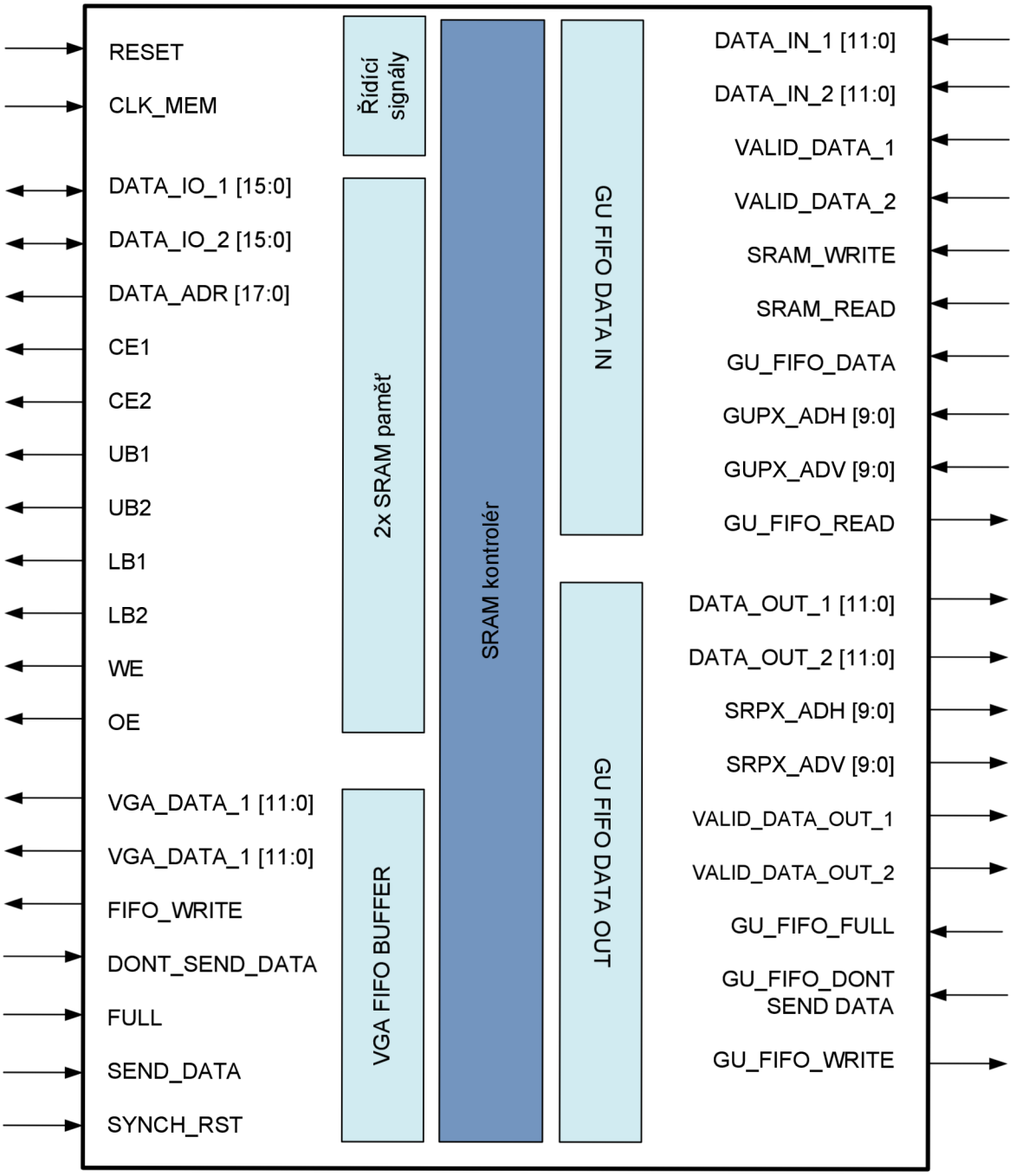
podporu předpokládaných transformací (posunutí atp.). Použité zobrazované rozlišení je 640 x 480 pixelů – což je viditelné okno, které je umístěno uprostřed celého obrazu vč. orámování. Z uvedeného vyplývá, že lze, pokud bude třeba přidat do SRAM kontroléru i část, která bude ukládat uživatelská data do zbylých 86,5kB paměti, která není využitá. Další možností je rozšíření plochy na maximum – tj. např. 840 x 624 bodů a velmi tak omezit „rám“ a možné operace v něm při současném zvýšení zobrazované plochy na 800 x 600 pixelů, což je maximum dosažitelné na tomto vývojovém kitu při barevné hloubce 4 bity.

Výpočet adresy dvojice pixelů je následující:

Vertikální adresa pixelu (např. 352tý řádek) * celkový počet pixelů v horizontálním směru (800 pixelů) + horizontální adresa pixelu (např. 250, což znamená pixely 500 a 501 – vysvětlení viz. výše). Výsledná adresa v dekadické soustavě pro bod(y) [250;352] je tedy 281850.

5.3 Vývodové zapojení bloku SRAM kontroléru do obvodu

Jak již bylo uvedeno v úvodu kapitoly, blok SRAM kontroléru komunikuje s paměťovými čipy SRAM a se třemi FIFO bufery – vstupem a výstupem z generátoru grafiky a výstupem zobrazované grafiky do VGA kontroléru. S ohledem na tyto skutečnosti bylo navrženo zapojení bloku v obvodu, které je na obrázku 12.



Obrázek 12 Zapojení bloku SRAM kontroléru do obvodu

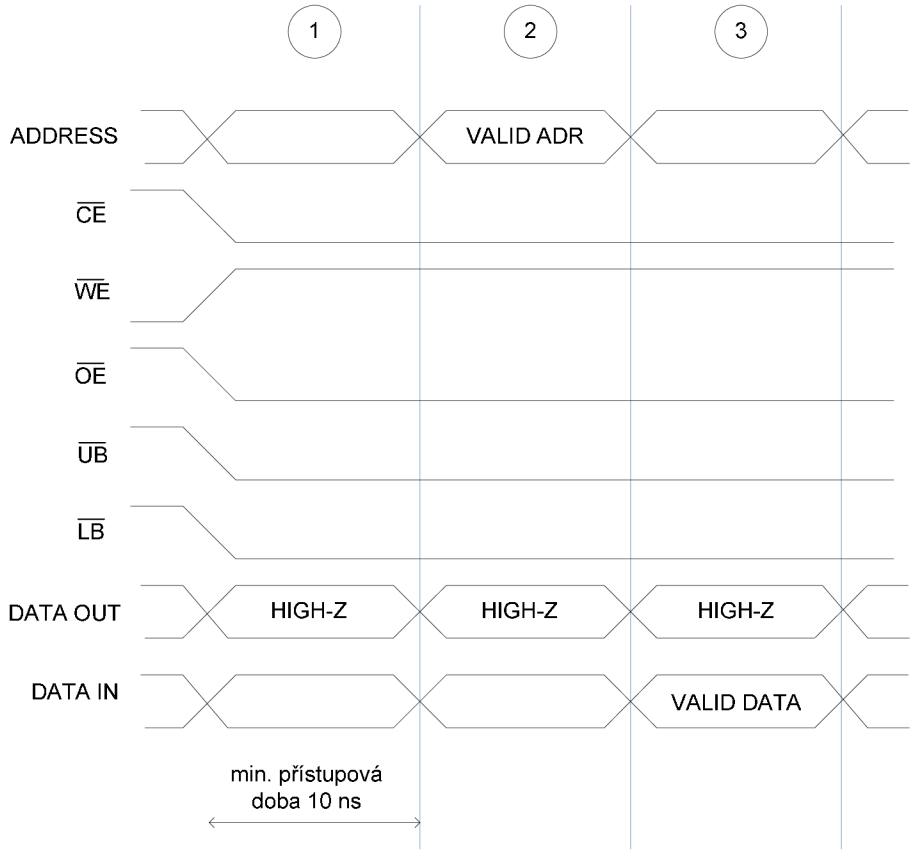
5.4 Protokol komunikace s SRAM pamětmi

SRAM kontrolér přistupuje k pamětem v níže naznačených cyklech. Tyto cykly jsou navrženy v souladu s technickým listem k příslušným pamětovým čipům (viz. použitá literatura [3]). Protokol komunikace je zjednodušen – plné časování je k dispozici v příslušené literatuře.

5.4.1 Cyklus čtení z paměti

Na obrázku 13 je znázorněn postup při čtení dat z paměti. Cyklus je samozřejmě platný pro použité paměti a je nutné jej revidovat v případech, kdy se použije jiný typ paměťových čipů.

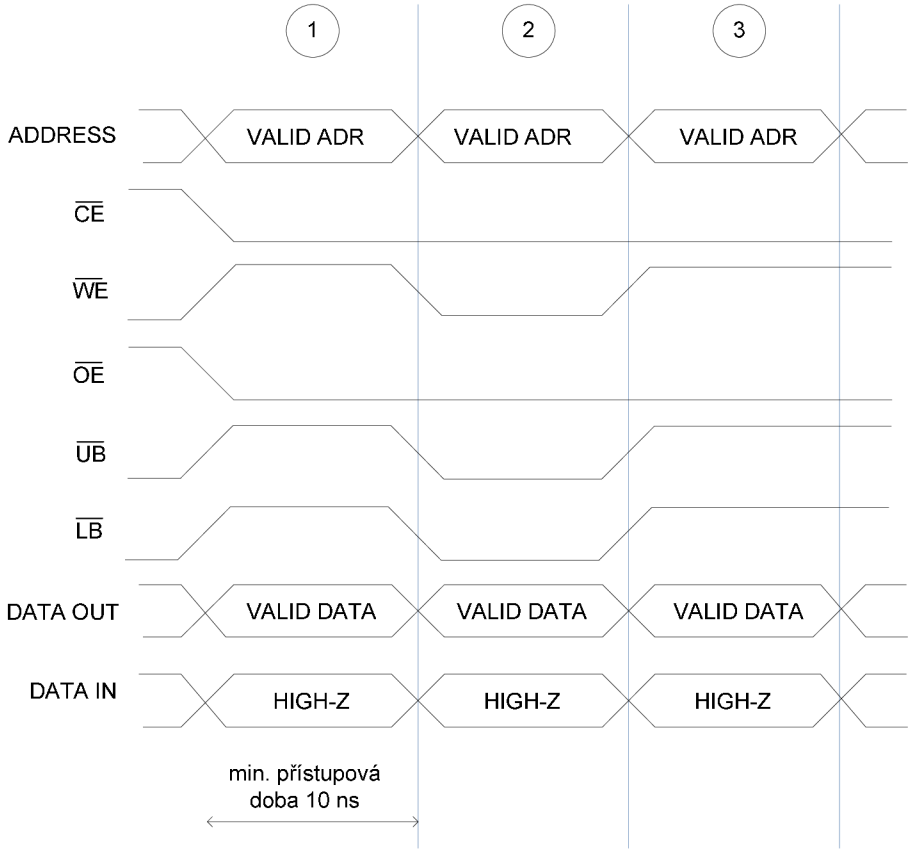
V případě, že v diagramu, který je zobrazen na obrázku 13, není uvedena hodnota, kterou očekáváme, resp. kterou máme na sběrnici odeslat, pak na této hodnotě nezáleží. Nutno podotknout, že signály DATA OUT a DATA IN používají v reálu stejný vstupně výstupní signál jak na čipu paměti, tak i vlastním SPARTAN-3 čipu. Při syntéze programu jsou k těmto signálů přiřazeny bloky IO BUF (IO bufery). V horní části obrázku 13 jsou označeny čísla cyklů (1 až 3).



Obrázek 13 Cyklus čtení z SRAM paměti

5.4.2 Cyklus zápisu do paměti

Pro popis cyklu zápisu do paměti platí stejné zásady jako v případě cyklu čtení.



Obrázek 14 Cyklus zápisu do SRAM paměti

5.5 Priority přístupu do paměti

S ohledem na to, že SRAM kontrolér musí emulovat dvou portový přístup do paměti z důvodů popsaných v kapitole 5.1, je nutné stanovit priority přístupu k paměti (od nejvyšší):

1) Globální reset
2) Požadavek na zaslání dat ze strany FIFO VGA kontroléru, variantně kombinovaný se synchronním resetem, který nuluje pozici požadovaného pixelu
3) Zápis z paměti nebo čtení ze strany FIFO generátoru grafiky
4) Čtení z paměti následující pixely pro FIFO VGA kontroléru v případech, kdy není naplněn. Tato možnost zefektivňuje přístup do paměti

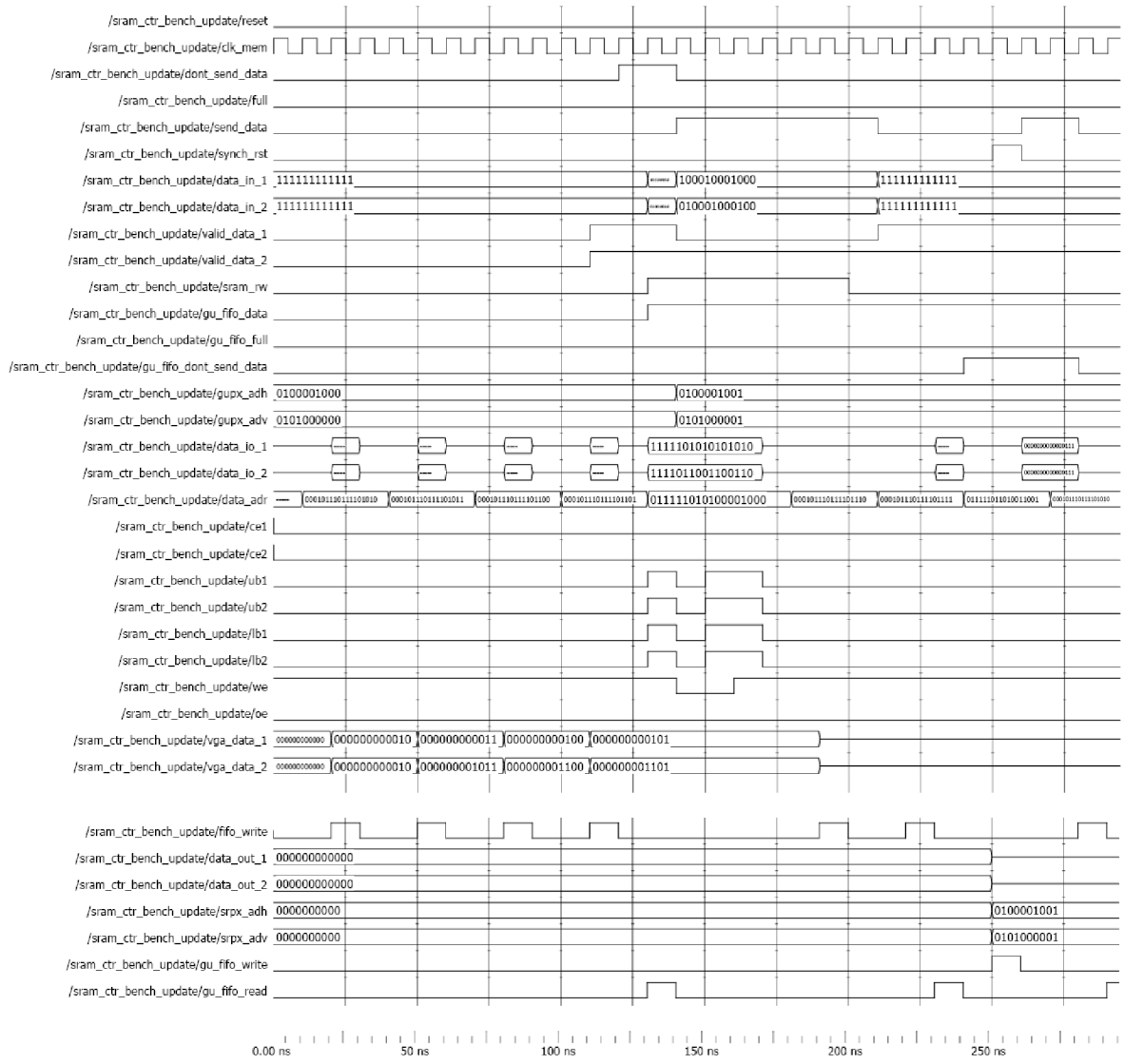
Emulace dvou portové paměti je založena na tom, že čtení i zápis do paměti je mnohem rychlejší než požadavek od FIFO VGA kontroléru i požadavky FIFO generátoru grafiky. V praxi rychlost čtení dat při rozlišení obrazovky 640 x 480 pixelů a 25MHz pixel clocku třikrát rychlejší než zápis na obrazovku. V tomto určení rychlosti navíc ještě nevstupuje faktor okraje obrazovky, což poměr rychlosti čtení z paměti vůči rychlosti výstupu na VGA monitor ještě zvětšuje. Dá se tedy říci, že zvolený přístup k emulaci dvou portové paměti je dostatečný.

5.6 Ověření vlastností programu

Program, který popisuje blok SRAM Controlleru je součástí přílohy na CD. Jako přílohu na CD naleznete i testovací program. Ten simuluje následující události:

- 0 ns – není plný buffer VGA FIFO, žádná jiná událost, tudíž se čtou data pro VGA FIFO
- 120 ns – plný buffer VGA FIFO, žádná jiná činnost, paměti nemají zadanou činnost
- 130 ns – zápis do paměti ze strany GU FIFO
- 140 ns – zápis přerušen požadavkem na dodání dat ze strany VGA FIFO
- 210 ns – pokračuje se se zápisem do paměti, část zápisu jde pouze do jedné z obou pamětí
- 240 ns – takřka plný zásobník GU FIFO, data stornována – tento stav je hypotetický
- 260 ns – synchronní reset ze strany VGA kontroléru, VGA FIFO je prázdný, doplňují se data

Výstup je zobrazen na následujícím obrázku 15:



Obrázek 15 Časový průběh testování bloku SRAM kontrolér

6 FIFO registry

Jako funkční bloky pro FIFO registry byla zvolena jádra generovaná pomocí nástroje IP CORE Generator. Tato utilita je implementovaná přímo do vývojového prostředí XILINX ISE.

