České vysoké učení technické v Praze
Fakulta elektrotechnická

BAKALÁŘSKÁ PRÁCE

Knihovna pro FPGA vývojovou desku Altera-tPad

2013 MICHAL ŠVANDRLÍK

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Michal Švandrlík**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Knihovna pro FPGA vývojovou desku Altera-tPad**

Pokyny pro vypracování:

1. Prostudujte si materiály dodané k vývojové desce Altera tPad.
2. Navrhněte knihovny programů ve VHDL pro obsluhu vybraných součástí tPad.
3. Navržené knihovny ověřte pomocí sestavení vzorových úloh vhodných pro studenty předmětu Struktury počítačových systémů.

Seznam odborné literatury:

[1] Dokumentace k Altera tPad
[2] Pong P. Chu: RTL Hardware Design Using VHDL - Coding for Efficiency, Portability, and Scalability, Wiley 2006

Vedoucí: Ing. Richard Šusta, Ph.D.

Platnost zadání: do konce zimního semestru 2013/2014

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 20. 12. 2012

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, dne 13. 5. 2013 .....

.....
Podpis

Anotace

Bakalářská práce seznamuje s použitími dotykového displeje řízeného programovatelným hradlovým polem (FPGA). Dále popisuje moduly vytvořené pro usnadnění práce s dotykovým displejem ve cvičeních předmětu A0B35SPS (Struktury počítačových systémů), který se přednáší na ČVUT-FEL v bakalářské etapě studijního programu „Kybernetika a robotika“ a „Otevřená informatika“. Studenti pomocí modulů mohou řešit úlohy, jejichž některé návrhy budou obsaženy v závěru této práce.

Annotation

This bachelor work introduces into the usage of touch screens controlled by field programmable gate array (FPGA). It describes modules created for simplifying work with touch screens in laboratory exercises in course A0B35SPS (Struktury počítačových systémů) that is taught at Czech technical university – Faculty of electrical engineering in bachelor grade of “Cybernetics and robotics” and “Open informatics” program. Students can solve more complex exercises with the aid of these modules. Some examples of such exercises are contained at the end of this work.

Obsah

Seznam obrázků:

Seznam tabulek:

Kapitola 1

Úvod

V dnešní době vídáme dotykové displeje téměř na každém kroku. Chytré telefony by si většina lidí asi nedokázala bez dotykového displeje skoro představit. Vídáme je také na jiných zařízeních, na tabletech, na noteboocích či na informačních panelech, apod.

Tato bakalářská práce seznamuje se základním principem funkce dotykového displeje na odporovém principu a navrženými moduly k usnadnění jeho použití s programovatelným hradlovým polem (anglicky: „Field programmable gate array“, dále jen FPGA) na vývojové desce tPad od firmy Altera.

1.1. Deska Altera DE2-115 a tPad

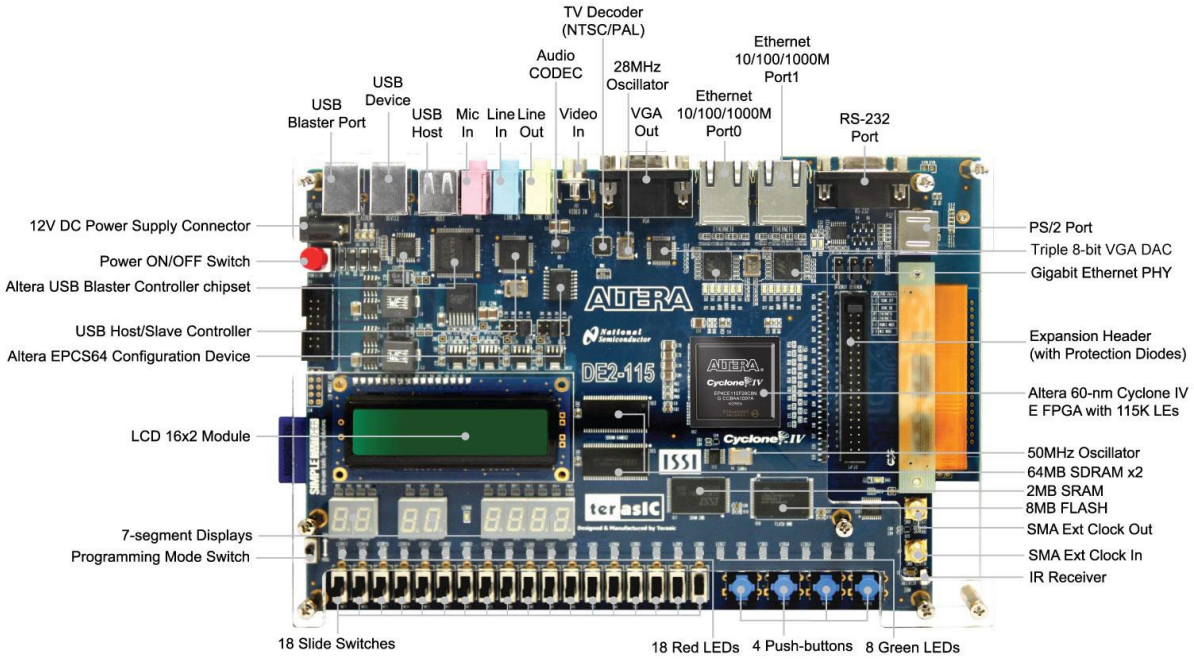
V úvodní kapitole se seznámíme s vývojovou deskou s FPGA s přídavným modulem tPad, pro které vytvoříme řídící moduly. Při vývoji a ladění digitálních logických obvodů se často využívají obvody FPGA, jejichž výrobou se zabývá mnoho firem. Jednou z nich je firma Altera [5]. Z jejího výrobního programu pochází mnoho vývojových desek různého výkonu, použitých periferií a ostatního vybavení.

Altera DE2-115 [1] (DE – Development and education board [5]), který se používá na cvičeních, je určen pro použití na univerzitách k výuce digitální logiky. Umožňuje tvorbu od jednoduchých logických obvodů, až po složitá komplexní řešení. Deska dovoluje programovat FPGA Cyclone® IV 4CE115 a je osazena nejen základními vstupními a výstupními periferiemi, ale i mnoha standardními periferiemi, častými na základních deskách PC.

Přehled hlavních periferií desky:

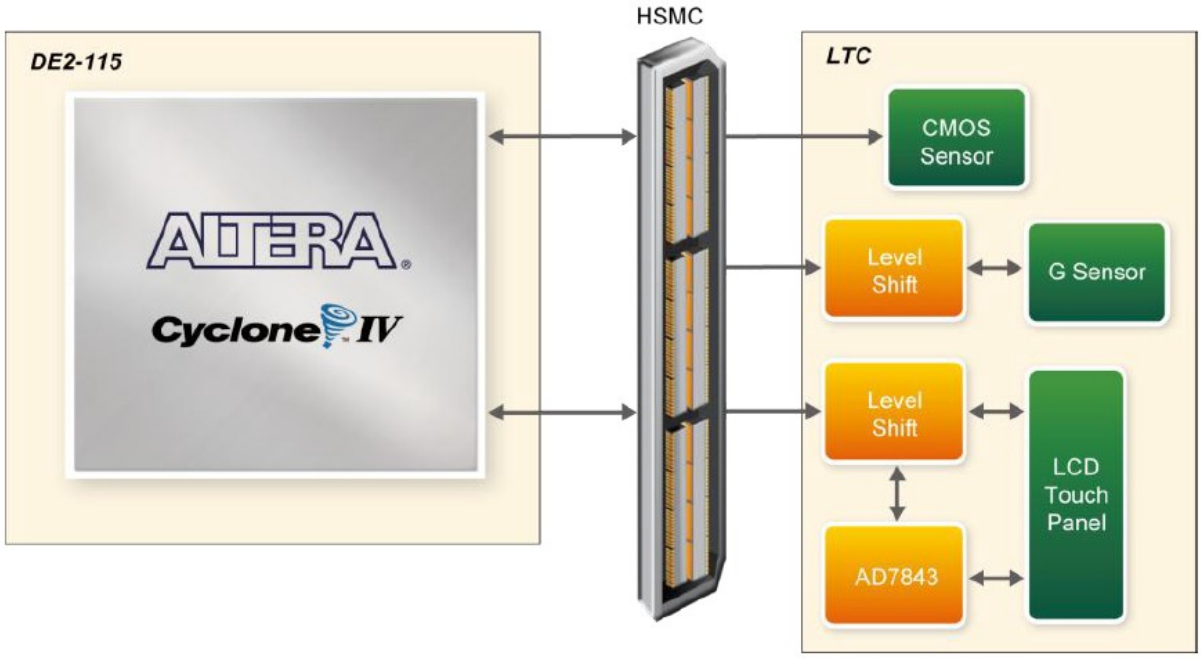
- Přepínače (18), tlačítka (4), LED červené (18) a zelené (9), sedmisegmentové zobrazovače (8)
- Paměť SRAM (2 MB), SDRAM (2x64 MB), Flash (8 MB)
- Oscilátory 50 MHz (3 kanály), SMA konektory pro vstup/výstup vnějšího zdroje hod. signálu

- USB Host (konektor typu A) a Slave (konektor typu B)
- Gigabitový Ethernet s konektorem RJ45 (2)
- VGA s 8bitovým trojitým DA převodníkem
- RS232 sériové rozhraní
- Slot pro SD kartu
- Konektor PS/2 pro připojení myši či klávesnice
- Další periferie můžete nalézt v uživatelské příručce k desce ([1]).



Obr. 1: Deska DE2-115 [1]

tPad [2] je doplňkový modul firmy Terasic rozšiřující desku DE2. S deskou je spojen přes rozhraní HSMC (High Speed Mezzanine Card) jehož detailní popis najdete v uživatelské příručce k desce DE2 [1].



Obr. 2: Konektor HSMC k připojení tPadu [2]

Modul tPad obsahuje následující periferie:

- Dotykový displej s úhlopříčkou 8 palců.
- Digitální kamera s rozlišením 5 Megapixelů.
- Tříosý digitální akcelerometr.

Obr. 3: Modul tPad [2]

Periferie desce propůjčují potenciál pro tvorbu komplexních systémů, jejichž příklady můžeme nalézt na přiložené SD kartě či CD. Složitost podobných úloh však překračuje rámec kurzu A0B35SPS. Slouží spíše pro demonstraci možností využití FPGA či pro velice hloubavé jedince. Hlubší detaily o desce DE2 a modulu tPad najdete v uživatelských příručkách (DE2 – [1], tPad – [2]).

Kapitola 2

Zobrazovací jednotka LCD

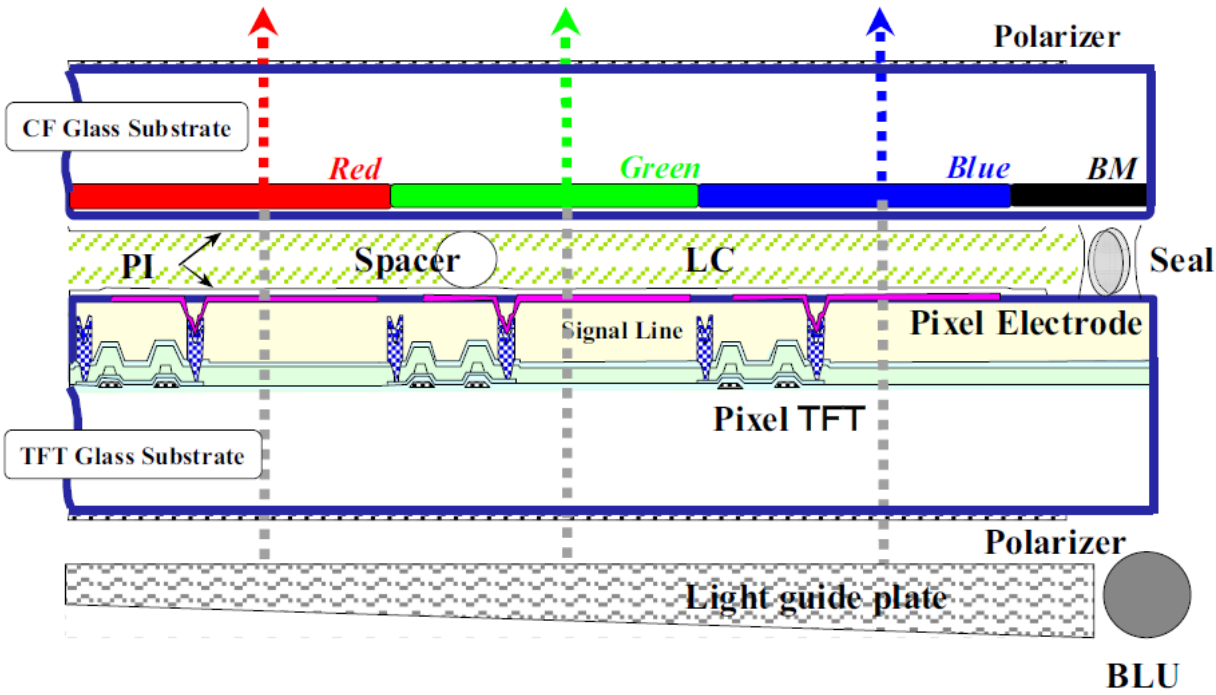
V této kapitole se seznámíme s technologií TFT-LCD zobrazovačů. Podíváme se blíže na řídící signály displeje, kterým je osazen tPad a na fázový závěs, důležitou část úpravy hodinového signálu. V poslední části se budeme věnovat našemu návrhu řídícího obvodu LCD displeje.

2.1. Základní vlastnosti LCD zobrazovače

TFT-LCD (Thin Film Transistor – Liquid Crystal Display) je displej na bázi tekutých krystalů, jehož jednotlivé body (pixely) jsou řízeny pomocí matice tenkovrstvých tranzistorů. Tekuté krystaly jsou uzavřeny mezi dvě vrstvy skla. Na nich jsou naneseny vrstvy ITO (Indium tin oxide). Blíže k pozorovateli je celistvá vrstva a slouží jako společná elektroda. Vrstva vzdálenější od něj je rozdělena podle jednotlivých subpixelů (každý pixel se skládá ze tří subpixelů červené, zelené a modré barvy). Jednotlivé subpixelové elektrody jsou připojeny pomocí TFT tranzistorů k řídícím vodičům.

Mezi elektrodami je nutné udržovat konstantní mezeru, a proto bývají mezi skla umístěny plastové kuličky, které spolehlivě vymezí vzdálenost obou elektrod. Obě skla obsahují polarizační filtry, které mají vzájemně kolmou osu polarizace. Když je mezi elektrodami nulové napětí, krystaly během průchodu světla otočí jeho osu polarizace o 90°, a tak světlo může projít oběma filtry. Pokud přiložíme k elektrodám napětí, krystaly začnou rovnat svoji strukturu podle elektrického pole. Světelný paprsek neprojde přes druhý filtr (má odlišnou osu polarizace) a pixel se jeví jako černý [3].

K podsvícení displeje se používají jedna či více světelných trubic s rozptylovači, které rozptýlí světlo po celé ploše displeje. V dnešní době spíše dominuje LED technologie. Panel je prosvícen po celé své ploše rozmístěnými LED. Hlavní výhoda LED podsvícení je konstantní svítivost v každém bodu displeje. Více o technologii výroby a konstrukci LCD displejů se můžeme dozvědět v příslušné literatuře (např. [3]).



Obr. 4: TFT-LCD [3]

2.2. LCD z modulu tPad, řídící signály

V katalogovém listu k LCD displeji ([4]) najdeme všechny potřebné údaje. Displej je vyroben technologií a-Si TFT (a-Si – amorphous silicon, více v [3]), má rozlišení 800x600 pixelů a jeho úhlopříčka činí 8 palců. O podsvícení displeje se stará LED technologie.

Řadič displeje je připojen k FPGA přímo pomocí konektoru typu HSMC. V uživatelské příručce k desce DE2 ([1]) je přiřazení pinů FPGA popsáno obecně pro konektor HSMC a podrobné popisy signálů lze najít v uživatelské příručce pro tPad ([2]), kde jsou piny pojmenovány podle názvu jednotlivých signálů. Pro potřeby dalšího výkladu uvádíme jen zjednodušenou tabulku signálů řadiče displeje:

Jméno signálu	**Popis**
LCD_DIM	Podsvícení displeje
LCD_NCLK	Vstupní hodinový signál
LCD_DEN	Data jsou platná
LCD_R[5..0]	Červená (6 bitů)
LCD_G[5..0]	Zelená (6 bitů)
LCD_B[5..0]	Modrá (6 bitů)

Tabulka 1: Signály LCD

Pro správnou funkci displeje musíme dodržet jeho časovací charakteristiku. Ta popisuje sled řídících signálů (reakce na sestupnou či náběžnou hranu hodinového signálu) a jejich požadované vlastnosti (frekvence či perioda, střída apod.). V katalogovém listu k displeji [4] najdeme kompletní časovací tabulku s minimálními, maximálními a typickými hodnotami signálů. Pro naši potřebu je tabulka dosti složitá, a proto uvádíme její zjednodušenou verzi, která nám postačí. V tabulce jsou použity pouze typické hodnoty. Položky ve sloupci „Symbol“ jsou přeznačené pro zjednodušení[1].