Použití takto předdefinovaných bloků má několik výhod:

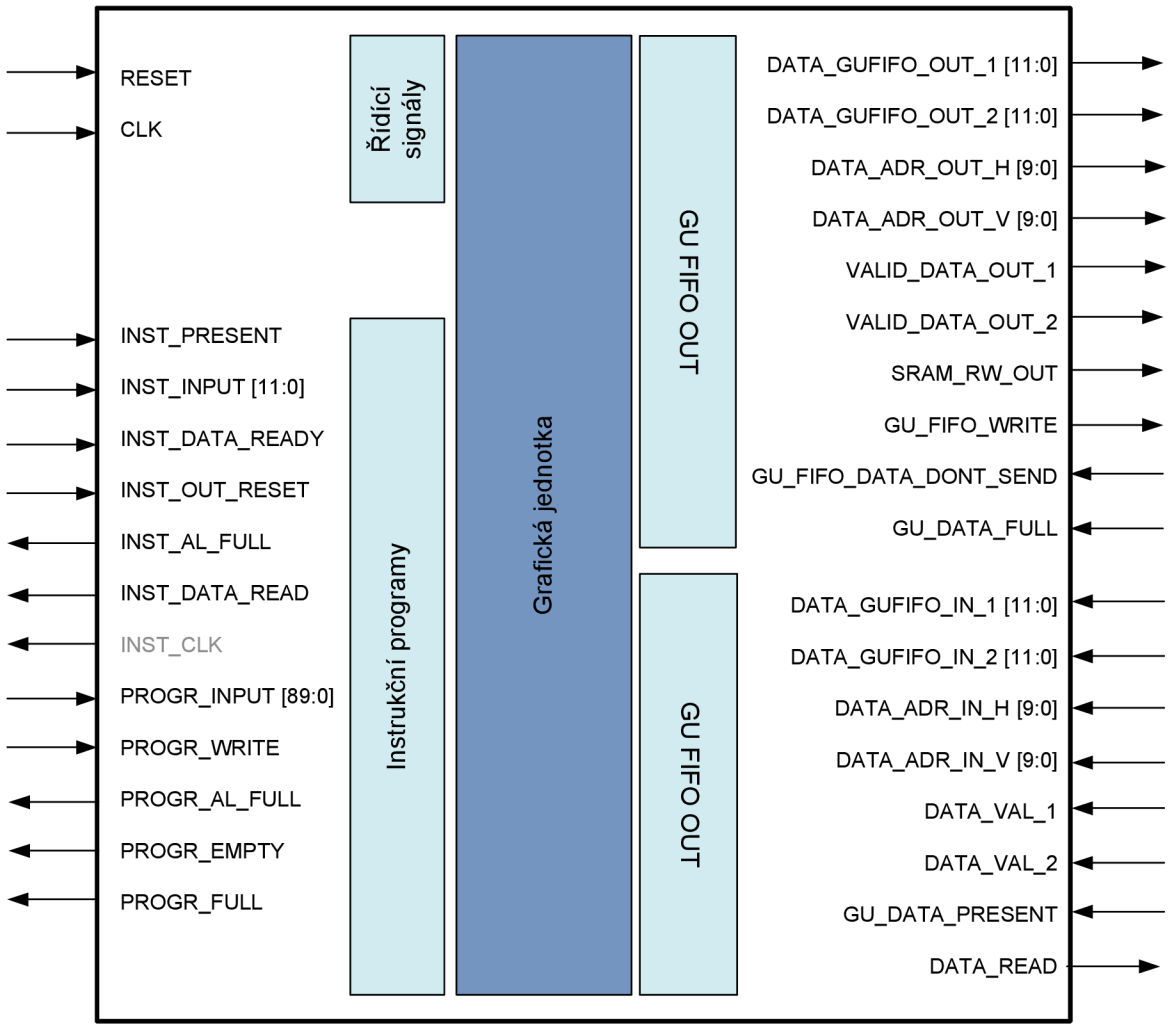
- jedná se o bloky, které jsou vyvinuty přímo výrobcem pro dané čipové řády
- z tohoto důvodu je rychlost nejvyšší, jakou můžeme dosáhnout, taktéž se tyto bloky vyznačují malým počtem potřebných hradel a bezproblémovou syntézou a překladem i pro FIFO s rozdílnými hodinovými signály pro čtení a zápis
- bloky jsou univerzální, lze je modifikovat pro většinu potřebných variant

Použití má i některé nevýhody:

- hlavní nevýhodou je to, že blok se chová jako black box – nelze tedy modifikovat v některých potřebných variantách
- z tohoto důvodu je velmi problematické provádět simulace bloků, jejichž součástí jsou podbloky FIFO – simulaci je možné provést pouze na úrovni post-translate a nižší, což je spojeno s horší kontrolou vnitřních signálů a větší nepřehledností

7 Grafická jednotka

Posledním ze základních bloků, které byly navrženy v kapitole 3 na obrázku 1 je blok grafické jednotky. Tento blok je odpovědný za komunikaci s externím, resp. interním programem, který zadává instrukce a na základě provedené instrukce zapisuje do FIFO bloku mezi grafickou jednotkou a SRAM kontrolérem, resp. z druhého použitého FIFO bloku čte. Informace, které zapisuje nebo čte, jsou vždy pro dva obrazové body na dané adrese a jsou označeny atributy validity. Adresa je pak v souřadnicích [x;y], vlastní adresa je vypočtena až v SRAM kontroléru. Toto řešení bylo zvoleno z důvodu vyšší kompatibility a přehlednějšího zpracování informací.



Obrázek 16 Zapojení grafické jednotky do obvodu

7.1 Zpracování příchozí instrukce

Jak bylo naznačeno na obrázku 2, Generátor grafiky je připraven pro zpracování instrukcí jak z vnějšího, tak i vnitřního zdroje. Vzhledem k tomu, že počet dostupných vývodů na použitém SPARTAN-3 Starter kitu je konečný a značně omezený, byl zvolen sériově-paralelní přenos informace. Je nezbytně nutné, aby externí zdroj používal stejný hodinový signál jako grafická jednotka. Pokud by přesto byla nutnost rozdílných kmitočtů, lze v rámci grafické jednotky vytvořit další FIFO bufer, který dokáže číst a zapisovat při rozdílných hodinových signálech.

7.1.1 Dekodér

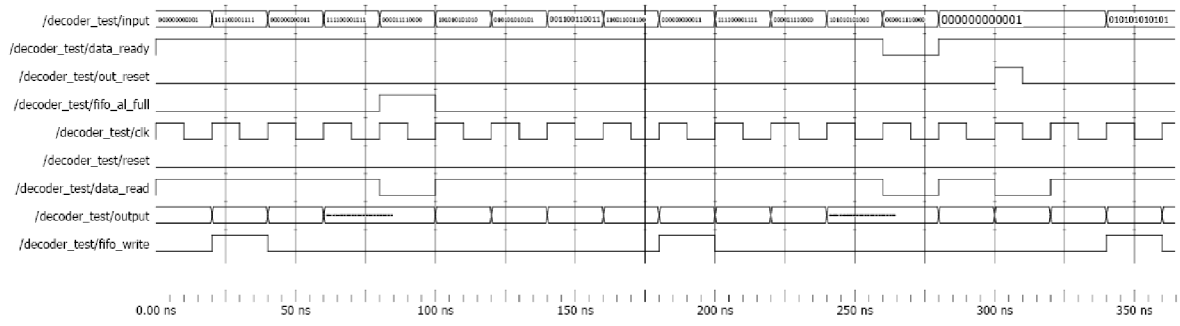
Dekodér je do bloku zařazen z důvodu postupného převodu 12ti bitové informace na informaci 90ti bitovou, kterou následně zpracovává procesor. Dekodér podle první informace rozezná, jak dlouhá instrukce je, následně očekává příslušný počet 12ti bitových dat.

Funkci nejlépe ilustruje průběh časové simulace na obrázku 17:

0-20ns V prvním kroku čte dekodér kód instrukce 001 – mazání obrazovky, následně očekává 1x 12bit. dat, při posledním, tj. druhém kroku zapisuje do FIFO buferu.

20-200ns Dekodér čte kód instrukce 003 – kreslení čáry, následně očekává 6x12bit. dat, v čase 80ns je příznak skoro zaplněného FIFO buferu. Tento příznak je v tomto čase hypotetický – v praxi by příznak mohl být aktivní ihned po posledním zápisu do buferu. Po překlopení příznaku zpátky na log. 0 pokračuje čtení datové informace, ukončeno zápisem do buferu v čase 180ns

180-360ns Opět čtení stejné instrukce, přerušeno v čase 260ns, kdy externí zdroj dat signalizuje nepřipravenost (DATA_READY v log. 0), následně v čase 300ns je aktivní vnější reset (OUT_RESET v log. 1), což způsobí vymazání dosavadního stavu a od času 340 ns následuje čtení instrukce 001.



Obrázek 17 Časový průběh simulace funkce bloku Dekodéru

7.1.2 Multiplikace vstupů a interní program

Interní program funguje pro lepší ilustraci na bázi jednoduchého stavového automatu. Každým čtením se tento stavový automat posouvá na hodnotu další instrukce. Pro lepší přehlednost lze rozdělit jinak nesnadno čitelnou 90ti bitovou informaci na několik 12ti bitových. Tyto 12ti bitové informace lze ještě zapsat jako převod čísla typu *integer* na *std_logic_vector*. V případě potřeby lze do tohoto bloku naprogramovat požadované funkce (např. vyhodnocování měření a jejich převod na instrukce).

Přepínání mezi interním a externím zdrojem instrukcí je v projektu přiloženém na CD ovládán tlačítkem SW7 na použité vývojové desce. Poloha HI znamená použití externího zdroje signálu. Přes multiplexor se zapisují informace do FIFO buferu pro instrukce.

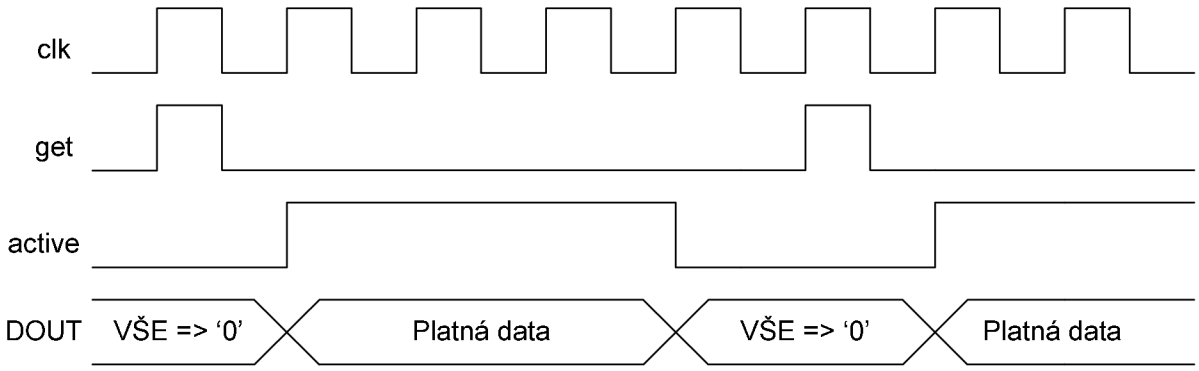
7.2 Procesor

Úloha bloku v Procesoru v bloku Generátoru grafiky je přijmout instrukci a vykonat ji. K tomuto účelu je v procesoru demultiplexor, který prioritně odebírá instrukce z vnitřního Prioritního FIFO buferu, sekundárně pak z vnějšího FIFO buferu instrukcí. Na základě kódu instrukce (bity 89-84) spustí daný blok instrukce.

Bloky instrukcí jsou do bloku procesoru zařazeny jako zásuvné moduly – každý z těchto bloků má jinou funkci, je relativně jednoduché ji do bloku procesoru začlenit a po začlenění lze velmi jednoduše vypnout. Instrukce má na vstupu spouštěcí příznak *GET* a data potřebná pro vykonání instrukce, na výstupu pak informace o daných obrazových bodech a aktivitě modulu. Volitelně lze připojit jako výstup prioritní FIFO bufer, např. pro instrukci, která má vykreslovat čtverec, budou na výstupu 4 informace o vykreslení čáry. Výrazně se takto dá zjednodušit a zpřehlednit vykreslování složitějších obrazců. Na vstup lze pak připojit výstup z FIFO buferu mezi SRAM kontrolérem a Generátorem grafiky v případech, kdy je nutné číst informace z paměti (např. pro instrukci vyplňování).

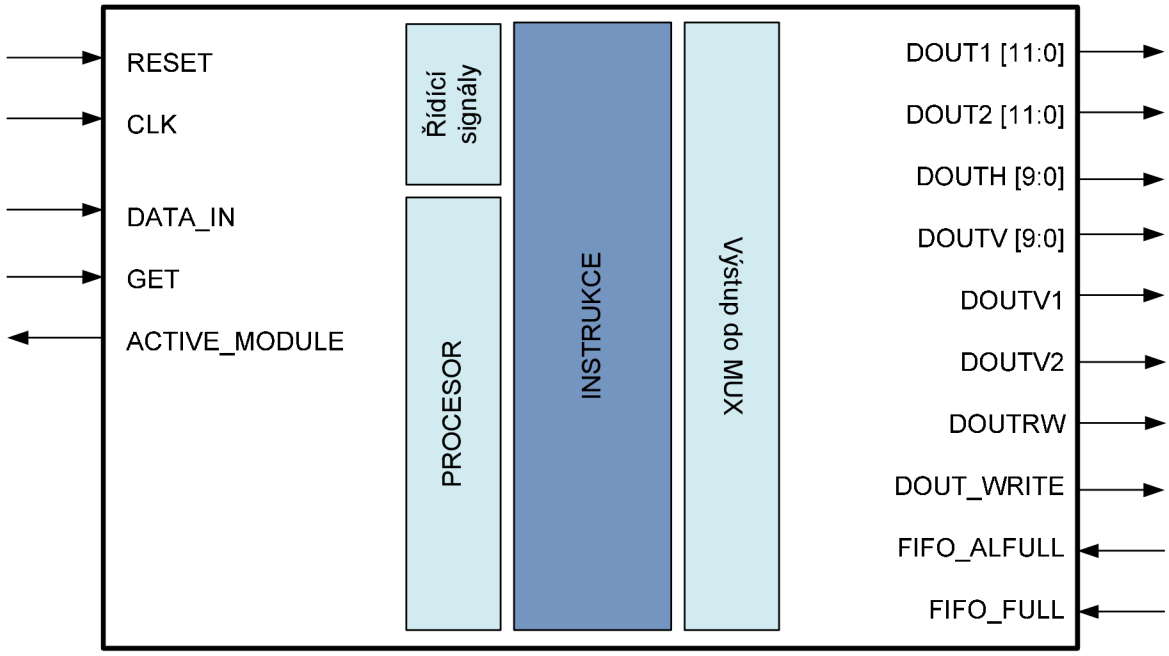
Na obrázku 18 je zobrazen obecný průběh vykonávání instrukce. Ze strany demultiplexoru dojde nejprve signál *GET*, následně se příznak *ACTIVE* změní na log. 1 a na výstupu jsou platná data. V případech, kdy je příznak *ACTIVE* v log.0 je nezbytné, aby byla všechna výstupní data v log.0. Multiplexor, který je zařazen za výstupy jednotlivých bloků instrukcí, je totiž založen na funkci OR a jiný než nulový výstup tedy mění výstup dané instrukce směrem k FIFO buferu (tj. změna adresy, příkazu čtení nebo zápisu do paměti atd.). V případě, že by byl multiplexor naproprogramován klasickým způsobem, kdy příznaky *ACTIVE* fungují jako řídící signály, by bylo použito neúměrné množství logických jednotek,

výsledek by byl ovšem stejný jako při dodržení pravidla nulového výstupu při log. 0 příznaku *ACTIVE*.



Obrázek 18 Časový průběh vykonávání obecného bloku Instrukce

Obecné zapojení instrukce do bloku procesoru je na obrázku 19.



Obrázek 19 Zapojení obecného bloku instrukce do bloku procesoru

8 Zobrazovací algoritmy

Rozlišujeme dva základní způsoby zobrazování: vektorové a rastrové. U vektorového typu jsou základními prvky úsečky, oblouky, kružnice atp. a z nich dále sestavujeme tzv. vyšší složené útvary jako např. obdélníky, znaky atd.

Rastrový způsob zobrazování naproti tomu pracuje s pixely (zkratka picture element), což jsou elementární obrazové prvky, ze kterých se sestavuje výsledný obraz na displeji nebo v tiskárně. Pixelu můžeme definovat určité atributy jako barvu a jas. Algoritmy popsané dále určují vlastnosti jednotlivých obrazových bodů a jejich výstupem jsou pak atributy pixelu na konkrétních souřadnicích.

Grafická zařízení mohou zobrazovat pouze s určitou rozlišovací schopností – v rastru (mřížce). Při kreslení hladké (ideální) křivky (v některých případech i úsečky) je tedy nutno dopočítat, u kterých pixelů bude nejvýhodnější změnit jejich atributy tak, aby výsledek byl co možná nejpodobnější požadované hladké křivce. Tuto činnost označujeme jako interpolaci. Tou je tedy stanovení polohy a hodnoty atributů pixelů nejlépe nahrazujících ideální křivku.