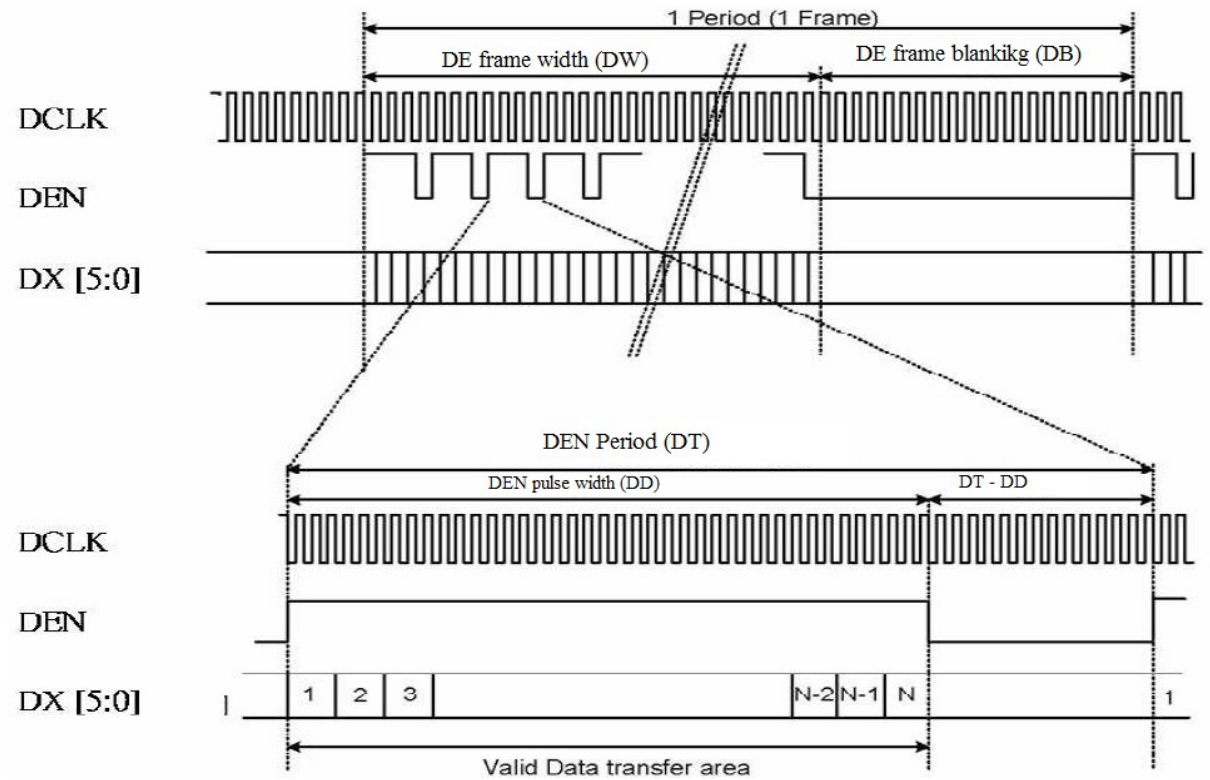
Parametr	**Symbol**	**Hodnota**	**Jednotka**
CLK frequency	-	39,79	MHz
CLK pulse duty	-	50	%
DEN period	DT	1056	CLK pulse
DEN pulse width	DD	800	CLK pulse
DEN frame blanking*	DB	28	DT
DEN frame width	DW	600	DT

Tabulka 2: Časovací tabulka

** frame blanking: počet „řádků“ navíc, slouží k synchronizaci*

Úvod do problematiky časování displeje by nebyl úplný, bez ukázky průběhu signálů. Schema průběhu je převzato z katalogového listu k displeji [4] a je přeznačeno, aby korespondovalo s tabulkou časovací charakteristiky.

1 Oproti [2], kde jsou symboly nekonzistentní, tudíž matoucí



Obr. 5: Průběhy signálů LCD [4]

Poslední informace potřebná pro správný chod LCD je, že řadič displeje čte data barev ze sběrnice s náběžnou hranou hodinového signálu. Pro úspěšné předání dat, musí být stálá jistou dobu před i po náběžné hraně. Nejvhodnější proto bude provádět změnu dat při sestupné hraně hodin.

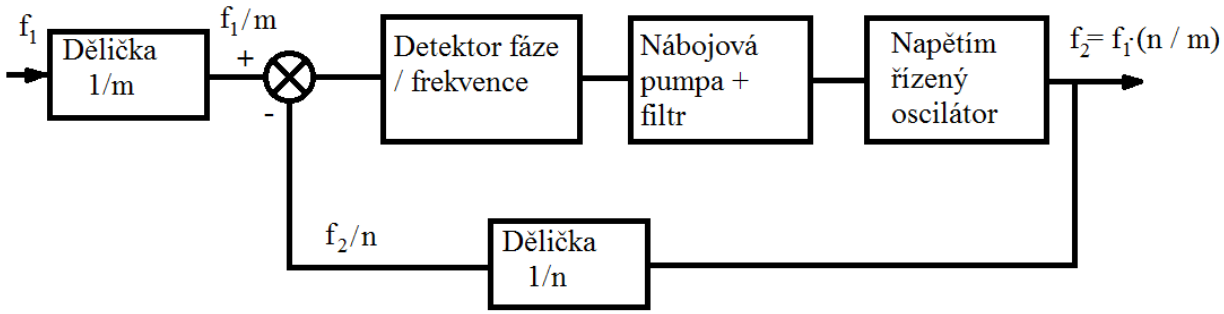
2.3. Fázový závěs

Fázový závěs (anglicky „Phase-locked loop“ - PLL [6]) má široké použití v mikroprocesorech a další digitální technice. Lze ho také využít k násobení frekvence. Schéma fázového závěsu lze vidět na Obr. 6. Fázový závěs se skládá z následujících hlavních bloků [6]:

- Detektor fáze/frekvence: snímá rozdíl fáze/frekvence příchozího signálu a signálu ze zpětné vazby. Vysílá signál, jehož polarita a velikost je úměrná fázovému/frekvenčnímu rozdílu obou signálů.
- Nábojová pumpa a filtr: Nábojová pumpa přijme signál od detektoru a vyšle proud odpovídající jeho velikosti. Filtr má dvě základní funkce. Zaprvé funguje jako

převodník proud-napětí a zadruhé pracuje jako dolní propust, která zablokuje vysoké frekvence.

- Napětím řízený oscilátor (anglicky: Voltage controlled oscillator - VCO): Frekvence oscilátoru je ovlivňována velikostí napětí, které přichází z výstupu filtru. Požadované frekvence je dosaženo, když je rozdíl fází/frekvencí, dosáhlo se požadované frekvence.
- Zpětnovazební dělička (1/n): Dělí výstupní frekvenci oscilátoru celým číslem n. Pokud je na vstupu detektoru detekován nulový fázový/frekvenční rozdíl, frekvence f_2 na výstupu oscilátoru je přesně $f_1 * n$.
- Vstupní a výstupní dělička: Rozšiřuje možnosti PLL. Používá se pokud chceme frekvenci násobit podílem čísel n/m.



Obr. 6: Fázový závěs

Kompletní problematika fázových závěsů je náročná a obsáhlá. Pro zájemce odkáži na literaturu (např. [6]), či přednášky předmětů zaměřených na procesorovou techniku (např. [13])

Kapitola 3

Dotykové ovládání

Nyní se seznámíme s technologií dotykových displejů. Popíšeme si jednotlivé typy displejů, jejich výhody a nevýhody, a nakonec se seznámíme s displejem kitu tPad.

3.1. Technologie dotykových displejů

Pro snímání dotyku z LCD displeje jsou používány převážně dvě technologie [9]:

- Resistivní
- Kapacitní

Resistivní technologie je založena na měření odporu dotykové obrazovky. Skládá se ze dvou tenkých vrstev ITO, které mají po celé ploše mezi sebou mezeru. Při dotyku dojde k deformaci plochy a vzájemnému dotyku obou vrstev. V tomto místě se mezi vrstvami utvoří napěťový dělič a podle naměřeného napětí se určí souřadnice dotyku. Tato technologie je velmi jednoduchá a napomáhá tak nízké ceně výsledného produktu. Nevýhodou je však malá přesnost a nemožnost používat více dotyková gesta [9], [11].

Kapacitní displej funguje na odlišném principu. Spočívá v měření změn kapacity v matici elektrod bez deformace vrstev ITO. Pohodlně může detekovat více dotyků najednou. Změna kapacity je však velmi malá a tudíž je kapacitní snímání náchylné k vnějšímu rušení. Tato nevýhoda je potlačována užitím speciálních vyhodnocovacích algoritmů [9]. Kapacitní displej je masově nasazován v chytrých telefonech, zatímco odporové displeje se již v nových výrobcích tolik nevyskytují.

tPad je osazen displejem resistivního typu. Jeho srdcem je 12bitový analogově-digitální převodník s postupnou aproximací AD7843 [11] od firmy Analog devices. Jak funguje převodník s postupnou aproximací se můžeme dozvědět například v literatuře či předmětech zaměřených na elektrická měření [10].

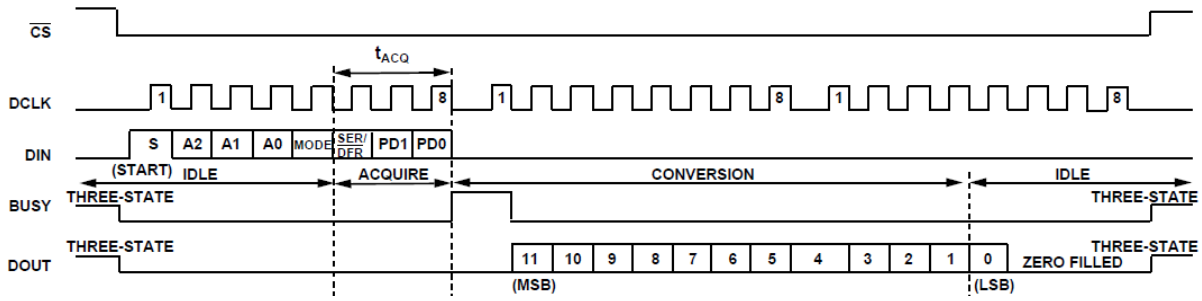
3.2. Převodník AD7843 a princip snímání

Dříve než zmíním problematiku komunikace s AD převodníkem, uvádím seznam signálů, pomocí nichž komunikace probíhá (podrobnosti v [2]):

Signál	Popis
TOUCH_PENIRQ_N	Přerušení od „pera“
TOUCH_DOUT	Sériová data výstup
TOUCH_BUSY	Sériové rozhraní zaneprázdněno
TOUCH_DIN	Sériová data vstup
TOUCH_CS_N	Výběr obvodu (chip select)
TOUCH_DCLK	Vstupní hodinový signál

Tabulka 3: Signály AD převodníku

Vzhledem k použití resistivní technologie snímání je převodník schopen měřit vždy jen jednu souřadnici. Převodník komunikuje přes sériová rozhraní. Sériovým vstupním rozhraním (TOUCH_DIN) přijímá řídící Byte, který nastaví režim měření. Po přijetí posledního bitu přejde signál TOUCH_BUSY do logické „1“. S další sestupnou hranou hodinového signálu se vrací zpět do logické „0“ a převodník začne vysílat data přes výstupní rozhraní (TOUCH_DOUT). Jako první se vysílá MSB („most significant bit“ – bit nejvyššího významu). Nejvyšší doporučená hodinová frekvence je 2 MHz. Podrobnosti o možnostech sériové komunikace nalezneme v katalogovém listu [11].



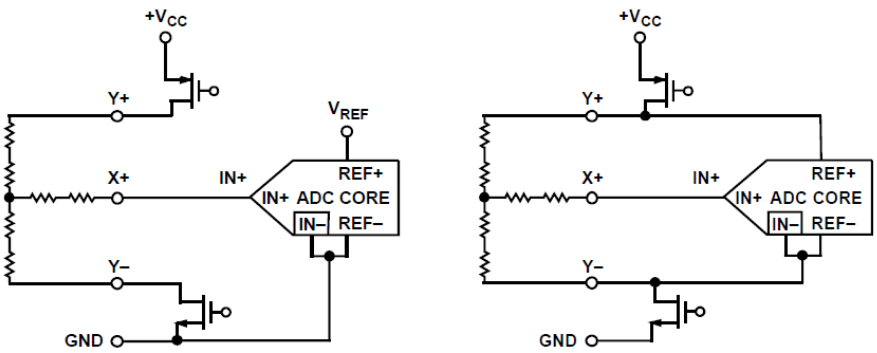
Obr. 7: Průběh komunikace s AD7843 [11]

Při velkém počtu možných nastavení převodníku považuji za vhodné se jim věnovat hlouběji. Funkci budeme odvozovat od jednotlivých bitů řídícího Bytu.

MSB							LSB
S	A2	A1	A0	MODE	SER/$\overline{\text{DFR}}$	PD1	PD0

Tabulka 4: Řídící Byte AD převodníku

- Bit S – Start bit, vždy logická „1“.
- Bity A[2..0] – „Adresové“ bity řídí přepínače vnitřního multiplexeru AD převodníku (nastavení reference apod., více viz [11]).
- Bit MODE – volba mezi 8bitovou (log. „1“) či 12bitovou (log. „0“) přesností.
- Bit SER/$\overline{\text{DFR}}$ – volba mezi „single-ended“ a „differential“ měřením viz Obr. 8, podrobnosti [11].
- Bity PD[1..0] – bity „power management“. Nastavují režim šetření energie.



Obr. 8: Zapojení měřících elektrod v režimu "single-ended" (vlevo) a "differential" (vpravo). Měření souřadnice y [11].

Pomocí adresových bitů a bitu SER/$\overline{\text{DFR}}$ se nastavuje připojení AD převodníku k displeji. Jak je vidět na Obr. 8, při zapojení v režimu „differential“ jsou referenční svorky převodníku přímo připojené na měřící elektrody displeje. Na rozdíl od zapojení „single-ended“ není měření ovlivňováno úbytky napětí na spínačích, tudíž budeme používat „differential“ měření (bit SER/$\overline{\text{DFR}}$ v log. „0“). Adresové bity nastavíme na hodnoty:

- Pro měření souřadnice X: A[2..0] = “101”.
- Pro měření souřadnice Y: A[2..0] = “001”.

Přesnost převodníku určuje bit MODE. 8bitový režim slouží převážně pro snížení spotřeby či zvýšení rychlosti vzorkování [11]. My budeme používat 12bitový režim (bit MODE v log. „0“). Bit PD1 řídí režim šetření energie a bit PD0 zapíná, či vypíná funkci přerušení při dotyku (signál TOUCH_PENIRQ_N). My použijeme nastavení PD[1..0] = “10” (šetření energie vypnuto, přerušení dotykem zapnuto). Více podrobností k nastavení AD převodníku v katalogovém listu [11].

3.3. Problémy a jejich řešení

Během tvorby jsme narazili na vzájemně související problémy způsobené odporovou technologií displeje:

- Nepodařilo se nám zprovoznit funkci „přerušení dotykem pera“.
- Při střídavém měření obou souřadnic vykazoval nestisknutý displej vysokou naměřenou hodnotu.

Nefungující přerušení dotykem pera jsme se rozhodli nahradit neustálým měřením. Při vývoji modulu, který začal měřením pouze jedné souřadnice, jsme zjistili, že nestisknutý displej vrací velmi malou hodnotu, která by se dala vyjmout z intervalu platných dat. Tento jev je vyvolán přivedením napětí na elektrody měřící vrstvy, která se při nestisknutém displeji chová jako kondenzátor. Náboj mezi jednotlivými vrstvami pak AD převodník změří a vzniká chyba.

Když vývoj dospěl do fáze, kdy se začaly měřit střídavě obě souřadnice, tato chyba vzrostla na hodnotu u obou souřadnic kolem 0x800, což je velmi blízko geometrického středu displeje. Taková hodnota se již nedala vyjmout z platných dat, neboť střed displeje je frekventované místo dotyku.

Nejprve jsme zkoušeli prodloužit periodu mezi jednotlivými měřeními prodloužením hlavní čítací smyčky, ovšem bez úspěchu. Druhý způsob byl zvýšit počet měření jedné souřadnice a potom přejít k druhé. Toto řešení se ukázalo být lepší. Pomocí 128 měření jedné souřadnice a poté druhé se nám podařilo snížit naměřenou hodnotu při nestisknutém displeji na zhruba 0x2C0. Tu je již možné vyloučit pomocí podmínky. Podařilo se nám takto eliminovat problém, ale výrazně jsme tím snížili vzorkovací frekvenci.

Kapitola 4

Návrhy modulů pro tPad

Následující kapitola se věnuje vlastnímu návrhu modulů pro Veškeré moduly budou napsány v jazyce VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) o kterém se můžeme dozvědět podrobnosti v literatuře [7], [8], [12].

4.1. Programování synchronní logiky

Programujeme tak, aby výsledný kód měl strukturu stavového automatu [8]. Náš kód se skládá vždy ze tří částí:

- Registr: Obsahuje proces („process“ [8]), který má ve svém „sensivity list“ [8] pouze hodinový signál, s jehož náběžnou či sestupnou hranou se aktualizují stavové proměnné. Sensivity list může obsahovat ještě asynchronní reset. Vstupní signál registru bývá označen jako „next“ a výstupní jako „registr“ (či jen „reg“).
- Logika následujícího stavu (next-state logic): Na základě stavových proměnných a vstupních hodnot vytváří vstup pro registr čili následující stav.
- Výstupní logika (output logic): Buď jen pomocí stavových proměnných (Moore state machine [7]), a nebo i vstupních hodnot (Mealy state machine [7]) ovlivňuje výstupy celého automatu.

Problematika stavových automatů je velmi rozsáhlá. Více informací lze najít v příslušné literatuře [7], [8].