Je nutné, aby interpolace splňovala následující podmínky:

- výsledná křivka musí být z hlediska vizuálního vjemu hladká
- kresba musí začínat a končit v zadaných bodech rastru
- intenzita čáry nesmí záviset na délce a sklonu (alespoň ne do té míry, ve které má vliv na vizuální vjem)
- výpočet umístění v rastru musí být co možná nejrychlejší

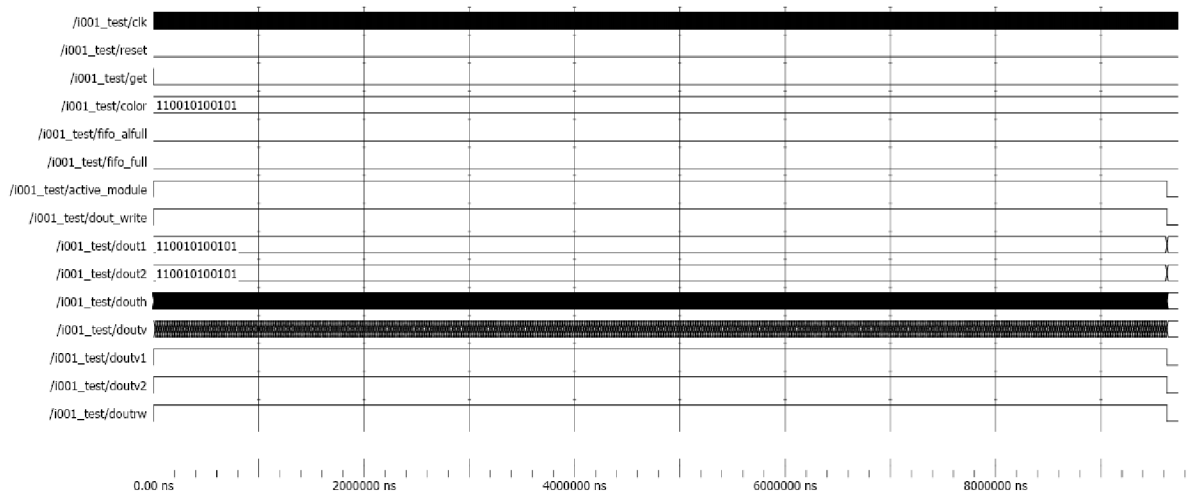
8.1 Mazání obrazovky

První instrukcí, kterou by měl instrukční program vždy vyslat by mělo být smazání obrazovky. Tato nutnost vychází přímo z principu SRAM paměti, kdy po vypnutí a následném zapnutí napájení jsou stavy jednotlivých buněk nahodilé.

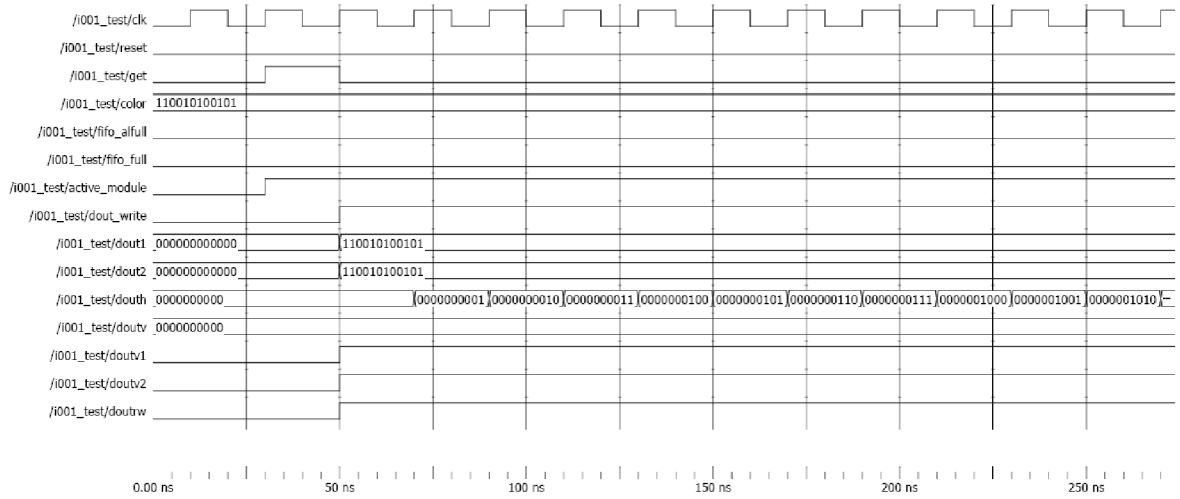
Algoritmus mazání obrazovky zapisuje do paměti páry pixelů s danou definovanou barvou – většinou černou.

Na obrázcích 20 až 22 je vidět postupně celý časový průběh testování modulu Instrukce 001, dále pak její aktivace a ukončení a klidový stav.

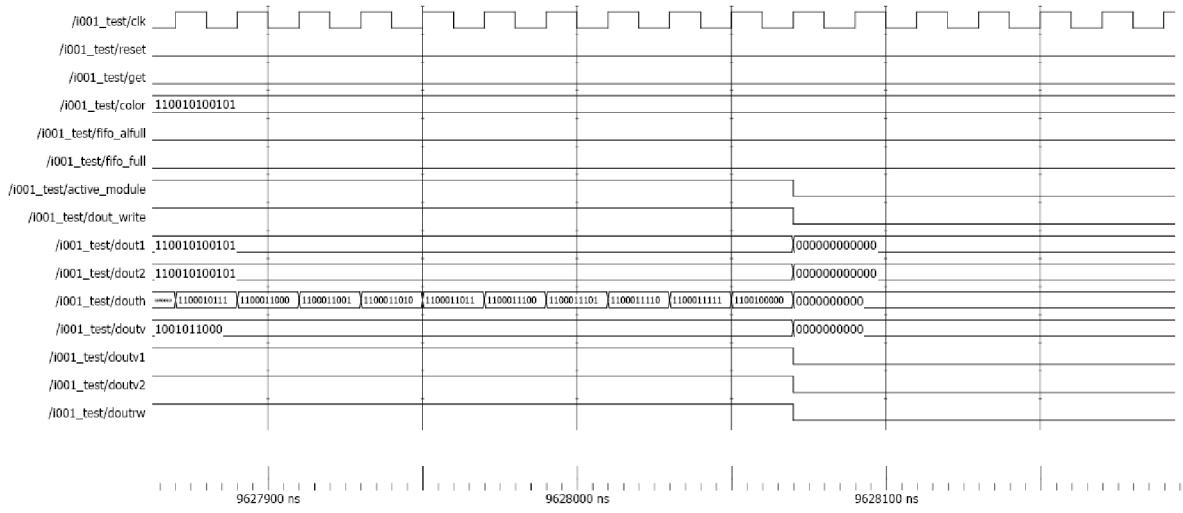
Zdrojové kódy jsou přiložené na CD.



Obrázek 20 Časový průběh testování modulu Instrukce 001 – celkový pohled



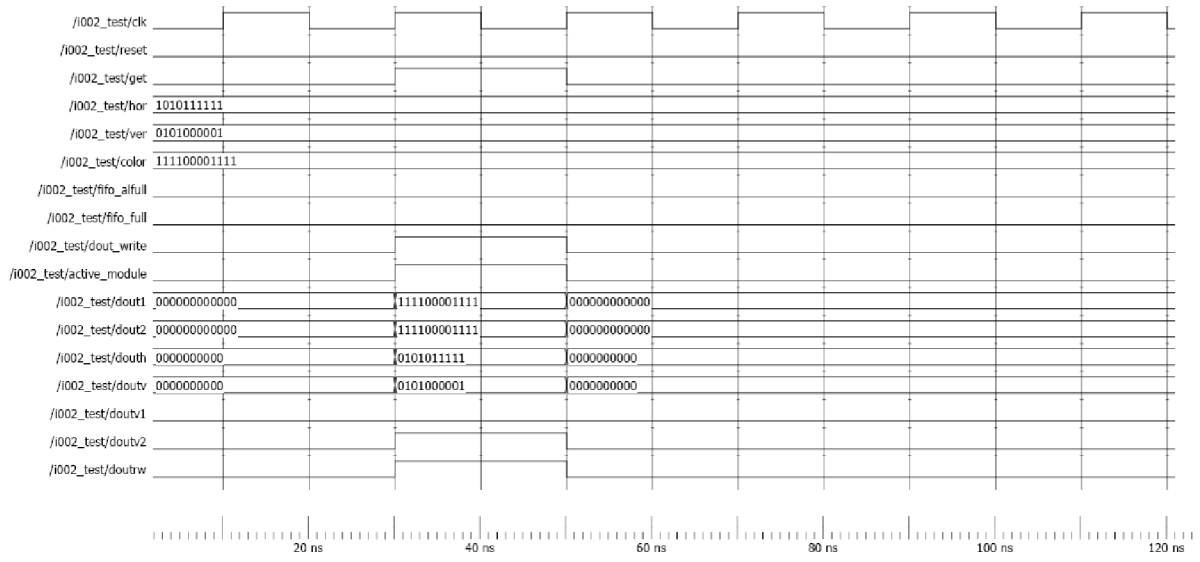
Obrázek 21 Časový průběh testování modulu Instrukce 001 – detailní pohled na aktivaci mudulu



Obrázek 22 Časový průběh testování modulu Instrukce 001 – detailní pohled na deaktivaci mudulu

8.2 Vykreslení pixelu

V některých případech může být požadavek na vykreslení konkrétního pixelu. Z těchto důvodů byla napsána Instrukce 002. Instrukce očekává informace o poloze bodu (*HOR*, *VER*) a barvě daného bodu (*COLOR*). Časový průběh simulace této instrukce je na obrázku 23, zdrojové kódy jsou na přiloženém CD.



Obrázek 23 Časový průběh testování modulu Instrukce 002 – celkový pohled

8.3 Algoritmy pro zobrazování přímých čar

Algoritmy můžeme obecně rozdělit na algoritmy se sekvenčním přístupem a na algoritmy s přímým přístupem k bodu. Většina metod je založena na přírůstkovém principu, tzn. že z dané pozice kreslíme čáru do pozice vzdálené o (Δx, Δy).

8.3.1 Digitální diferenciální analyzátor – DDA

Jednou z možností, jak lze algoritmus digitálního diferenciálního analyzátoru interpretovat, je diferenciální rovnice (1).

$$\frac{\Delta y}{\Delta x} = \frac{y_{konec} - y_{start}}{x_{konec} - x_{start}} \tag{1}$$

Vzhledem k tomu, že budeme uvažovat diskrétní přístup, lze zapsat:

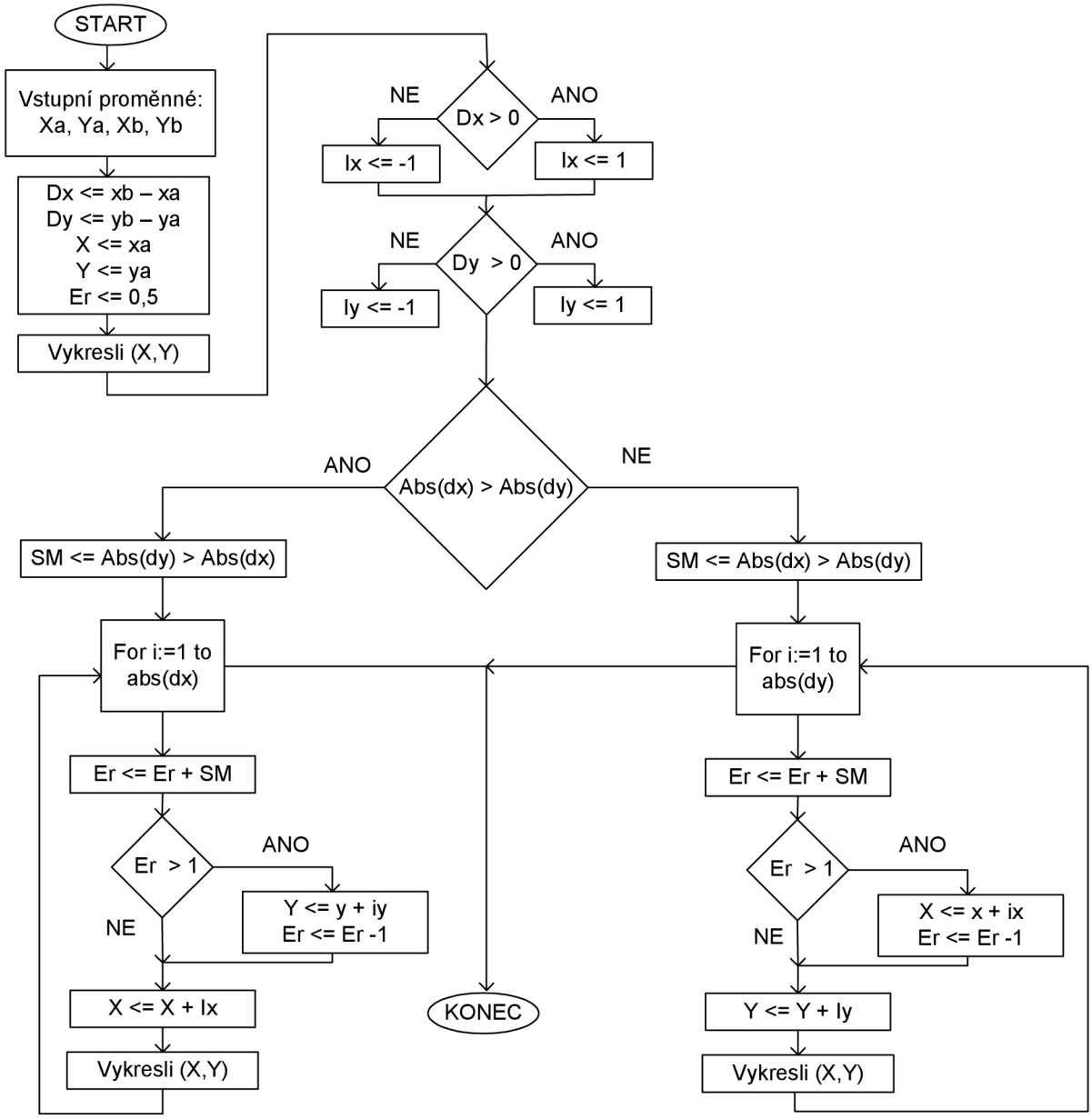
$$x_{i+1} = x_i + 1 \Rightarrow x_{i+1} - x_i = 1 = \Delta x \tag{2}$$

kde x_i je kterýkoliv z vypočtených bodů na x-ové ose. Od následujícího kroku se liší právě o 1. Obdobně pak lze odvodit následující:

$$y_{i+1} = y_i + \Delta y = y_i + \frac{y_{konec} - y_{start}}{x_{konec} - x_{start}} \tag{3}$$

Je zřejmé, že takto lze použít algoritmus pouze tam, kde $\Delta x \geq \Delta y$. V ostatních případech se musí zaměnit souřadnice x a y.

Algoritmus lze popsat pomocí blokového diagramu následovně:



Obrázek 24 Vývojový diagram DDA algoritmu

Uvažujme o vykreslení úsečky z <0;0> do <-8;-4>. Algoritmus bude mít pak následující stavy:

xa	ya	x	y	xb	yb	dx	dy	ix	iy	sm	er
0	0	0	0	-8	-4	-8	-4	-1	-1	0.5	0.5

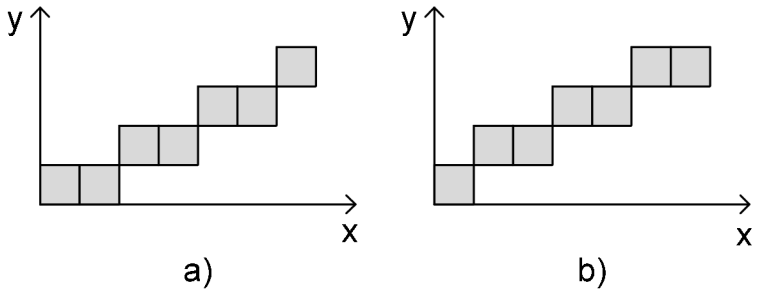
i	er	x	y
1	0	-1	-1
2	0.5	-2	-1
3	0	-3	-2
4	0.5	-4	-2
5	0	-5	-3
6	0.5	-6	-3
7	0	-7	-4
8	0.5	-8	-4

Pokud ovšem prohodíme počáteční a koncový bod, pak:

xa	ya	x	y	xb	yb	dx	dy	ix	iy	sm	er
-8	-4	-8	-4	0	0	8	4	1	1	0.5	0.5

i	er	x	y
1	0	-7	-3
2	0.5	-6	-3
3	0	-5	-2
4	0.5	-4	-2
5	0	-3	-1
6	0.5	-2	-1
7	0	-1	0
8	0.5	0	0

Výsledné zobrazení je pak na obrázku 25:



Obrázek 25 Porovnání vykreslování čáry pomocí DDA algoritmu z rozdílných bodů

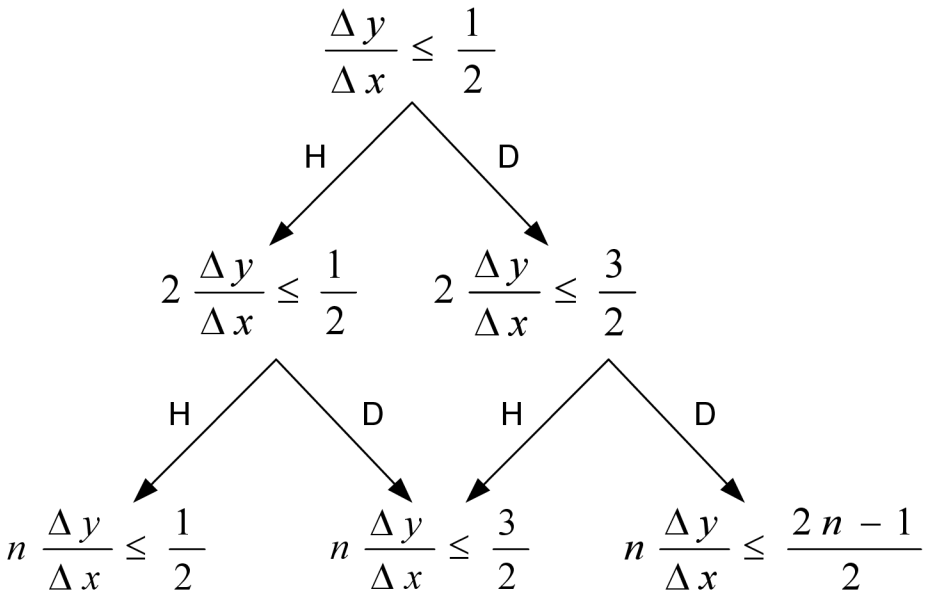
Porovnáním obou výsledků lze usoudit, že výpočet není symetrický. Pro výpočet je nutný procesor s pohyblivou řádovou čárkou. Tato skutečnost je pro použití v FPGA obvodech rozhodující – v jazyce VHDL totiž lze sice realizovat výpočet s pohyblivou čárkou, tento program pak lze použít pouze pro simulaci, nikoliv implementaci do cílového obvodu.