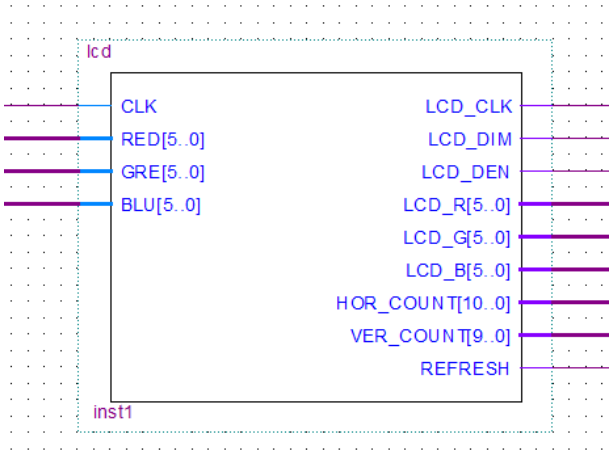
4.2. Vlastní návrh modulu pro řízení zobrazovače LCD

Nyní již máme všechny potřebné informace k vytvoření modulu zobrazovače LCD. Pro správný chod bloku potřebujeme hodinový signál o frekvenci požadované výrobcem. K jejímu dosažení využijeme fázový závěs. K dispozici máme 50 MHz oscilátor. Frekvence požadovaná řadičem displeje je 39,79 MHz (viz Tabulka 2). Fázový závěs vytvoříme podle návodu z [7], přednáška č. 8. Důležité je dát pozor na vstupní frekvenci, standardně je při tvorbě altpll

v programu Quartus 11.1 SP2 [5] nastavena na 100 MHz. Změníme ji na 50 MHz, které máme k dispozici. Výstupní frekvenci zvolíme požadovaných 39,79 MHz.

První je v kódu uvedena definice entity, která musí obsahovat všechny vstupy a výstupy bloku. Entita pro řízení LCD zobrazovače je nazvána „LCD“ a obsahuje následující vstupy/výstupy:

- CLK vstup pro hodinový signál (std_logic)
- RED, GRE, BLU – vstupy dat barev (vektory [8] po 6 bitech)
- výstupní signály pro řadič displeje viz Tabulka 1
- HOR_COUNT, VER_COUNT – výstupy odkazující na aktuálně zpracovávaný pixel (typ unsigned [8])
- REFRESH – výstup indikující konec snímku obrazovky.



Obr. 9: Instance modulu „LCD“

Jádrem našeho modulu jsou dva synchronní čítače (horizontální – sloupcový a vertikální – řádkový) určující zpracovávaný pixel. Podle hodnot z čítačů jsou nastavovány hodnoty výstupů řídících signálů. K realizaci čítačů používáme „signály“ (signal [8]) a nikoli proměnné (variable [8]), jelikož mohou činit problémy [8]. Pro čítače používáme signály typu unsigned (hor_reg, hor_next – 11 bitů, ver_reg, ver_next – 10 bitů). Označení odpovídá výše zavedenému (viz 4.1.).

Pro splnění nároků dané časovací tabulkou (Tabulka 2) se sloupcový čítač inkrementuje při každé sestupné hraně hodinového signálu. Při dosažení hodnoty 1055 se sloupcový čítač

nastaví zpět do nuly a řádkový čítač se inkrementuje. Řádkový čítač se nastaví do nuly společně se sloupcovým, když sloupcový má hodnotu 1055 a řádkový 627. Tím se celý cyklus opakuje a začíná se vysílat nový snímek obrazovky.

- Podsvícení displeje může zůstat neustále rozsvícené, tudíž výstup LCD_DIM zůstane trvale v logické „1“.
- Signál určující platnost dat LCD_DEN je v logické „1“, jen když jsou hodnoty čítačů ve viditelné oblasti displeje. Signál LCD_DEN se vrací do logické „0“ vždy po 800 hodinových pulzech v sloupcovém čítači a neaktivuje se v pozdějším, než 600 řádce.
- Data barev (LCD_R, LCD_G, LCD_B) jsou řízena signálem LCD_DEN. Pokud je LCD_DEN v logické „0“, vysílají se nulová data (černá) a pokud je v logické „1“, tak se na výstup předávají data ze vstupů (RED, GRE, BLU).
- Signál REFRESH slouží jako pomocný impulz k signalizaci konce snímku.

Je nutné si uvědomit, že signály hor_reg a ver_reg obsahují souřadnice právě zpracovávaného pixelu a blok, který bude řídit obrazová data, musí generovat data pro následující pixel. To je důležitá informace pro pozdější implementaci bloku spolupracujícího s blokem „LCD“. Vstupní hodinový signál CLK je připojen na výstupní LCD_NCLK. Zdrojový kód modulu „LCD“ je uveden v „Příloha A“.

4.3. Pomocné obvody pro komunikaci s AD převodníkem

Vzhledem k použití sériového rozhraní pro komunikaci s převodníkem a znalosti délky jednotlivých posílaných dat, můžeme s výhodou použít pro komunikaci posuvné registry. Ty můžeme vytvořit buď programem v jazyce VHDL či ve schematickém editoru. My jsme zvolili druhou možnost. Schémata najdeme v „Příloha B“. Vzhledem k velikosti obvodů uvádíme vždy jen část schématu (bitů). Logika zbylé části je stejná. Zde si oba moduly popíšeme.

4.3.1. 8bitový posuvný registr s paralelním vstupem

Posuvný registr je tvořen osmi klopnými obvody typu D [7] a 8 multiplexery (2 vstupy, 1 výstup). Ty obstarávají nastavování hodnot do registru. Na jeden vstup multiplexeru jsou

přiváděna data (DATA_IN[7..0]) a na druhý výstup předchozího obvodu D. Výstup posledního obvodu D je zároveň datovým výstupem z celého registru (S_OUT). Řídící vstupy multiplexerů jsou spojeny navzájem jako vstup LOAD_NMOVE. Pokud je na LOAD_NMOVE přivedena logická „0“, chová se obvod jako standardní posuvný registr a s každým hodinovým impulzem se data posunou o jeden bit vpřed. Pokud je v logické „1“, tak se na výstupy obvodů D nastavují bity z DATA_IN[7..0].

4.3.2. 12bitový posuvný registr se záchytným registrem [13]

Registr se skládá z dvanácti sériově řazených obvodů D (klasický 12bitový posuvný registr). Výstupy všech dvanácti bitů jsou přivedeny paralelně na dalších dvanáct obvodů typu D (záchytný registr), které mají za úkol přenést informaci na výstup DATA_OUT[11..0] až po naplnění celého registru. Hodinové vstupy obvodů záchytného registru jsou vyvedeny zvlášť na vstup SEND. Do posuvného registru vstupují data ze sériové linky vstupem S_IN. Nesmíme opomenout vyvést vstupy „enable“ (ENA), abychom mohli vyřadit registr z provozu při konverzi druhé souřadnice. Další možností by bylo přerušení hodinového signálu, ale to není doporučeno [8].

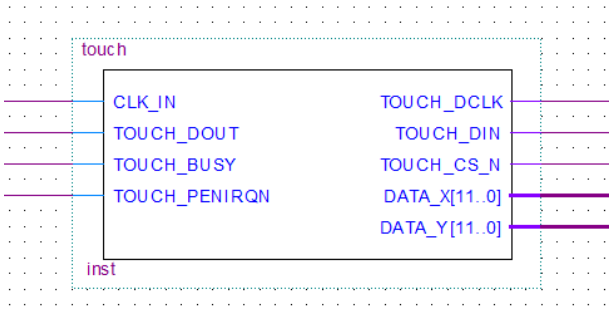
Na konec si necháme vygenerovat VHDL soubory, abychom viděli, jak vypadají deklarace entit jednotlivých bloků. Využijeme jich později při deklaraci komponentů (component [12]) v samotném modulu snímání dotyku.

4.4. Vlastní návrh modulu pro snímání dotyku z displeje

Opět jako první vytvoříme fázový závěs. Mohli bychom vytvořit nový, ale vzhledem k předpokládanému užití společně s blokem „LCD“, upravíme stávající (dostupný fázový závěs může generovat 5 nezávislých hodinových signálů).

Definice entity obsahuje vstupy a výstupy obvodu. Pojmenované jsou stejně jako v „Tabulka 3“ jen vstup zaměníme za výstup a opačně. Blok obsahuje navíc tyto signály:

- CLK_IN – Vstup hodinového signálu (std_logic).
- DATA_X[11..0], DATA_Y[11..0] – Datové výstupy (std_logic_vector).



Obr. 10: Instance modulu „TOUCH“

Před začátkem architektury si nadefinujeme typy komponentů, které využijeme při tvorbě bloku „TOUCH“ (každý pouze jedenkrát). Jednotlivé „instance“ komponentu již definujeme přímo v architektuře (v počtu kolik potřebujeme) spolu se signály, pomocí nichž bude instance komunikovat se zbytkem bloku. Detailnější seznámení s prací s komponenty je možné v literatuře ([7], [12]). V bloku „TOUCH“ používáme jednu instanci 8bitového registru a dvě instance 12bitového registru (pro každou osu jednu). Pro jednodušší orientaci v zapojení bloku jsme vytvořili náhradní schéma zapojení viz Obr. 16. Jména propojovacích signálů jsou shodná s návrhem bloku:

- BYTE_IN[7..0] – Řídící Byte, který je posílán řídícím blokem do 8bitového registru a z něj do AD převodníku.
- L_NM – Řízení režimu 8bitového posuvného registru (standardní režim či načítání dat ze vstupu DATA_IN[7..0]).
- WIRE_X, WIRE_Y – Aktivace 12bitových registrů pomocí jejich „enable“ vstupů.
- SEND_X, SEND_Y – Aktualizace obsahu záchytných registrů.