8.3.2 *Bresenhamův algoritmus*

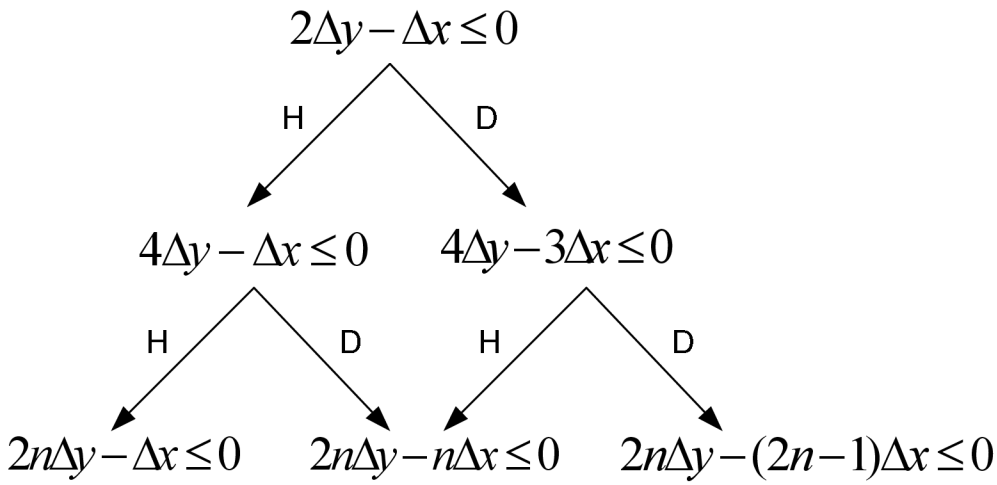
Tento algoritmus, publikovaný J.E.Bresenhamem v roce 1965, byl původně určen pro kreslící stoly (přírůstkové zapisovače). Je ale vhodný i pro výpočet úseček v rastrových zařízeních.

Algoritmus je založený na rozhodování, který pixel leží blíže hladké přímce. Rozhodování se děje za pomoci axiálního chybového členu a vybírá se bod podle směru přímky ve směru rychlejšího (rozuměj kratšího) pohybu. V tomto směru se pohybuje s přírůstkem jedné jednotky. Výhodou tohoto algoritmu je, že lze upravit do podoby, ve které není nutné používat operací s pohyblivou řádovou čárkou.

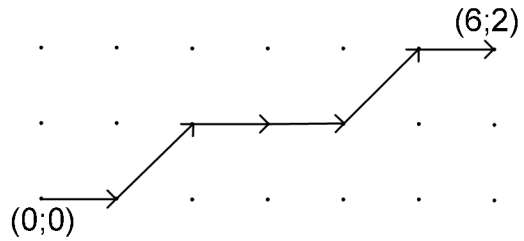
Odvození provedeme pro první oktant – tj. pro případ, že $0 \le |\Delta y| \le |\Delta x|$. Při rozhodování je rozhodující hodnota $\frac{\Delta y}{\Delta x} = 0{,}5$. Provedeme v každém bodě porovnání, ke kterému bodu v rastru má blíže hladká přímka. Tomuto bodu náležitě změníme atributy a pokračujeme při rozhodování dále. Popsaný algoritmus lze zobrazit následovně:



Naznačené směry v algoritmu označují H - horizontální směřování, D - diagonální směřování. Takto navržený algoritmus ovšem vyžaduje počítání s pohyblivou čárkou. Toto lze odstranit jednoduše vynásobením výrazů hodnotou $2\Delta x$ za předpokladu, že $\Delta x > 0$. Po úpravě dostaneme:



Pomocí tohoto algoritmu lze následovně prokreslit úsečku z bodu (0;0) do (6;2):

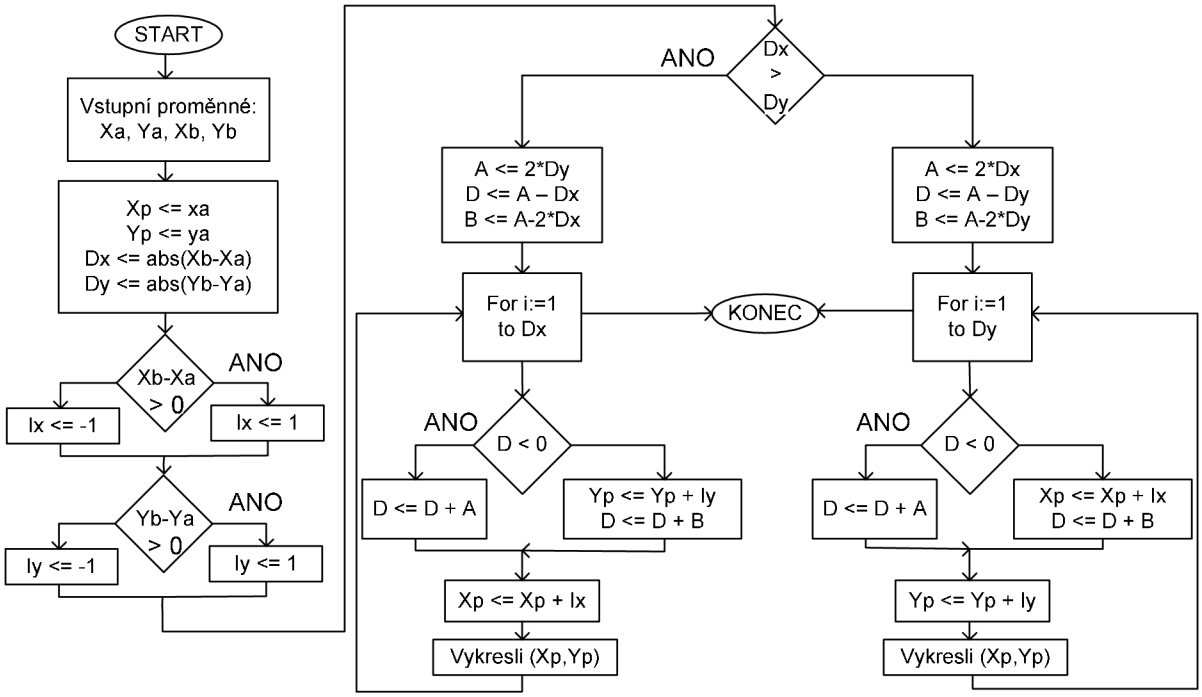


Obrázek 26 Vykreslování z bodu [0;0] do bodu [6;2] při použití Bresenhamova algoritmu

Postup výpočtu je následující: $\frac{\Delta y}{\Delta x}=\frac{2}{6}=\frac{1}{3}$

$$1)\ \frac{\Delta y}{\Delta x}=\frac{1}{3}\le\frac{1}{2}\xrightarrow[+]{}H$$
$$2)\ \frac{2\Delta y}{\Delta x}=\frac{2}{3}\le\frac{1}{2}\xrightarrow[-]{}D$$
$$3)\ \frac{3\Delta y}{\Delta x}=\frac{3}{3}\le\frac{3}{2}\xrightarrow[+]{}H$$
$$4)\ \frac{4\Delta y}{\Delta x}=\frac{4}{3}\le\frac{3}{2}\xrightarrow[+]{}H$$
$$5)\ \frac{5\Delta y}{\Delta x}=\frac{5}{3}\le\frac{3}{2}\xrightarrow[-]{}D$$
$$6)\ \frac{6\Delta y}{\Delta x}=\frac{6}{3}\le\frac{5}{2}\xrightarrow[+]{}H$$

Celý zjednodušený algoritmus lze popsat pro všechny oktanty následovně:

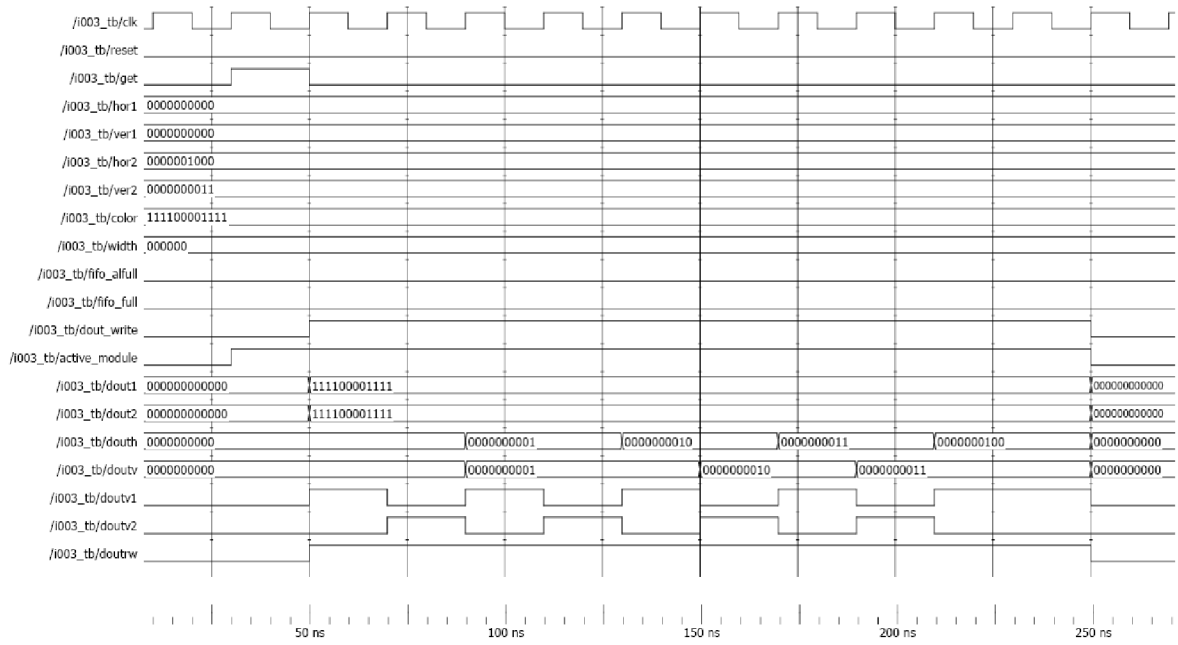


Obrázek 27 Vývojový diagram Bresenhamova přímkového algoritmu

Algoritmus je opět nesymetrický a záleží na konkrétním zadání souřadnic i samotném přepisu algoritmu (resp. znamének v podmínkách).

Při porovnání DDA a Bresenhamova algoritmu je zřejmé, že Bresenhamův algoritmus obsahuje v cyklu test zapříčiňující, že maximální chyba interpolace je $\pm\frac{1}{2}$ rastrovací jednotky. Algoritmus DDA je navíc více citlivý na přesnost a zejména u delších úseček může dojít ke kumulaci zaokrouhlovací chyby nepřesně stanovených přírůstků Δx a Δy.

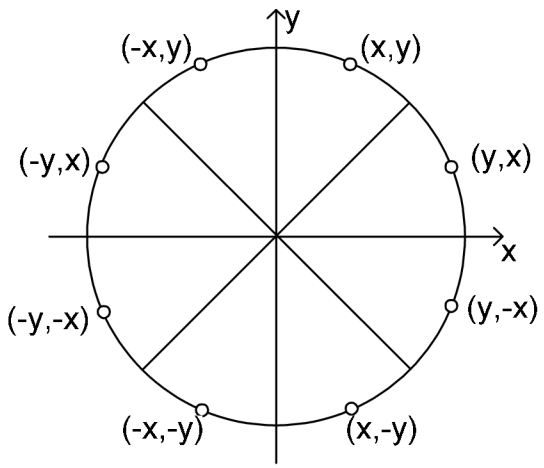
Časový průběh simulace testování obvodu je na obrázku 28. Při tomto testování program generoval posloupnost z bodu [0;0] do bodu [8;4].



Obrázek 28 Časový průběh testování modulu Instrukce 003 – celkový pohled vykreslování z [0;0] do [8;4]

8.4 Algoritmy pro zobrazování kružnice a jejích částí

Při vykreslování kružnice je výhodné uvědomit si, že je třeba vypočítat pouze 1 bod v každém oktantu. Následně lze dopočítat souřadnice podle obrázku 2-4. Takto lze samozřejmě dopočítat souřadnice jednotlivých bodů v případě, že střed kružnice leží v nule, ale při znalosti této symetrie lze dopočítat velmi jednoduše souřadnice i ve standardním případě, kdy střed v počátku neleží.



Obrázek 29 Převod souřadnic mezi všemi oktanty

V dalším textu nejsou uvedené algoritmy na bázi DDA – DDA a High precission DDA. Tyto algoritmy vykreslují kružnice za pomoci úseček, ale pro vykreslování potřebují vypočítávat hodnotu funkce sinus. Tato varianta by byla řešitelná ve VHDL za pomoci statických tabulek. Poměr výkonu a vizuálního vjemu je špatný a zůstaňme tedy pouze u konstatování existence těchto algoritmů.

8.4.1 Bersenhamův algoritmus pro kružnici

Při odvozování Bersenhamova algoritmu pro kružnici je nejprve nutné si uvědomit, že začátek generování bude v bodě $x=0$, $y=R$ a v počátku souřadné soustavy. Pokud budeme generovat ve směru hodinových ručiček, bude se zvětšovat x a zmenšovat y. Lze odvodit, že jsou možné pouze tři směry: vertikální (dále „V“), horizontální („H“) a diagonální („D“). Pokud vezmeme v úvahu rovnici kružnice

$$x^2 + y^2 - R^2 = 0 \tag{4}$$

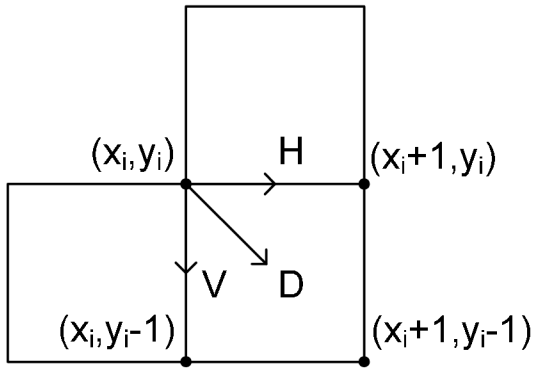
pak můžeme pro jednotlivé směry zvážit dráhu k následujícímu pixelu:

$$"H" = \left|(x_i + 1)^2 + y_i^2 - R^2\right| \tag{5}$$

$$"D" = \left|(x_i + 1)^2 + (y_i - 1)^2 - R^2\right| \tag{6}$$

$$"V" = \left|x_i^2 + (y_i - 1)^2 - R^2\right| \tag{7}$$

Vybrán bude ten směr, který bude nabývat nejmenší absolutní hodnoty. Situace je naznačena na obrázku 30:



Obrázek 30 Postup do dalšího bodu při generování pomocí Bresenhamova algoritmu pro kružnici

Nyní vypočtěme rozdíl kvadrátu vzdálenosti bodu $(x_i + 1, y_i - 1)$ od počátku kružnice - tedy při kroku D – a kvadrátu poloměru kružnice:

$$\Delta_i = (x_i + 1)^2 + (y_i - 1)^2 - R^2 \tag{8}$$

Obdobně jako u Bresenhamova algoritmu pro vykreslení čáry je zde podstatné znaménko chyby. Pokud bude $\Delta_i < 0$, pak bude bod $(x_i + 1, y_i - 1)$ ležet uvnitř kružnice a jsou přípustné pouze kroky „H“ a „D“. Zde lze rozhodnout porovnáním rovnic 5 a 6:

$$\delta = "H" - "V" = \left|(x_i + 1)^2 + y_i^2 - R^2\right| - \left|(x_i + 1)^2 + (y_i - 1)^2 - R^2\right| \tag{9}$$

Pokud bude $\delta \leq 0$, pak se vybere krok horizontálním směrem „H“, v opačném případě je krok diagonální – „D“.