Hodinový vstup 8bitového registru je negován, jelikož AD převodník čte data ze sériové linky s náběžnou hranou hodinového signálu, tudíž registr musí měnit výstup se sestupnou hranou. Stavový registr používá následující signály:

- STATE[3..0] – stavový registr který řídí výstupní signály bloku a bloky posuvných registrů.
- COUNT[4..0] – čítač základní smyčky z jehož hodnoty se odvíjí hodnota vektoru STATE.

- VALID[8..0] – čítač vnější smyčky užité pro vícenásobné měření jedné souřadnice.

Synchronní chování bloku „TOUCH“ zajišťuje proces s hodinovým signálem ve svém „sensivity listu“. V něm se aktualizují čítače (COUNT, VALID) a stavový registr (STATE).

Část logiky následujícího stavu obsahuje veškeré časování komunikace s posuvnými registry a samotným převodníkem. Čítač COUNT se s každým hodinovým pulzem zvýší o 1. Jeho nastavení do nulové hodnoty je zajištěno přetečením jeho 5bitového registru. Čítač VALID se inkrementuje při každém nulování čítače COUNT. Jeho nulování je též zajištěno přetečením registru. Stavový registr STATE si rozebereme po jednotlivých bitech:

- STATE_REG(0) – odpovídá signálu TOUCH_CS_N. Je v logické „0“ s výjimkou posledního stavu čítače COUNT (tedy COUNT_REG = "11111").
- STATE_REG(1) – zajišťuje aktivaci 12bitového registru pro příjem dat v daném časovém intervalu.
- STATE_REG(2) – řídí režim 8bitového registru (L_NM). Ve stavu logické „0“ se nachází mezi prvním a osmým hodinovým pulzem základní smyčky.
- STATE_REG(3) – krátký impulz po přenesení dat z převodníku do 12bitového registru zajistí průchod dat přes jeho záchytný registr (SEND).

Ve výstupní logice jsou kombinací čítače VALID a stavového registru STATE utvářeny výstupy bloku a řídící signály posuvných registrů. Nejvyšší bit čítače VALID slouží k výběru měřené souřadnice. Pokud je roven logické „0“, měří se souřadnice x, a pokud je roven logické „1“, tak se měří souřadnice y (BYTE_IN).

Signály WIRE_X/WIRE_Y jsou připojeny k signálu STATE_REG(1), když čítač VALID dosáhne hodnoty "01111111" pro osu x resp. "11111111" pro osu y. Registrem je tudíž přijata pouze poslední naměřená hodnota každé souřadnice. Obdobně je propojen signál SEND_X/SEND_Y se signálem STATE_REG(3). Signál STATE_REG(0) je přesměrován na výstup TOUCH_CS_N a STATE_REG(2) na výstup L_NM.

Vstupy bloku TOUCH_PENIRQ_N a TOUCH_BUSY zůstaly nezapojené. Vstup BUSY nebyl třeba, neboť chování AD převodníku je deterministické a je tudíž nadbytečný. Zdrojový kód modulu „TOUCH“ je uveden v „Příloha C“.

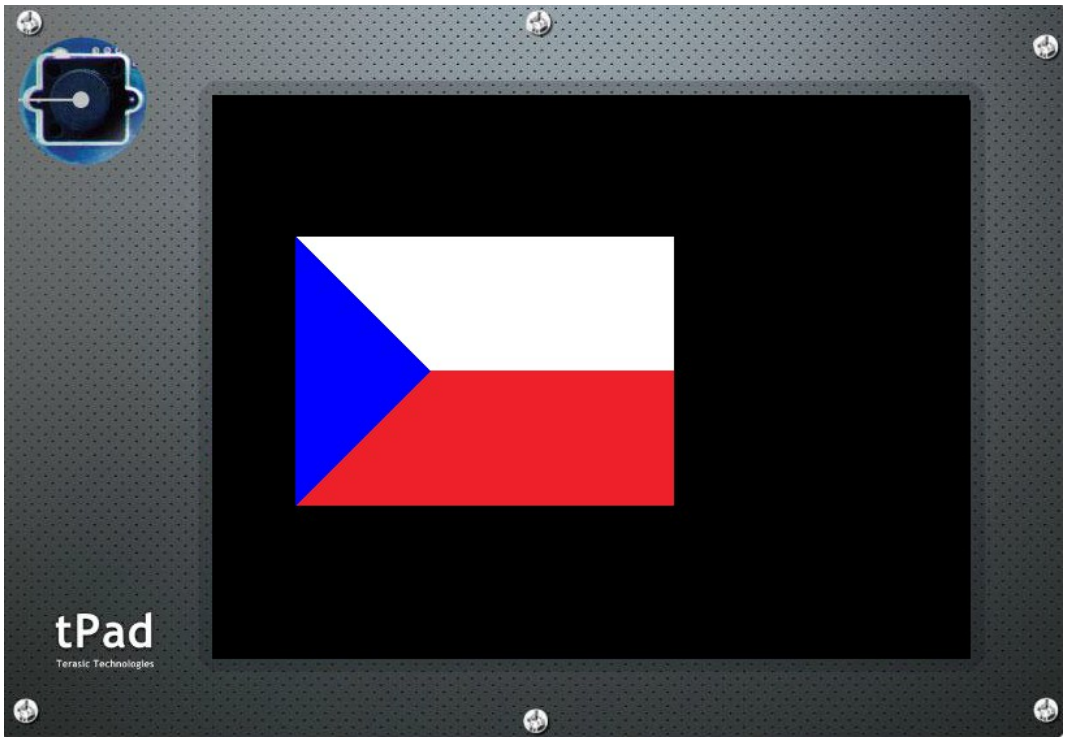
Kapitola 5

Vzorové úlohy

Kapitola se zabývá návrhem úloh, které využijí navržené moduly a prověří jejich funkčnost. Úlohy budou použity při výuce předmětu „Struktury počítačových systémů“.

5.1.1. Zadání úlohy: Vlajka

Pomocí jazyka VHDL vytvořte ve vývojovém prostředí Quartus II zobrazení vlajky na vestavěném dotykovém displeji, který obsahuje vývojová deska Altera DE2-115 tPad. Rozměry vlajky budou 400x300 pixelů, což odpovídá ¼ plochy obrazovky tPadu s rozlišením 800x600 pixelů. Vlajka se bude na displeji pohybovat. Její rychlost bude řízena přepínači na desce a bude se odrážet od stěn displeje. Ke zpracování využijte modul zobrazovací jednotky vytvořeného pro tPad (modul „LCD“), který bude umístěn na stránkách předmětu A0B35SPS (viz https://moodle.dce.fel.cvut.cz/course/view.php?id=5).