Pro první případ platí:

$$(x_i + 1)^2 + y_i^2 - R^2 \geq 0 \tag{10}$$

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 < 0 \tag{11}$$

Pokud známe znaménka, pak můžeme rovnici 9 přepsat následovně:

$$\begin{aligned} \delta &= (x_i + 1)^2 + y_i^2 - R^2 + (x_i + 1)^2 + (y_i - 1)^2 - R^2 = \\ &= 2\left[(x_i + 1)^2 + (y_i - 1)^2 - R^2 + y_i\right] - 1 = \\ &= 2(\Delta_i + y_i) - 1 \end{aligned} \tag{12}$$

Analogicky lze odvodit celou tabulku 2, která mapuje všechny přípustné možnosti kroků:

D_i	d_i	krok
$D_i < 0$	$d_i <= 0$	"H"
$D_i < 0$	$d_i > 0$	"D"
$D_i = 0$		"D"
$D_i > 0$	$d_i <= 0$	"D"
$D_i > 0$	$d_i > 0$	"V"

Tabulka 2 Vývoj výpočtu při použití Bresenhamova algoritmu

Abychom nemuseli neustále opakovat výpočet jednotlivých proměnných, upravíme vztahy do podoby rekurentního výpočtu . Nejprve uvažujme, že se bude pokračovat krokem „H“:

$$x_{i+1} = x_i + 1 \tag{13}$$

$$y_{i+1} = y_i \tag{14}$$

$$\Delta_{i+1} = (x_{i+1}+1)^2 + (y_{i+1}-1)^2 - R^2 = (x_{i+1}+1)^2 + (y_i - 1)^2 - R^2 =$$
$$= (x_i+1)^2 + (y_i-1)^2 - R^2 + 2x_{i+1} + 1 = \quad (15)$$
$$= \Delta_i + 2x_{i+1} + 1$$

Obdobně lze odvodit vyjádření pro krok „D“:¨

$$x_{i+1} = x_i + 1 \quad (16)$$

$$y_{i+1} = y_i - 1 \quad (17)$$

$$\Delta_{i+1} = \Delta_i + 2x_{i+1} - 2y_{i+1} + 2 \quad (18)$$

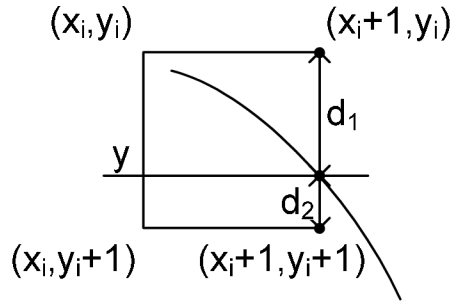
A obdobně i pro krok „V“:¨

$$x_{i+1} = x_i \quad (19)$$

$$y_{i+1} = y_i - 1 \quad (20)$$

$$\Delta_{i+1} = \Delta_i - 2y_{i+1} + 1 \quad (21)$$

Pokud chceme generovat pouze první oktant, pak budeme generovat z bodu $(0, R)$ do (x, y).



Obrázek 31 Variantní postavení hladké kružnice a rastru

Nyní uvažujme případ na obrázku 31. Při uvažovaní vyjděme z rovnice 4. Lze pak napsat následující:

$$d_1 = y_i^2 - y^2 = y_i^2 - R^2 + (x_i+1)^2 \quad (22)$$

$$d_2 = y^2 - y_{i-1}^2 = R^2 - (x_i+1)^2 - (y_i - 1)^2 \quad (23)$$

Pak :

$$p_i = d_1 - d_2 = 2(x_i+1)^2 - y_i^2 + (y_i - 1)^2 - 2R^2 \quad (24)$$

Po dosazení počátečního bodu $(0, R)$ získáme:

$$p_0 = 3 - 2R \tag{25}$$

Pro rekurentní výpočet je nezbytné vyjádřit p_{i+1} jako funkci p_i, resp. souřadnic v předchozím kroku. Potom:

$$\begin{aligned} p_{i+1} &= 2(x_{i+1}+1)^2 + y_{i+1}^2 + (y_{i+1}-1)^2 - 2R = \\ &= 2[(x_i+1)+1]^2 + y_{i+1}^2 + (y_{i+1}-1)^2 - 2R \end{aligned} \tag{26}$$

Po úpravě lze dostat následující tvar:

$$p_{i+1} = p_i + 4x_i + 6 + 2(y_{i+1}^2 - y_i^2) - 2(y_{i+1} - y_i) \tag{27}$$

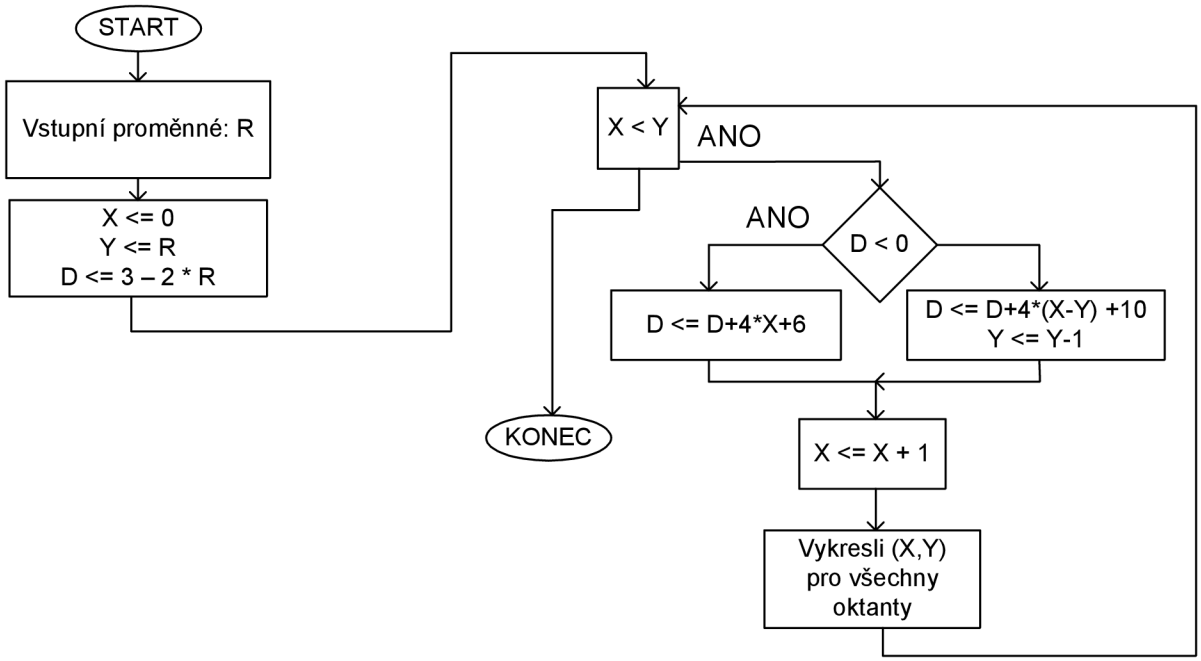
Pokud bude $p_i < 0$, pak bude následovat krok horizontální a po dosazení bodu $(x_i + 1, y_i)$ získáváme:

$$p_{i+1} = p_i + 4x_i + 6 \tag{28}$$

zatímco pokud bude $p_i < 0$, pak bude následovat krok diagonální a tedy po dosazení bodu $(x_i + 1, y_i - 1)$ získáváme:

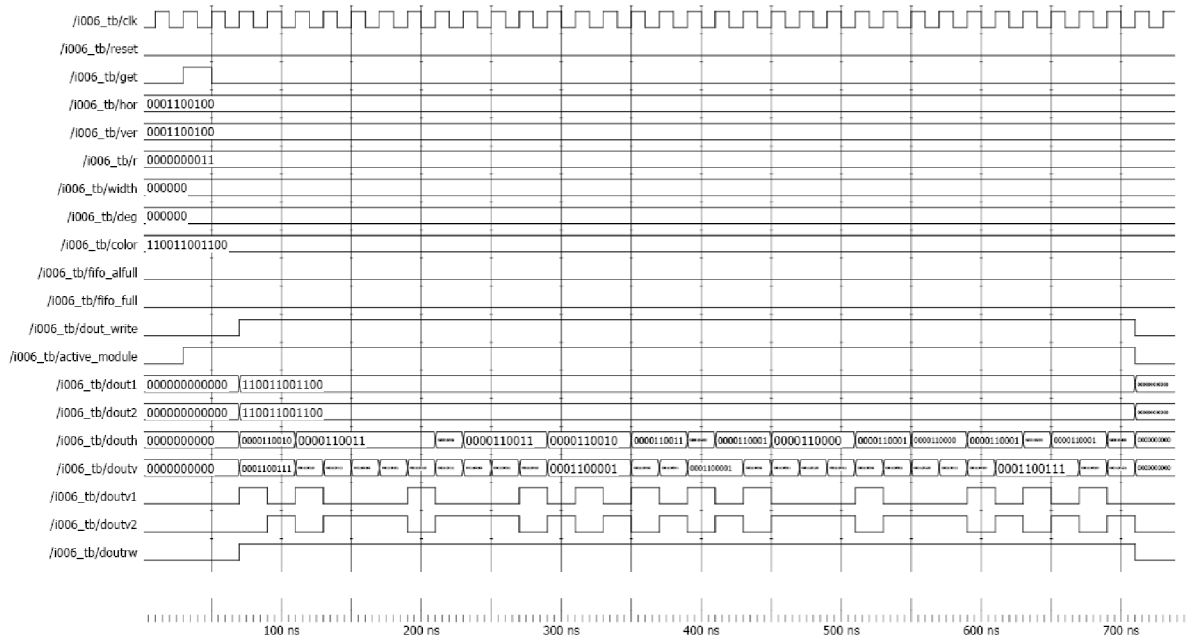
$$p_{i+1} = p_i + 4(x_i - y_i) + 10 \tag{29}$$

Zjednodušený algoritmus lze zapsat následovně:



Obrázek 32 Vývojový diagram Bresenhamova algoritmu pro kružnici

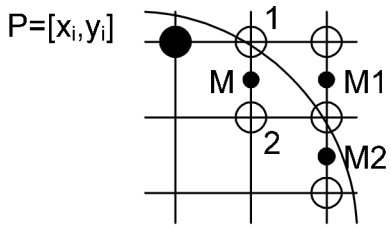
Na obrázku 33 je časový průběh simulace výpočtu kružnice o R=3 a středu [100;100]. Na simulaci jsou postupně vykresleny všechny oktanty.



Obrázek 33 Časový průběh testování modulu Instrukce 006 – vykreslení kružnice při použití Bresenhamova algoritmu pro kružnici

8.4.2 Kružnice s kritériem středového bodu

Pro volbu vhodného interpolačního kroku lze použít i kritérium středového bodu. Na obrázku 34 je naznačena interpolace kružnice se středem v počátku ve druhém oktantu v záporném směru otáčení.



Obrázek 34 Interpretace vykreslení kružnice s kritériem středového bodu

Při rozhodování mezi body 1 a 2 a tedy mezi krokem v horizontálním a diagonálním směru, vyjdeme opět ze znalosti rovnice 4. Určíme hodnotu rozhodovacího členu D pro bod M:

$$D_M = (x_i + 1)^2 + \left(y_i - \frac{1}{2}\right)^2 - R^2 \qquad (30)$$

Při volbě bodu 1 (horizontálního kroku) má následující rozhodovací člen v bodě M1 hodnotu

$$D_{M1} = D + 2x_i + 3 = D + D_1 \tag{31}$$

Při volbě bodu 2 (diagonálního směru) má pak následující rozhodovací člen v bodě M2 hodnotu

$$D_{M2} = D + 2x_i - 2y_i + 5 = D + D_2 \tag{32}$$

přičemž D_1 a D_2 představují možné přírůstky rozhodovacího členu D_M. Jedná se o diference prvního řádu. Při výpočtu nových hodnot D_1 a D_2 můžeme použít jednoduché vztahy – diference druhého řádu. Jednoduše pak lze odvodit, že při kroku horizontálním směrem jsou hodnoty rozhodovacích členů následující:

$$D_{M+1} = D_M + D_1 \tag{33}$$

$$D_{1+1} = D_1 + 2 \tag{34}$$

$$D_{2+1} = D_2 + 2 \tag{35}$$

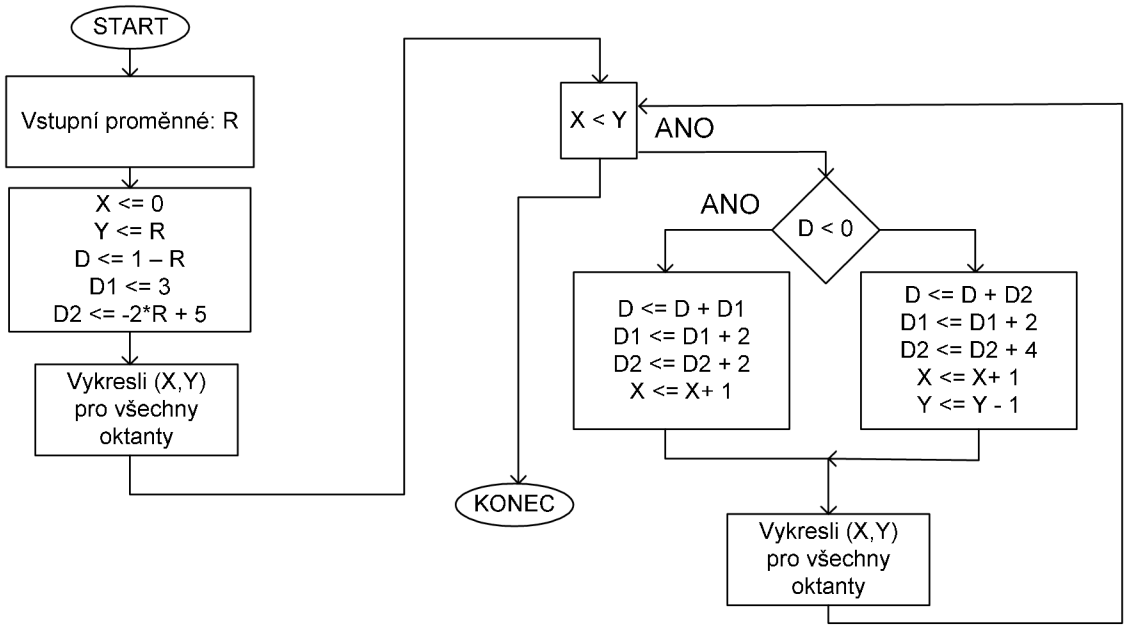
Při kroku diagoálním směrem jsou pak hodnoty rozhodovacích členů takovéto:

$$D_{M+1} = D_M + D_2 \tag{33}$$

$$D_{1+1} = D_1 + 2 \tag{34}$$

$$D_{2+1} = D_2 + 4 \tag{35}$$

S použitím těchto vztahů lze vytvořit jednoduchý algoritmus pro vykreslování kružnice.



Obrázek 35 Vývojový diagram při kresbě kružnice kritériem středového bodu

8.5 Algoritmy pro kresbu elipsy

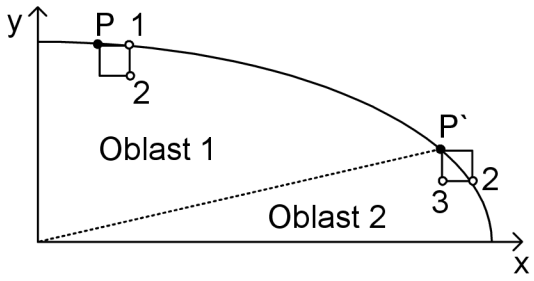
Algoritmy pro vykreslování elips byly odvozeny podobným způsobem jako algoritmy pro vykreslování kružnice. Elipsu se středem v bodě (x_c, y_c), jejíž hlavní poloosy a a b jsou rovnoběžné se souřadnicovými osami, lze popsat rovnicí 36:

$$\frac{(x-x_c)^2}{a^2}+\frac{(y-y_c)^2}{b^2}=1 \tag{36}$$

Při výpočtu bude jednodušší určovat body na elipse, která má střed v počátku, a rovnici 36 můžeme převést do tvaru:

$$F(x,y)=b^2x^2+a^2y^2-a^2b^2 \tag{37}$$

Pomocí symetrie lze dopočítat v případě elipsy pouze 4 body ležící na ní. Algortimus jako takový je velice podobný algortimu pro výpočet kružnice s kritériem středového bodu pouze s tím rozdílem, že je nutné rozdělit výpočet do dvou částí, které jsou naznačeny v obrázku 36.



Obrázek 36 Výpočet bodů elipsy

Hranice oblastí 1 a 2 je v bodě, kde $a^2y^2=b^2x^2$. V prvním kvadrantu lze stanovit tyto pravidla pro vykreslování:

Oblast 1:

$$D_{i+1}+D_1=D_i+b^2(2x+3) \tag{38}$$

$$D_{i+1}+D_2=D_i+b^2(2x+3)+a^2(-2y+2) \tag{39}$$

Oblast 2:

$$D_{i+1}+D_3=D_i+a^2(-2y+3) \tag{40}$$

$$D_{i+1}+D_2=D_i+b^2(2x+2)+a^2(-2y+3) \tag{41}$$

Pro počáteční predikci platí:

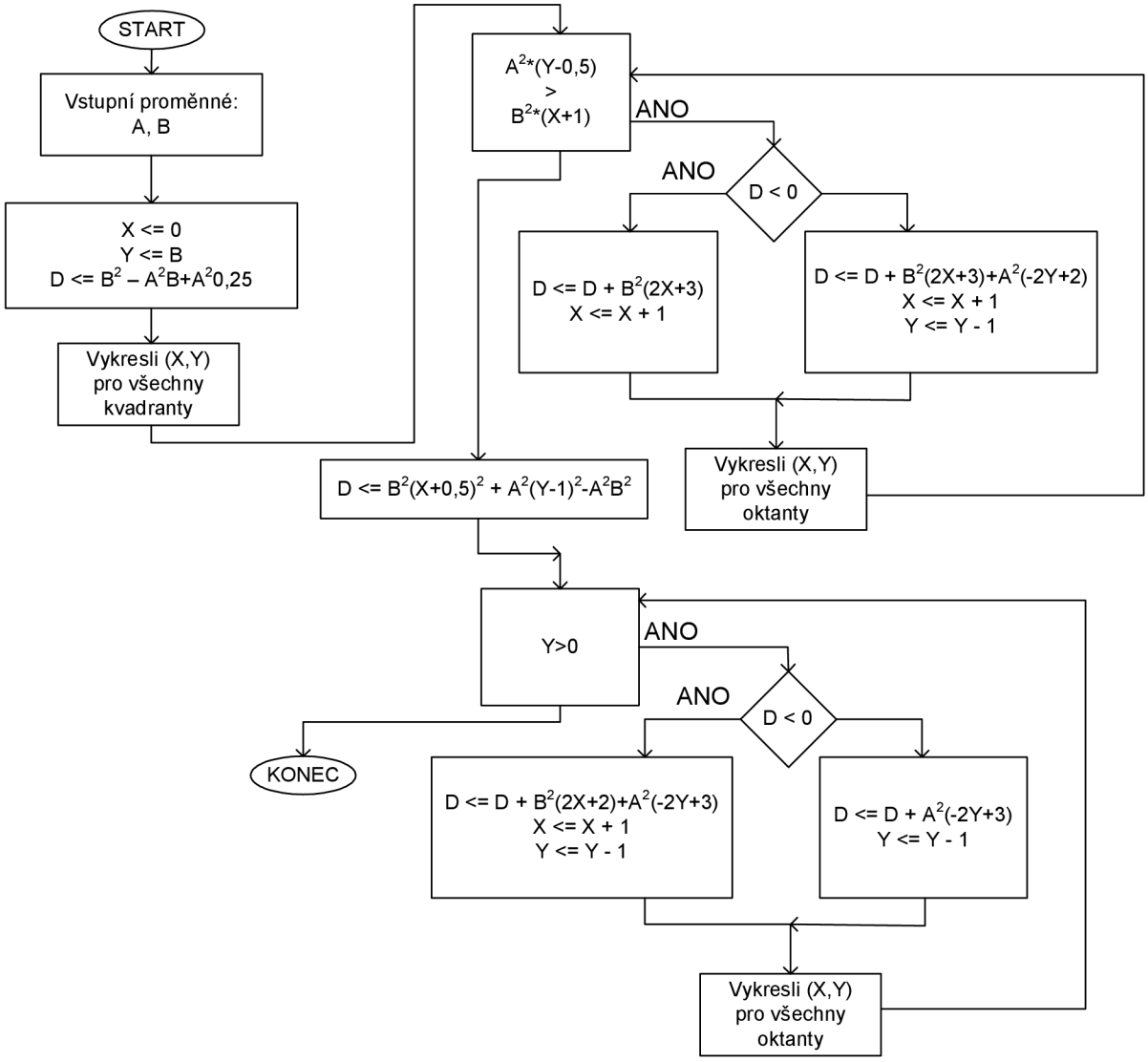
$$D_i = b^2 - ba^2 + \frac{a^2}{4} \tag{42}$$

Při této predikci může dojít k malé a ve většině případů i zanedbatelné chybě v případech, kdy *a* je liché. Při implementaci do cílového obvodu je nutno použít celočíselné výpočty. Dělení čtyřmi lze docílit velice jednoduše tak, že:

$$\frac{a(10:0)^2}{4} = a(10:2)^2 \tag{43}$$

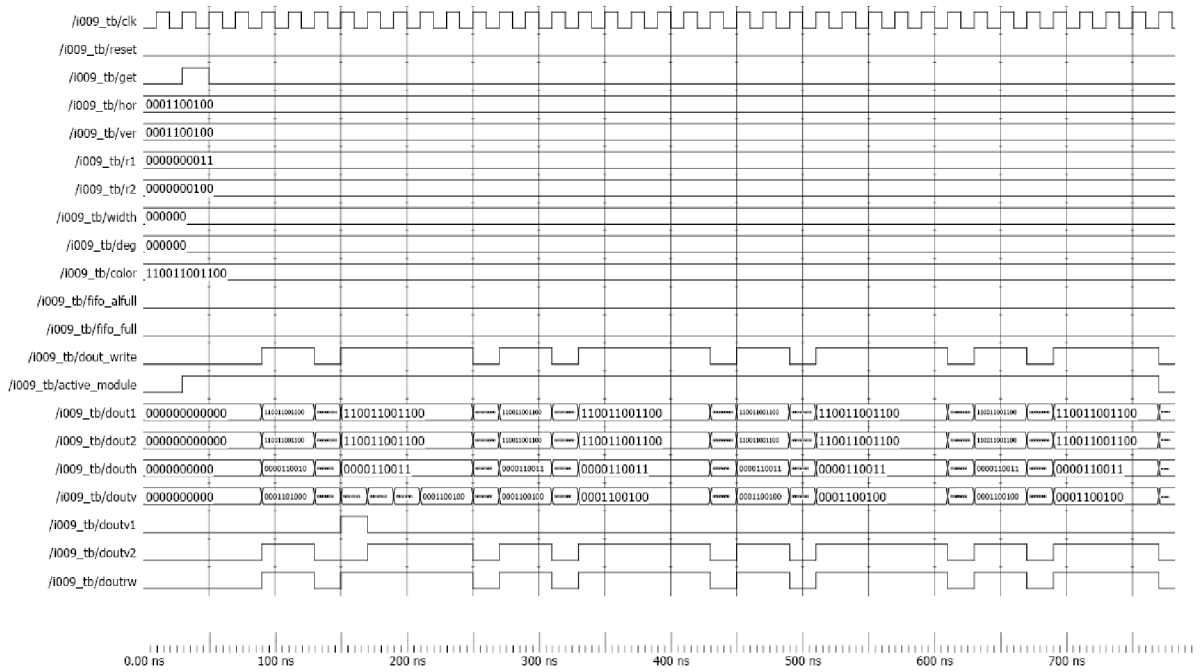
přičemž tedy dělení čísel 4, 5, 6 a čísla 7 atp. bude mít vždy stejný výsledek, tj. 1. V rovnici 43 je naznačeno bitové uvažování čísel – ve VHDL je pak číslo *a* označeno jako *std_logic_vector*.

Na následujícím obrázku je znázorněn algoritmus vykreslující elipsu:



Obrázek 37 Vývojový diagram algoritmu pro výpočet elipsy

Na obrázku 38 je časový průběh simulace pro A=3, B=4. K algoritmu elipsy je nutno podotknout, že je vzhledem k druhým mocninám a mezím výpočtům je implementace velice náročná na použité základní jednotky cílového obvodu SPARTAN-3.



Obrázek 38 Časový průběh testování modulu Instrukce 009 – vykreslení elipsy

8.6 Kopírování dat v paměti

Posunutí je nejjednodušší formou práce s vygenerovanými daty v paměti. Lze jej popsat v následujícím maticovém zápisu:

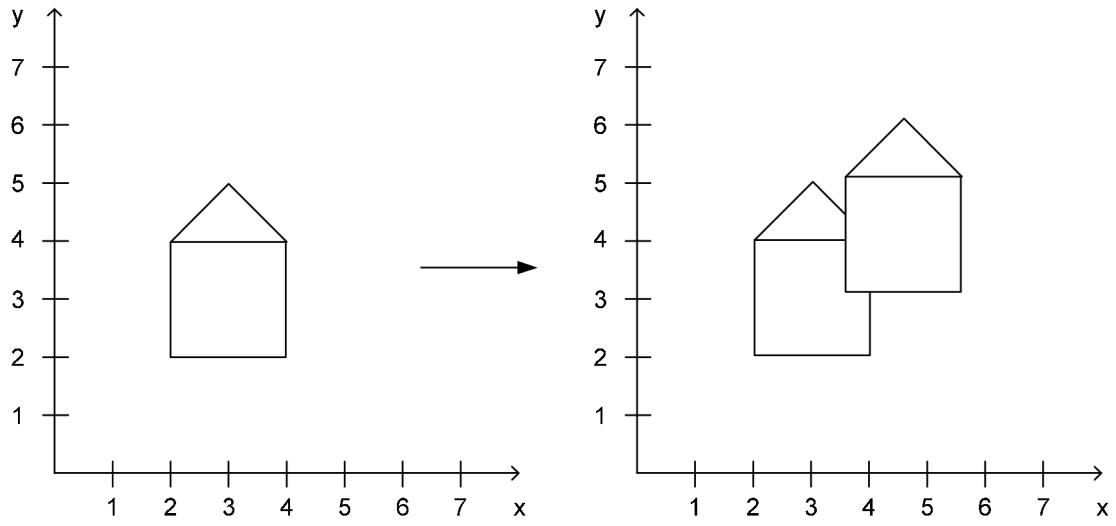
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a \\ b \end{bmatrix} \tag{44}$$

Kde $\begin{bmatrix} x' \\ y' \end{bmatrix}$ je nová poloha objektu, $\begin{bmatrix} x \\ y \end{bmatrix}$ je počáteční poloha objektu a $\begin{bmatrix} a \\ b \end{bmatrix}$ je vektor posunutí. Sčítání vektorů lze popsat následovně:

$$x' = x + a \tag{45}$$

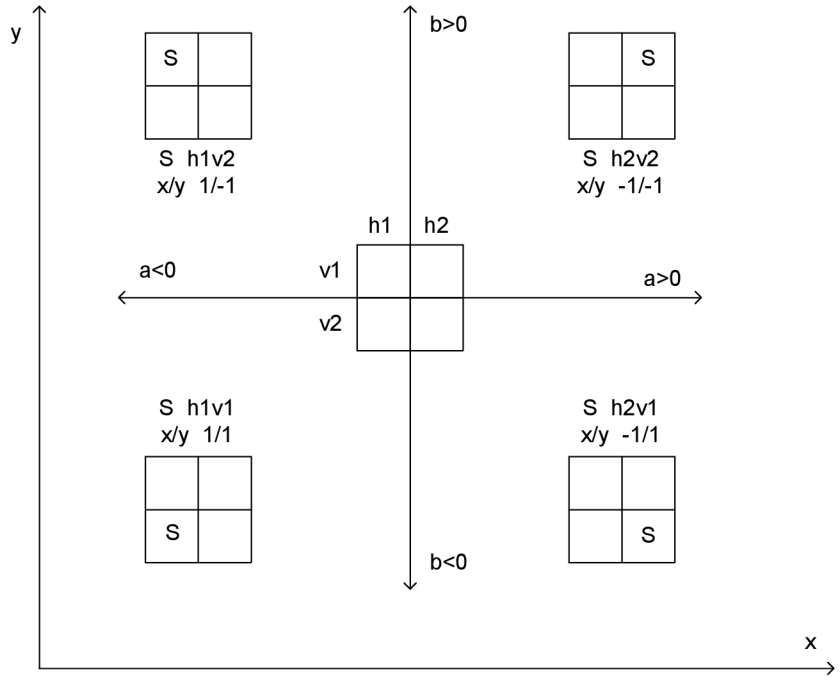
$$y' = y + b \tag{46}$$

Posunutí o vektor $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$ je zobrazeno na obrázku 39.



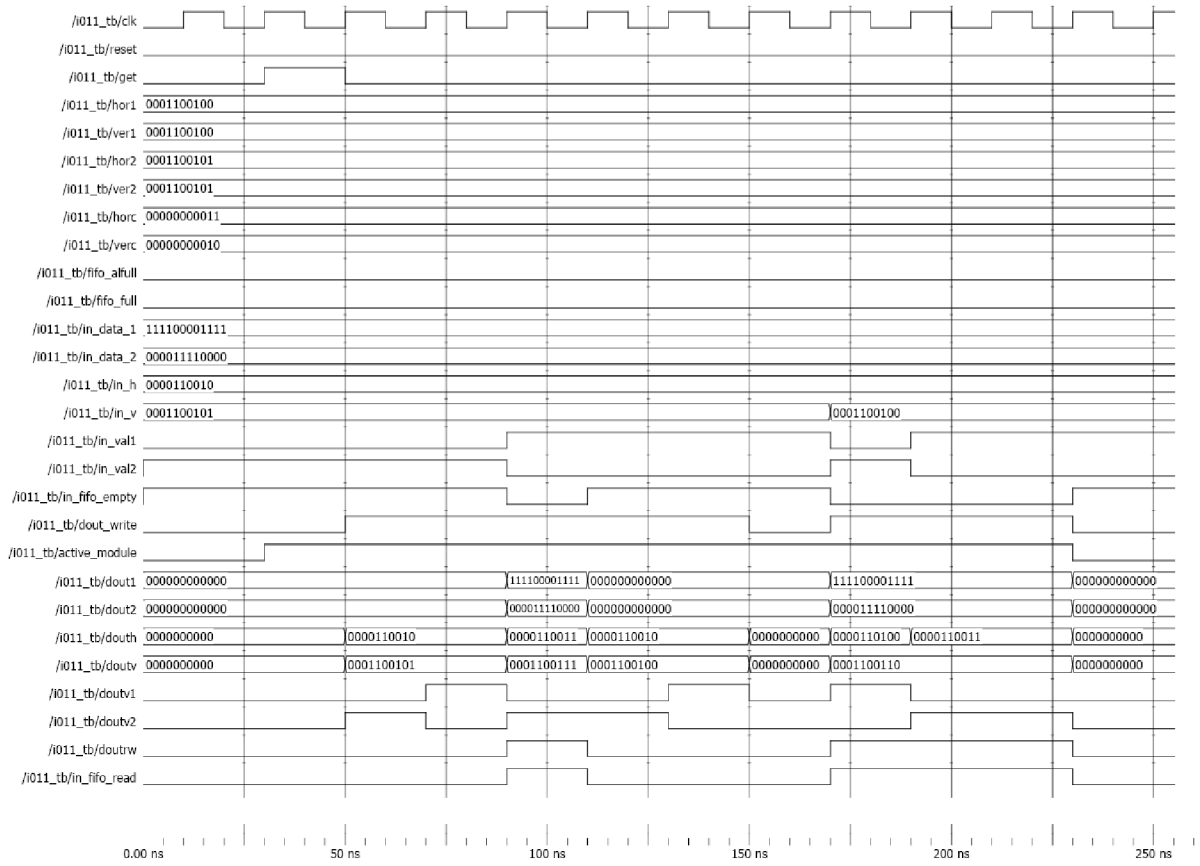
Obrázek 39 Kopírování dat v paměti

Jelikož je možnost, že se objekty budou navzájem překrývat, proto je potřeba pro všechny 4 směry stanovit počáteční body, od nich se pak budou odvíjet přírůstky +1 resp. -1 pro souřadnice X a Y:



Obrázek 40 Určení počátečních bodů a přírůstků jednotlivých souřadnic

Výstup z časové simulace je na obrázku 41. Jedná se o kopírování z oblasti [100,100] do [101,101] o +3 body v horizontální poloze a +2 body ve vertikální poloze.



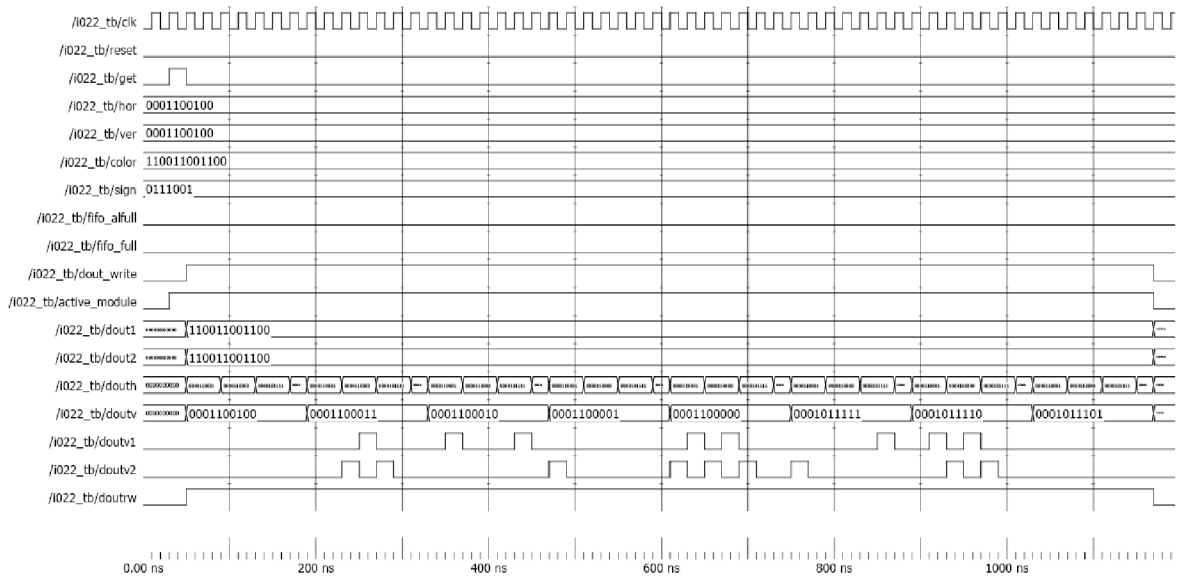
Obrázek 41 Časový průběh testování modulu Instrukce 011 – kopírování

8.7 Generátor znaků ASCII

ASCII (American Standard Code for Information Interchange) je americký standard pro výměnu informací. Jedná se o kódovou tabulku se znaky anglické abecedy a znaky používané v informatice. Tabulka kromě tisknutelných znaků jako jsou písmena, číslice a speciální znaky (interpunkce, závorky atd.) obsahuje i netisknutelné znaky, které byly původně určeny pro řízení tiskáren a dalších periferií.

Kód ASCII je 7mi bitový, tzn. umožňuje postihnout 128 znaků. V rámci Modulu Instrukce 022 lze zapisovat do obrazové paměti tisknutelné znaky, tj. znaky 33 až 126.

Na obrázku 42 je výstup časového průběhu simulace pro znak 57 (tj. „9“) a pro souřadnice [100,100].



Obrázek 42 Časový průběh testování modulu Instrukce 022 – generátor znaků ASCII

8.8 Vyplňování oblastí

Rozlišujeme dvě základní metody plnění (vyplňování) oblastí:

- řádkové vyplňování – v základní formě jej lze využít pouze tam, kde je oblast ohraničena geometricky, v případě rastrované grafiky lze po drobných úpravách algoritmů využít při vykreslování složitějších obrazců (obdélníky, kružnice, elipsy…)
- semínkové vyplňování – tvar oblasti může být teoreticky jakýkoliv, je potřeba znát barvu hranice nebo vnitřní oblasti, algoritmy jsou založeny na semínkovém vyplňování vycházejícího z uživatelem vybraného vnitřního bodu oblasti

8.8.1 Semínkové vyplňování

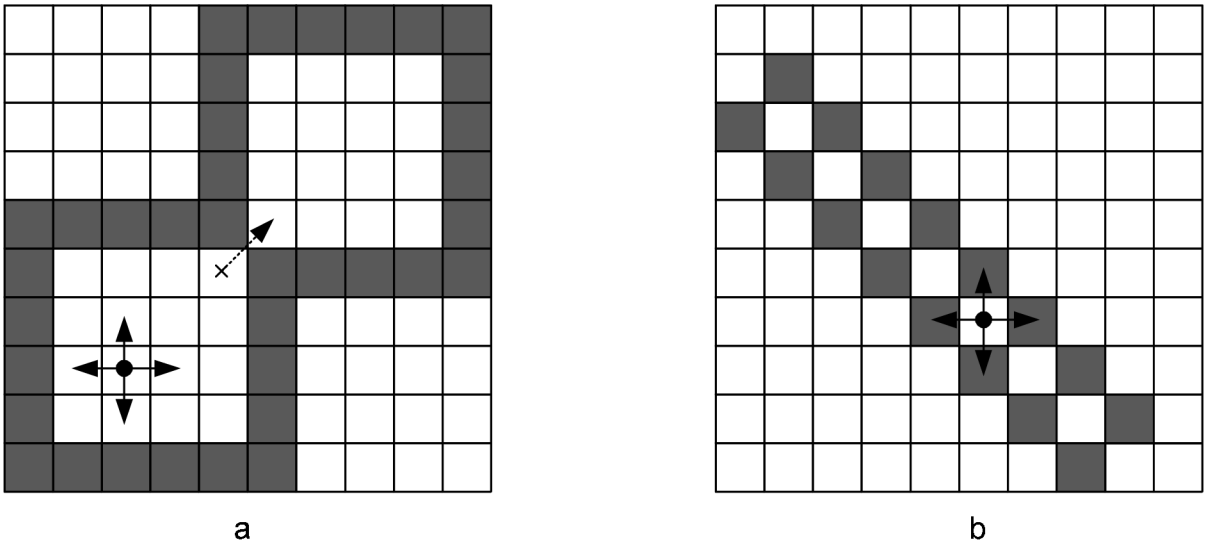
Algoritmus semínkového vyplňování je založený na aktivaci čtyř sousedních bodů, které pak slouží jako startovací body při dalším kroku. Hranice není pevně určena, ale všechny informace se zpětně načítají z obrazové paměti. Algoritmus je rekurzivní a je velice neefektivní. Toto řešení je v zápise elegantní, avšak tím, že v každém kroku vyvolává čtyřikrát sám sebe, může dojít ve větších obrazcích k přečerpání zásobníku pro předávání parametrů a přeplnění návratových adres podprogramu. Šíření semínek všemi směry má za následek to, že v některých případech dochází k testování jednoho pixelu pětkrát, zatímco k obarvení pouze jednou.

Existuje několik variant testování, zda bod náleží vnitřní oblasti:

- hraniční vyplňování – testovaný bod je vnitřní pokud má jinou barvu než je zadaná barva hranice
- záplavové vyplňování – testovaný bod je vnitřní, pokud má stejnou barvu jako zadané semínko – v některých literaturách je označována tato metoda jako lavinová nebo přebarvování
- měkké vyplňování – testovaný bod je vnitřní pokud má výrazně jinou barvu než je zadaná barva kružnice, tato varianta se používá v případech, kdy je hranice vyhlazena a má nestejnou barvu.

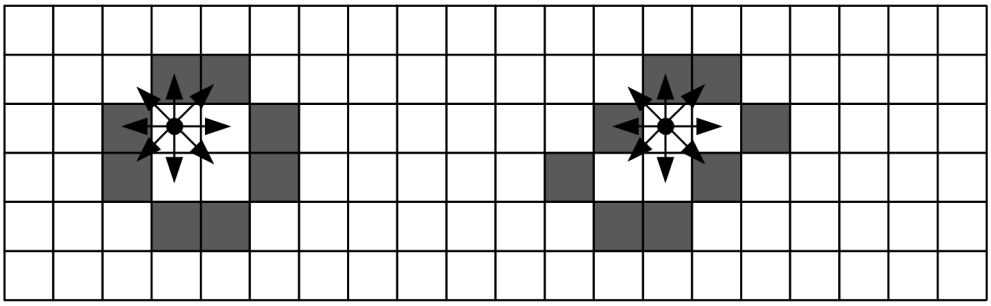
V konkrétním algoritmu se tyto varianty liší pouze v zápisu podmínek pro volání dalšího kroku.

V některých případech tento algoritmus selhává – a to zejména tam, kde jsou čáry daného mnohoúhelníku blízko sebe:



Obrázek 43 Problematické tvary pro semínkové vyplňování

Jak je patrno z obrázku 43, v případě a) bude vyplněna pouze jedna polovina obrazce, v případě b) dokonce dojde pouze k vybarvení semínka a proces již dále nepokračuje. Pokud bychom vyplňovali všemi osmi směry, mohlo by zase v případech na obrázku 44 dojít k tomu, že se dostaneme mimo vnitřní oblast. Ani jeden z přístupů tedy nemá pouze pro a nelze paušálně říci, který je lepší.



Obrázek 44 Problematické tvary pro semínkové vyplňování

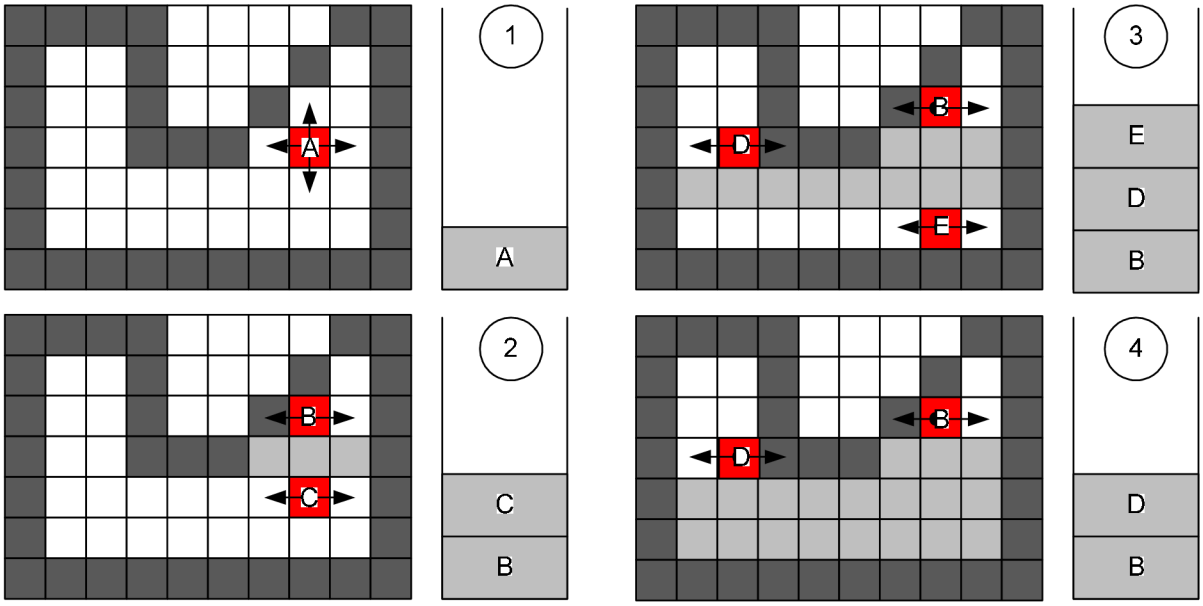
8.8.2 Řádkové semínkové vyplňování

Metoda snižující počet vstupů do obrazové paměti se nazývá řádkové semínkové vyplňování (v anglické literatuře Scan line seed fill). Rekurze je zde odstraněna použitím malého zásobníku, ve kterém jsou uchovány souřadnice několika málo vnitřních bodů z vyplňované oblasti.

Celý algoritmus lze popsat následovně:

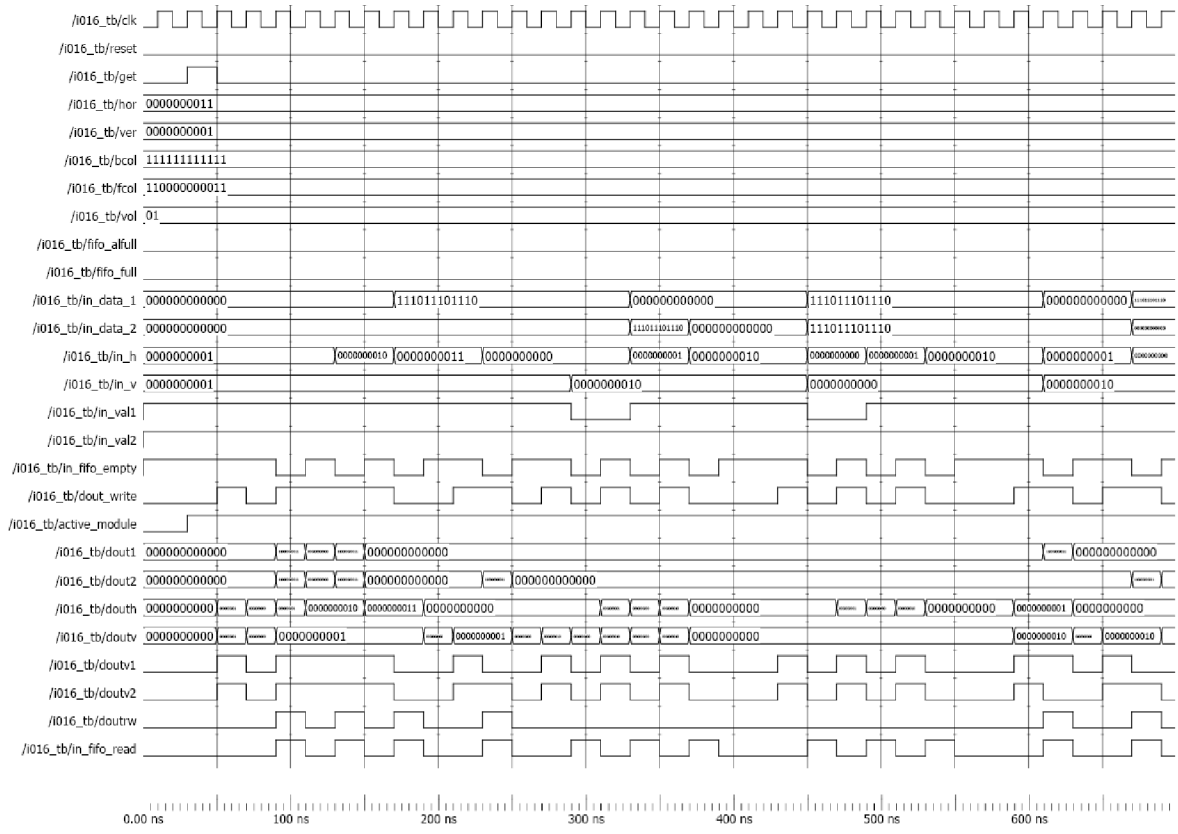
1) Vložení semínka do zásobníku
2) Vyjmutí semínka
3) Nalezení levé a pravé hranice na daném konkrétním řádku y a vykreslení těchto bodů
4) Na vyšší úsečce (stejné xové souřadnice jako v bodě 3, ale se souřadnicemi y+1) hledej souvislé vnitřní úseky z nichž vlož do zásobníku vždy souřadnice jednoho vnitřního bodu pro každou z nich.
5) To stejné jako v bodě 4 ale na souřadnicích y-1
6) Opakuj od bodu 2 dokud není zásobník prázdný

Průběh vyhodnocování je naznačen na následujícím obrázku:

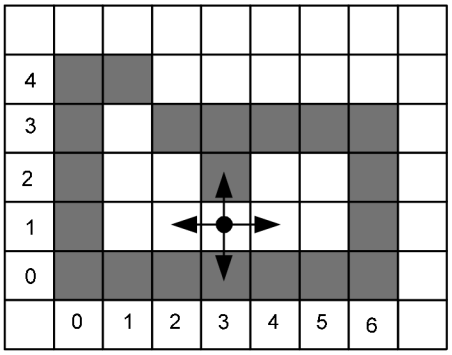


Obrázek 45 Algoritmus řádkového semínkového vyplňování

Na obrázku 46 je znázorněn průběh simulace pro vyplňování oblasti zobrazené na obrázku 47. Ze simulace pro přehlednost uvádím pouze první část, kdy se řádek y=1 od bodu x=3 vyplní, následuje zjišťování možných semínek ve vyšším a nižším řádku. Při čase 590 ns je aktivováno další semínko na souřadnicích [2,2].



Obrázek 46 Časový průběh simulace Instrukce 016 – semínkové vyplňování



Obrázek 47 Obrazec použitý pro testování Instrukce 016

8.9 Volání instrukcí ze strany instrukčního programu

V příloze 2 jsou zveřejněny syntaxe pro správné volání instrukcí. V horní části tabulky jsou označena čísla příchozích dat po 12ti bitech tak, v jakém pořadí je očekává dekodér. V případě použití interního programu a tedy vysílání instrukcí pomocí 90ti bitové informace, jsou označeny čísla bitů. Pokud je požadovaná informace o menší datové šířce, pak je LSB dané informace shodné s LSB daného 12ti bitového bloku. Např. informace o šířce čáry v instrukci 003 je 6ti bitová v bloku příchozích sériově-paralelních dat č.6 a konečné formě paralelní dat v bitech (23:12). Výsledná 12ti bitová informace na těchto bitech tedy bude mít hodnotu 000000111111 pro max. šířku čáry, tj. 64 bodů.

9 Dosažené rychlosti u jednotlivých modulů

Na základě syntézy v prostředí XILINX ISE lze stanovit následující rychlosti příslušných modulů. Tyto rychlosti jsou zveřejněny v tabulce 3.

blok / modul	dosažená rychlost	poznámka
CLK_MANAGER	376,932 MHz	
DECODER	166,744 MHz	
INT_PROGRAM	168,606 MHz	
GU_BLOCK	166,744 MHz	s procesorem bez aktivních modulů, vč. dekodéru
PROCESSOR	264,830 MHz	bez aktivních modulů instrukcí a prioritního FIFO buferu
PROCESSOR	62,956 MHz	aktivní moduly I001, I002 a I003
SRAM_CONTROLLER	108,897 MHz	
VGA_CTR	135,062 MHz	
I001	129,702 MHz	mazání obrazovky
I002	236,128 MHz	vykreslení pixelu
I003	83,250 MHz	vykreslení čáry
I006	66,011 MHz	vykreslení kruhu
I009	31,231 MHz	vykreslení elipsy
I011	119,048 MHz	kopírování v paměti
I016	74,068 MHz	výplň v paměti
I022	155,593 MHz	vykreslení ASCII znaků

Tabulka 3 Rychlost jednotlivých modulů

Jak je patrné z tabulky 3, nejkritičtějším blokem je modul pro generování elipsy. Jak již bylo napsáno v kapitole 8.5, tento modul je značně náročný na výpočet kvůli rozsahu proměnnách a tato náročnost se projevila i do celkové rychlosti modulu.

10 Závěr

Na základě zadání práce byl vytvořen komplexní program umožňující vykreslování grafických dat. Jako výstup byl zvolen VGA monitor s připojeným 4-bitovým převodníkem. V souladu se standardy VGA byl vytvořen SRAM kontrolér, který emuluje dvou portovou VideoRAM.

Grafika se vytváří na základě provádění instrukcí. Tyto instrukce lze zadávat jak v rámci implementace do cílového obvodu (interního programu), tak i prostřednictvím externího programu. Pro komunikaci s tímto externím programem byl navržen sériově-paralelní dekodér. Jakmile blok procesoru přijme instrukci, zadává ji příslušnému bloku ke zpracování.

Jako základní sada instrukcí byly vytvořeny bloky s programy pro instrukce mazání obrazovky, vykreslování jednotlivého obrazového bodu (pixelu), kreslení čar a odvozených obrazových prvků (obdélníků) a to vč. kontroly šířky čáry, kreslení kružnic vč. kontroly šířky obvodové čáry a odvozených obrazových prvků (kreslení oblouků), kreslení elipsy, kopírování ploch v rámci paměti, vyplňování oblastí v rámci paměti a vykreslování znaků z ASCII tabulky. Program je navržen tak, aby všechny potřebné instrukce bylo možné velmi jednoduše a velmi rychle zařadit do systému.

Koncepce programu je odvislá od použitého SPARTAN-3 Starter kitu a tím pádem použití páru SRAM pamětí. Z tohoto důvodu jsou horizontální souřadnice adresovány po párech. Při implementaci programu do jiného cílového obvodu by musela být tato vlastnost respektována a musely by být připojeny opět 2 SRAM paměti se společnou adresací.

Všechny bloky byly podrobeny simulaci, jejichž výsledky jsem uvedl v textu a v případě potřeby jsou simulační zdrojové kódy k dispozici na přiloženém CD. Stejně tak jsou na přiloženém CD k dispozici všechny vytvořené zdrojové kódy programů.

11 Seznam použitých zdrojů

[1] *Spartan-3 FPGA Starter Kit Board User Guide*. 2008th enl. edition. [s.l.] : [s.n.], 2008. 64 s. Dostupný z WWW:

<http://www.xilinx.com/support/documentation/boards_and_kits/ug130.pdf>.

[2] *Spartan-3 Generation FPGA User Guide*. 2008th enl. edition. [s.l.] : [s.n.], 2008. 520 s. Dostupný z WWW:

<http://www.xilinx.com/support/documentation/user_guides/ug331.pdf>.

[3] *IS61LV25616AL Datasheet*. [s.l.] : [s.n.], c2003. 17 s. Dostupný z WWW: <http://www.issi.com/pdf/61LV25616AL.pdf>.

[4] *Spartan-3A/3AN FPGA Starter Kit Board User Guide*. 2008th enl. edition. [s.l.] : [s.n.], 2008. 140 s. Dostupný z WWW:

<http://www.xilinx.com/support/documentation/boards_and_kits/ug334.pdf>.

[5] KOLOUCH, Jaromír. *Programovatelné logické obvody : Počítačové cvičení*. 2005. vyd. Brno : FEKT VUTBR, c2005. 94 s.

[6] PINKER, Jiří, POUPA, Martin. *Číslicové systémy a jazyk VHDL*. 2006. 1. vydání vyd. Praha : BEN, 2006. 352 s. ISBN 80-7300-198-5.

[7] HRADECKÝ, David. *VGA řadič na FPGA*. [s.l.], 2005. 28 s. Seminární práce. Dostupný z WWW: <http://amber.feld.cvut.cz/fpga/teaching/fpga/VGA_FPGA.pdf>.

[8] *VGA timing information* [online]. 1994-2007 [cit. 2008-12-17]. Dostupný z WWW: <http://www.epanorama.net/documents/pc/vga_timing.html>.

[9] *VGA Signal Timing* [online]. 2008 [cit. 2008-12-17]. Dostupný z WWW: <http://tinyvga.com/vga-timing>.

[10] *ASCII* [online]. 2004 , 30. 4. 2009 [cit. 2009-05-25]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/ASCII>.

[11] SOCHOR, Jiří, ŽÁRA, Jiří, BENEŠ, Bedřich. *Algoritmy počítačové grafiky*. Praha : Vydavatelství ČVUT, 1998. 184 s.

[12] SKALA, Václav. *Algoritmy počítačové grafiky*. Plzeň : Ediční středisko ZČU, 1992. 3 sv. (117, 167, 133 s.).

[13] SOBOTA, Branislav, MILIÁNOVÁ, Lucia, MILIÁN, Ján. *Grafické editory*. České Budějovice : KOPP, 1997. 239 s.

[14] ZÁMOŽÍK, Jozef, et al. *Základy počítačovej grafiky*. Bratislava : Slovenská technická univerzita v Bratislave, 1999. 265 s.

[15] *Digital Clock Manager (DCM) Module*. [s.l.] : [s.n.], 2009. 6 s. Dostupný z WWW: <http://www.xilinx.com/support/documentation/ip_documentation/dcm_module.pdf>.

[16] *LogiCORE™ IP FIFO Generator v5.1*. [s.l.] : [s.n.], 2009. 106 s. Dostupný z WWW: <http://www.xilinx.com/support/documentation/ip_documentation/fifo_generator_ug175.pdf>.

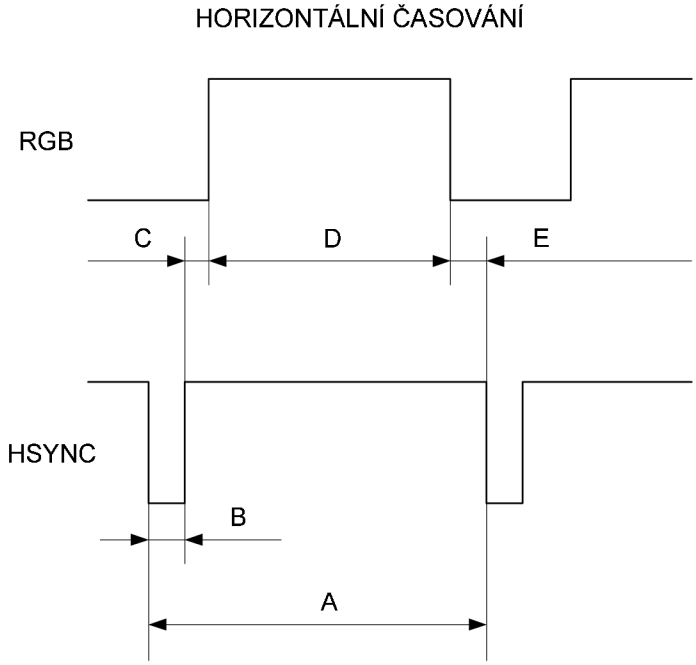
12 Seznam použitých zkratek a symbolů

ASCII	American Standard Code for Information Interchange
BP	back porch, čas. interval mezi konecm synch. impulzu a začátkem vykreslování
bufer	vyrovnávací paměť
CLK	clock, hodinový signál
CRT	Cathod Ray Tube
DDA	digitální diferenciální analyzér
DDC	Display Data Channel
DPS	deska plošných spojů
FP	front porch, čas. interval mezi koncem vykreslování a začátkem synch. impulzu
FPGA	Field-Programmable Gate Array
GU	graphics unit, generátor grafiky
HI	high, vysoká úroveň signálu
HOR	horizontální
HS	horizontální synchronizace
HSYNC	horizontální synchronizace
LO	low, nízká úroveň signálu
log. 0	nízká úroveň signálu
log. 1	vysoká úroveň signálu
LSB	Least Significant Bit
MSB	Most Significant Bit
ns	nano sekunda
pixel	obrazový bod
RGB	Red Green Blue
RST	reset
SynchRST	synchronní reset
VER	vertikální
VGA	Video Graphics Adapter
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VRAM	Video Random Access Memory
VS	vertikální synchronizace
VSYNC	vertikální synchronizace

13 Seznam příloh

Příloha 1 - Časování signálu

Popisování vlastností signálu lze rozdělit pro větší přehlednost do tří částí:



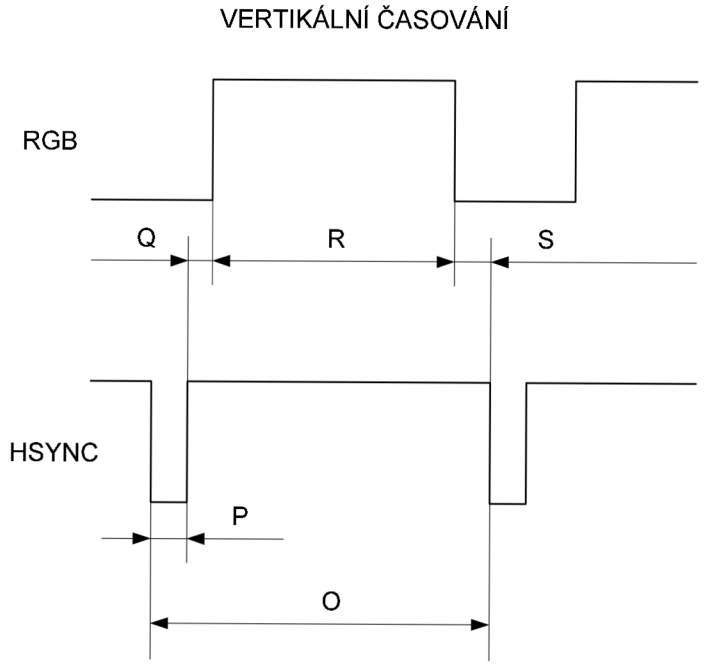
Obrázek 48 Horizontální časování – časový průběh

rozlišení	obnovovací frekvence	pixel clock MHz	F_HSYNC kHz	A		B	
				μs	pixelů	μs	pixelů
640x350	70Hz	25,4271	31,476	31,770	808	3,770	96
640x400	70Hz	25,4271	31,476	31,770	808	3,770	96
640x480	60Hz	25,1750	31,469	31,778	800	3,813	96
640x480	72Hz	31,5000	37,861	26,413	832	1,270	40
640x480	75Hz	31,5000	37,500	26,667	840	2,032	64
640x480	85Hz	36,0000	43,269	23,111	832	1,556	56
720x350	70Hz	28,3465	31,496	31,750	900	3,810	108
720x400	70Hz	28,3220	31,469	31,777	900	3,813	108
800x600	56Hz	36,0000	35,156	28,444	1024	2,000	72
800x600	60Hz	40,0000	37,879	26,400	1056	3,200	128
800x600	72Hz	50,0000	48,077	20,800	1040	2,400	120
800x600	75Hz	49,5000	46,875	21,333	1056	1,616	80
800x600	85Hz	56,2500	53,674	18,631	1048	1,138	64
1024x768	75Hz	78,7500	60,023	16,660	1312	1,219	96
1024x768	85Hz	94,5000	68,677	14,561	1376	1,016	96

Tabulka 4 Časování horizontálního běhu – 1. část

rozlišení	obnovovací frekvence	pixel clock MHz	C		D		E	
			μs	pixelů	μs	pixelů	μs	pixelů
640x350	70Hz	25,4271	1,890	48	25,170	640	0,940	24
640x400	70Hz	25,4271	1,890	48	25,170	640	0,940	24
640x480	60Hz	25,1750	1,907	48	25,422	640	0,636	16
640x480	72Hz	31,5000	3,968	125	20,508	646	0,667	21
640x480	75Hz	31,5000	3,810	120	20,317	640	0,508	16
640x480	85Hz	36,0000	2,222	80	17,778	640	1,558	56
720x350	70Hz	28,3465	1,799	51	25,612	726	0,529	15
720x400	70Hz	28,3220	1,907	54	25,422	720	0,636	18
800x600	56Hz	36,0000	3,472	125	22,389	806	0,583	21
800x600	60Hz	40,0000	2,125	85	20,150	806	0,925	37
800x600	72Hz	50,0000	1,220	61	16,120	806	1,060	53
800x600	75Hz	49,5000	3,232	160	16,162	800	0,323	16
800x600	85Hz	56,2500	2,702	152	14,222	800	0,589	32
1024x768	75Hz	78,7500	2,235	176	13,003	1024	0,203	16
1024x768	85Hz	94,5000	2,201	208	10,836	1024	0,508	48

Tabulka 5 Časování horizontálního běhu – 2. část



Obrázek 49 Vertikální časování – časový průběh

rozlišení	obnovovací frekvence	pixel clock	F_VSYNC	O		P	
		MHz	Hz	ms	řádků	ms	řádků
640x350	70Hz	25,4271	70,077	14,270	449	0,060	2
640x400	70Hz	25,4271	70,077	14,270	449	0,060	2
640x480	60Hz	25,1750	59,940	16,683	525	0,064	2
640x480	72Hz	31,5000	72,807	13,735	520	0,079	3
640x480	75Hz	31,5000	75,000	13,333	500	0,080	3
640x480	85Hz	36,0000	85,008	11,764	509	0,671	29
720x350	70Hz	28,3465	70,087	14,268	449	0,063	2
720x400	70Hz	28,3220	70,087	14,268	449	0,064	2
800x600	56Hz	36,0000	56,259	17,775	625	0,056	1
800x600	60Hz	40,0000	60,317	16,579	628	0,106	4
800x600	72Hz	50,0000	72,187	13,853	666	0,125	6
800x600	75Hz	49,5000	75,000	13,333	625	0,064	3
800x600	85Hz	56,2500	85,061	11,758	631	0,056	3
1024x768	75Hz	78,7500	75,029	13,328	800	0,050	3
1024x768	85Hz	94,5000	84,997	11,765	808	0,044	3

Tabulka 6 Časování vertikálního běhu – 1. část

rozlišení	obnovovací frekvence	pixel clock	Q		R		S	
		MHz	ms	řádků	ms	řádků	ms	řádků
640x350	70Hz	25,4271	1,880	60	11,13	350	1,2	37
640x400	70Hz	25,4271	1,080	34	12,72	400	0,41	13
640x480	60Hz	25,1750	1,048	33	15,253	480	0,318	10
640x480	72Hz	31,5000	0,686	26	12,782	484	0,184	7
640x480	75Hz	31,5000	0,427	16	12,8	480	0,027	1
640x480	85Hz	36,0000	0,578	25	11,093	480	0,023	1
720x350	70Hz	28,3465	1,811	57	11,25	354	1,144	36
720x400	70Hz	28,3220	1,080	34	12,711	400	0,413	13
800x600	56Hz	36,0000	0,568	20	17,177	604		-1
800x600	60Hz	40,0000	0,554	21	15,945	604		-1
800x600	72Hz	50,0000	0,436	21	12,263	604	0,728	35
800x600	75Hz	49,5000	0,448	21	12,8	600	0,021	1
800x600	85Hz	56,2500	0,503	27	11,179	600	0,019	1
1024x768	75Hz	78,7500	0,466	28	12,795	768	0,017	1
1024x768	85Hz	94,5000	0,524	36	11,183	768	0,015	1

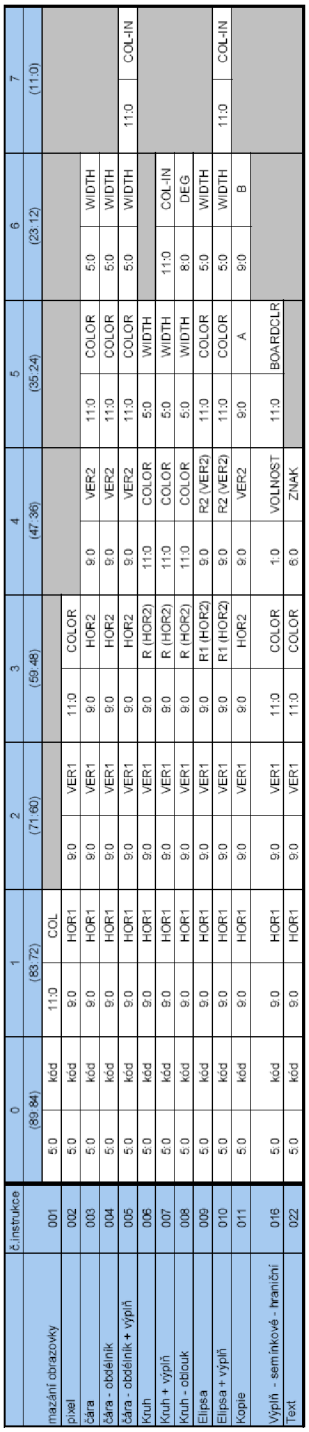
Tabulka 7 Časování vertikálního běhu – 2. část

Poslední částí signálu, kterou musíme popsat, je orientace synchronizačních signálů:

rozlišení	obnovovací frekvence	Polarita HSYNC	Polarita VSYNC
640x350	70Hz	neg	neg
640x400	70Hz	neg	neg
640x480	60Hz	neg	neg
640x480	72Hz	neg	neg
640x480	75Hz	neg	neg
640x480	85Hz	neg	neg
720x350	70Hz	neg	neg
720x400	70Hz	neg	pos
800x600	56Hz	neg	neg
800x600	60Hz	pos	pos
800x600	72Hz	pos	pos
800x600	75Hz	pos	pos
800x600	85Hz	pos	pos
1024x768	75Hz	pos	pos
1024x768	85Hz	pos	pos

Tabulka 8 Orientace synchronizačních signálů

Příloha 2 – Instrukční sada



mazání obrazovky	č. instrukce	0 (89:84)		1 (83:72)		2 (71:60)		3 (59:48)		4 (47:36)		5 (35:24)		6 (23:12)		7 (11:0)	
	001	5:0	kód	11:0													
pixel	002	5:0	kód	9:0	HOR1	9:0	VER1	11:0	COLOR								
čára	003	5:0	kód	9:0	HOR1	9:0	VER1	9:0	HOR2	9:0	VER2	11:0	COLOR	5:0	WIDTH		
čára - obdélník	004	5:0	kód	9:0	HOR1	9:0	VER1	9:0	HOR2	9:0	VER2	11:0	COLOR	5:0	WIDTH		
čára - obdélník + výplň	005	5:0	kód	9:0	HOR1	9:0	VER1	9:0	HOR2	9:0	VER2	11:0	COLOR	5:0	WIDTH	11:0	COL-IN
Kruh	006	5:0	kód	9:0	HOR1	9:0	VER1	9:0	R (HOR2)	11:0	COLOR	5:0	WIDTH				
Kruh + výplň	007	5:0	kód	9:0	HOR1	9:0	VER1	9:0	R (HOR2)	11:0	COLOR	5:0	WIDTH	11:0	COL-IN		
Kruh - oblouk	008	5:0	kód	9:0	HOR1	9:0	VER1	9:0	R (HOR2)	11:0	COLOR	5:0	WIDTH	8:0	DEG		
Elipsa	009	5:0	kód	9:0	HOR1	9:0	VER1	9:0	R1 (HOR2)	9:0	R2 (VER2)	11:0	COLOR	5:0	WIDTH		
Elipsa + výplň	010	5:0	kód	9:0	HOR1	9:0	VER1	9:0	R1 (HOR2)	9:0	R2 (VER2)	11:0	COLOR	5:0	WIDTH	11:0	COL-IN
Kopie	011	5:0	kód	9:0	HOR1	9:0	VER1	9:0	HOR2	9:0	VER2	9:0	A	9:0	B		
Výplň - semínkové - hraniční	016	5:0	kód	9:0	HOR1	9:0	VER1	11:0	COLOR	1:0	VOLNOST	11:0	BOARDCLR				
Text	022	5:0	kód	9:0	HOR1	9:0	VER1	11:0	COLOR	6:0	ZNAK						

Tabulka 9 Instrukční sada