Obr. 11: Zobrazení vlajky na tPadu

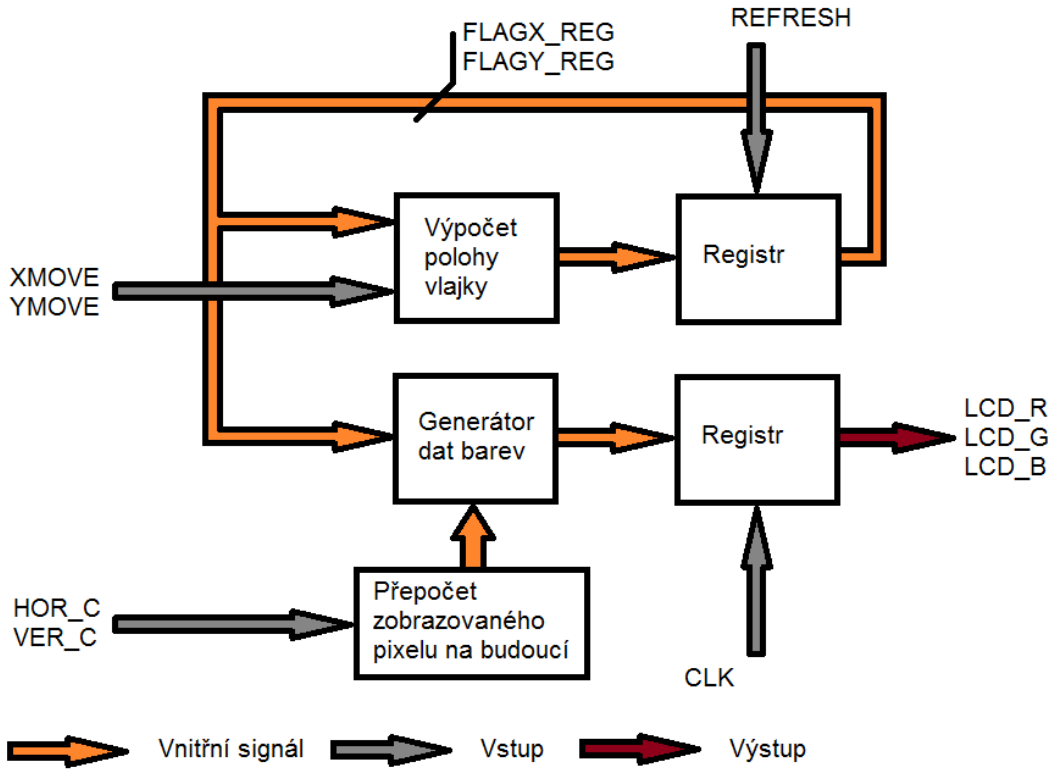
5.1.2. Návod

Řídící blok vlajky spolupracuje s modulem „LCD“. Modul „LCD“ posílá obvodu displeje data, která obdrží na svých vstupech, když hodnoty jeho horizontálního a vertikálního čítače odpovídají viditelné oblasti displeje. Hodnoty čítačů jsou zároveň vysílány na výstup modulu, takže uživatel má přehled o tom, který pixel se právě zpracovává. To je potřeba mít na paměti pokud chceme řádně určit souřadnice pixelu, pro který chceme generovat data. Hodnoty čítačů bude potřeba pozměnit. Blok musí obsahovat následující vstupy a výstupy.

- CLK – vstupní hodinový signál.
- XMOVE, YMOVE – vstupní signály určující rychlost pohybu vlajky (std_logic_vector).
- HOR_COUNT, VER_COUNT – souřadnice právě zobrazovaného pixelu od modulu „LCD“ (vstup – unsigned).
- REFRESH – vstupní pomocný signál z bloku „LCD“ (std_logic).
- LCD_R, LCD_G, LCD_B – výstupy dat barev (std_logic_vector).

Polohu vlajky je nutné měnit s každým novým obrazovým snímkem. Tím bude zajištěno, že se poloha vlajky nezmění během vykreslování. Doporučujeme tedy všechny signály spojené s pohybem vlajky obnovovat pouze s novým obrazovým snímkem, k čemuž nám dobře poslouží signál REFRESH vysílaný modulem „LCD“. Vstupy XMOVE a YMOVE budou připojeny na přepínače desky, kde se bude nastavovat rychlost pohybu vlajky v jednotlivých souřadnicích.

Je důležité si uvědomit, že LCD displej čte data barev ze sběrnice s náběžnou hranou hodinového signálu, tudíž bude vhodné data vysílat se sestupnou hranou, aby měla čas se ustálit. Buď hodinový signál budeme negovat, či bude proces aktualizující data barev řízen sestupnou hranou. Hodinový signál musí mít stejnou frekvenci a fázi jako hodinový signál přiváděný na vstup modulu „LCD“. Pokud by neměl, synchronizace obou bloků by nebyla možná. Předpokládaná struktura výsledného kódu je ukázána na Obr. 12.



Obr. 12: Blokové schéma úlohy „Vlajka“

5.1.3. Řešení

Entita řídícího bloku obsahuje vstupy a výstupy uvedené výše. Na ukázku zvolíme českou vlajku. K realizaci budeme potřebovat následující signály stavového automatu:

- RED, GRE, BLU – data barev (vše unsigned[5..0]).
- FLAGX, FLAGY – souřadnice levého horního rohu vlajky (unsigned).
- RIGHT, DOWN – pomocné signály pro uchování informace o směru pohybu (std_logic).

Pro získání správné informace o pixelu, pro který budeme generovat data, vytvoříme pro jeho souřadnice signály:

- HOR, VER – souřadnice budoucího promítaného pixelu.

Proces, který aktualizuje barevná data (RED, GRE, BLU) vysílaná do modulu „LCD“ a dále do řadiče displeje, má ve svém „sensitivity listu“ hodinový signál CLK. Obnova souřadnic polohy vlajky se nachází v samostatném procesu, který má v „sensitivity listu“ signál

REFRESH. Při pohybu se bude vlajka znovu vykreslovat s každým novým snímkem.

Polohu vlajky budeme měnit pomocí přičítání či odčítání hodnot ze vstupů XMOVE, YMOVE od aktuální hodnoty registru FLAGX, FLAGY. Správnou početní operaci zajistí signály RIGHT a DOWN, které zachovávají informaci o směru pohybu. Jejich hodnoty budeme měnit v moment, když se vlajka dotkne okraje. Může nastat situace, že hodnota např. XMOVE bude větší než FLAGX. Musíme ji vyřešit přidáním dodatečné podmínky, jinak by mohla nastat chyba.

Data barev se odvozují od souřadnic budoucího zobrazovaného pixelu, od pozice vlajky na displeji a její velikosti. Jelikož blok „LCD“ vysílá souřadnice právě zpracovávaného pixelu, musíme jejich hodnotu upravit, aby odpovídala té budoucí. Barvy se vysílají následujícím způsobem:

- V bílém pruhu se vysílají všechny barvy (všechny s hodnotou "111111").
- V červeném pruhu se vysílá pouze červená, ostatní jsou nulové (red: "111111", gre a blu: "000000").
- V modrém klínu je vysílána pouze modrá (blu: "111111", gre a red: "000000").

Modrý klín je poněkud užší, neboť jeho šířka je odvozena od pozice vlajky a jejího řádku[2] pomocí sčítání/odčítání celých čísel. Nelze tudíž vytvořit klín přesně do poloviny vlajky. Zbytek plochy displeje vysíláme černou barvu (všechny barvy "000000").

2 Myšlen je řádek zobrazované vlajky, tudíž v rozmezí 0 – 299

5.2.1. Zadání úlohy: Vlajka ovládaná dotykem na displeji

Pomocí jazyka VHDL vytvořte ve vývojovém prostředí Quartus II zobrazení vlajky na vestavěném dotykovém displeji, který obsahuje vývojová deska Altera DE2-115 tPad. Rozměry vlajky zvolte 400x300 pixelů, což odpovídá ¼ plochy obrazovky tPadu s rozlišením 800x600 pixelů. Rychlost vlajky a směr pohybu se bude odvozovat od rychlosti tahu prstu po displeji. Pohyb vlajky naprogramujte tak, aby byla stále celá viditelná a při dosažení okraje displeje se odrazila, tj změnila směr v dané ose na opačný. Ke zpracování využijte modul zobrazovací jednotky vytvořeného pro tPad (modul „LCD“) a modul pro snímání dotyku z dotykového displeje (modul „TOUCH“). Oba moduly najdete na stránkách předmětu A0B35SPS (viz https://moodle.dce.fel.cvut.cz/course/view.php?id=5).

5.2.2. Návod

Úlohu lze vyřešit modifikací té předchozí. Vyjdeme z toho, že máme již hotovou vlajku, která se pohybuje a její rychlost je řízena přepínači na desce. Rychlost pohybu vlajky bude možné určovat například podle rozdílu dvou posledních naměřených platných bodů na dotykovém displeji.

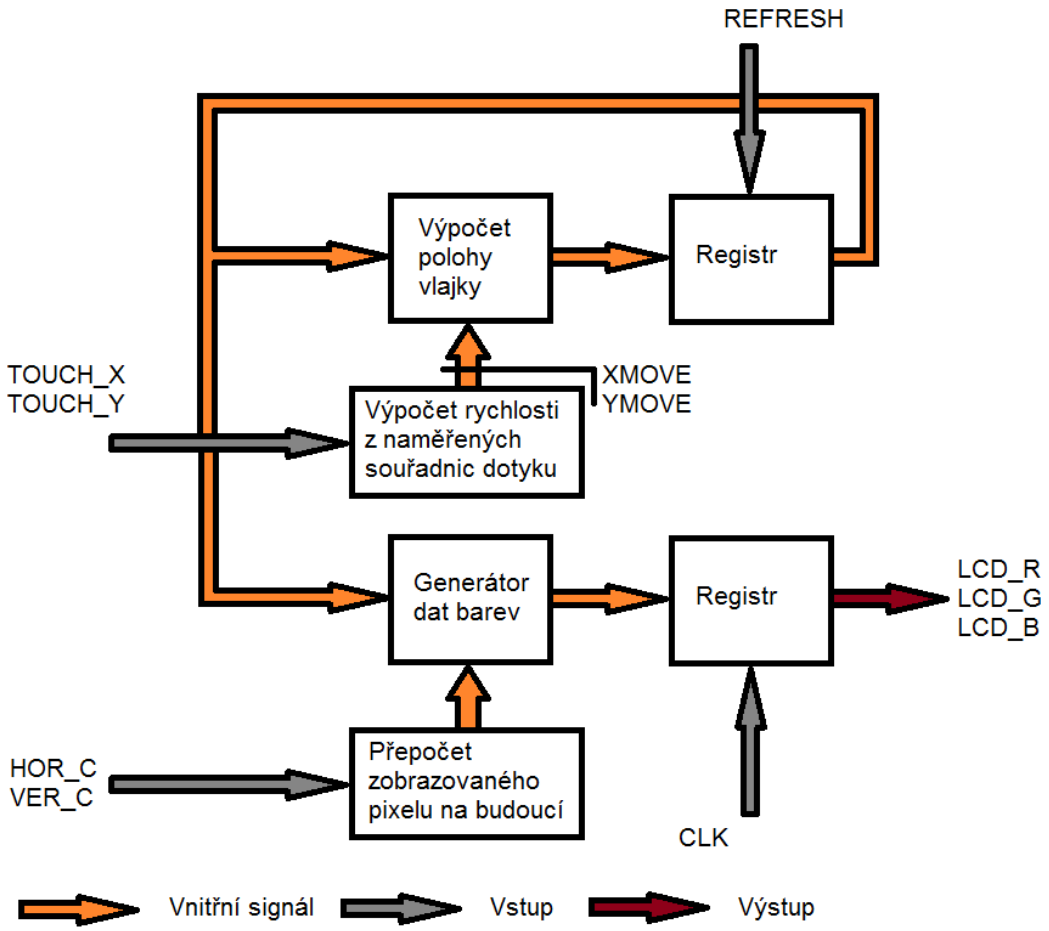
Budeme využívat souřadnic dotyku zasílaných modulem „TOUCH“. Dotykový displej však vykazuje nenulovou hodnotu při nestisknutém displeji (tu lze zjistit připojením výstupů z modulu na sedmisegmentové zobrazovače). Hodnoty souřadnic při nestisknutém displeji je nutné odfiltrovat dodatečnou podmínkou. Pro správné propojení s moduly „LCD“ a „TOUCH“ musí entita obsahovat tyto vstupy a výstupy:

- CLK – vstupní hodinový signál.
- TOUCH_X, TOUCH_Y – souřadnice dotyku na displeji.
- HOR_COUNT, VER_COUNT – souřadnice právě zobrazovaného pixelu (vstup).
- REFRESH – vstupní pomocný signál z bloku „LCD“.
- LCD_R, LCD_G, LCD_B – výstupy dat barev.

Je nutné si uvědomit, že tahem prstu po displeji se bude určovat i směr pohybu vlajky. Nezapomeňte tudíž upravit nejen logiku rychlosti pohybu, ale i logiku směru. Také

doporučujeme useknout několik LSB z naměřených hodnot přicházejících z bloku „TOUCH“, jelikož by způsobovali šum a příliš vysokou rychlost pohybu vlajky. Předpokládaná struktura kódu je naznačena na Obr. 13.

Pro úspěšné zkompilování modulu „TOUCH“ je nutné mít v cílové složce také soubory s bloky 8bitového a 12bitového registru, které naleznete spolu s modulem.



Obr. 13: Blokové schéma úlohy „Vlajka s dotykovým ovládáním“

5.2.3. Řešení

Řešení úlohy spočívá v modifikaci předchozí úlohy „Vlajka“. Pro větší přehlednost můžeme změnit názvy vstupů z XMOVE na TOUCH_X a YMOVE na TOUCH_Y. Stávajícímu návrhu přidáme do procesu řízení polohy následující stavové signály:

- XMOVE, YMOVE – Signály uchovávající rychlost pohybu vlajky (unsigned).
- XCO, YCO – Signály souřadnic posledního dotyku (unsigned).
- XPREV, YPREV – Signály souřadnic předchozího dotyku (unsigned).

Všechny tyto signály musí mít shodnou datovou šířku. Do procesu řízení barev přidáme následující signál:

- TOUCH – Signál indikace platnosti dotyku (std_logic).

Nyní si popíšeme stavy jednotlivých signálů a také modifikaci logiky již používané v předchozí úloze.

- Signál TOUCH je v logické „1“ pouze tehdy, když souřadnice dotyku přicházející od modulu „TOUCH“ budou splňovat omezující podmínku pro platná data, jinak je signál TOUCH v logické „0“.
- Pokud je signál TOUCH v logické „1“, tak se s náběžnou hranou signálu REFRESH přesune hodnota registrů XCO a YCO do registrů XPREV a YPREV. Zároveň se aktualizují hodnoty XCO a YCO na hodnoty vstupů TOUCH_X, TOUCH_Y, které jsou nejprve převedeny na datový typ unsigned a popřípadě zkráceny o několik jejich LSB.
- Z hodnot aktuálních a předchozích naměřených souřadnic se vypočítá rychlost pohybu vlajky (signály XMOVE, YMOVE) v jednotlivých směrech a to jejich odečtením. Pro výpočet polohy vlajky z aktuálních souřadnic a rychlosti platí stejné požadavky jako v první úloze.
- Kromě velikosti rychlosti ještě určujeme směr pohybu (signály DOWN, RIGHT). Pokud se prst dotýká displeje tak směr získáme porovnáním aktuální a předchozí souřadnice dotyku. Když není zaznamenán dotyk na displeji, směr se mění vždy při překročení hranice, jako v předchozí úloze.

Zdrojový kód vyřešené úlohy „Vlajka ovládaná dotykem na displeji“ je uveden v Příloha D. Úloha „Vlajka“ lze vytvořit drobnou degradací tohoto kódu.

5.3. Další využití

Moduly mohou být využity pro celou řadu dalších úloh, které již nejsou součástí této bakalářské práce. Modul „LCD“ může být využit k jakémukoli zobrazování na obrazovce tPadu, kde bude využito RGB kódování barev. Pomocí modulu „TOUCH“ by šel realizovat například zámek odemykaný kombinací pohybů (tuto funkci mají některé Smartphony), či další úlohy nevyžadující dotyk více prstů současně.

Kapitola 6

Závěr

V rámci této bakalářské práce jsme vytvořili dva moduly pro usnadnění práce s dotykovým displejem na desce tPad Altera DE2-115.

- Modul „LCD“ – obstarává komunikaci s řadičem LCD displeje, zajišťuje synchronizaci snímků. Vysílá data barev ve formátu RGB přicházející na jeho vstup dále do řadiče. Pro účely ostatních obvodů vysílá souřadnice zpracovávaného pixelu. Použitelné pro veškeré aplikace na desce tPad, které potřebují obrazový výstup.
- Modul „TOUCH“ – měří souřadnice dotyku na displeji s 12bitovou přesností v obou osách. Informace o naměřených souřadnicích jsou uchovávány v záchytných registrech. Při nestisknutém displeji vrací nenulovou hodnotu, kterou lze vyjmout z intervalu platných dat. Použití v širokém spektru aplikací, které nevyžadují více dotyků současně.

Výhodou těchto modulů je snadné začlenění do dalších projektů, jelikož nevyžadují žádnou zvláštní obsluhu.

Dále byla za pomocí těchto modulů vytvořena úloha „Vlajka“, která může být zadána ve dvou různých stupních obtížnosti.

- Pohybující se vlajka s ovládáním rychlosti pomocí přepínačů na desce.
- Pohybující se vlajka s ovládáním rychlosti pomocí dotykového displeje.

Výhodou použití zobrazení vlajky bylo množství rozdílných zadání, takže každá skupina může mít jiné, což pomáhá proti vzájemnému opisování.

Pro reprezentaci čísel byl použit výhradně datový typ „unsigned“ a to hlavně kvůli jeho snadné obvodové implementaci. Nevýhodou jeho použití byla některá omezení, která se musela řešit dodatečnými podmínkami zmíněnými v části věnované popisu úloh.

Moduly mají GNU General public license (GPL), tudíž volně šiřitelná s podmínkou, že veškerá odvozená díla od kódu pod GPL musí být též pod GPL (viz http://www.gnugpl.cz/). Firma Altera sice dodává ukázkové moduly v úlohách, ty jsou však obvykle chráněny licencí a soubory zakódovány. Moduly, které jsme vytvořili, mohou být využity nejen pro výuku v předmětu „Struktury počítačových systémů“ na ČVUT, ale i na jiných školách a každým, kdo splní GPL podmínku.

Seznam použité literatury

[1] DE2-115 User manual, Terasic technologies, přiložený na CD dodávaným s deskou

[2] tPad User manual, Terasic technologies, přiložený na CD dodávaným s deskou

[3] Yasuhiro Ukai, TFT-LCD Manufacturing Technology – Current Status and Future Prospect, International Workshop on Physics of Semiconductor Devices, 2007. IWPSD 2007

[4] Specifications for LCD module AM-800600GTMQW-T00H, Ampire co., ltd., přiložený na CD dodávaným s deskou

[5] Stránky výrobce desky DE2, www.altera.com

[6] D. DiClemente, F. Yuan, An Area-Efficient CMOS Current-Mode Phase-Locked Loop, 49th IEEE International Midwest Symposium on Circuits and Systems, 2006. MWSCAS '06

[7] Richard Šusta, přednášky předmětu A0B35SPS, FEL – ČVUT, 2012

[8] Pong P. Chu, RTL hardware design using WHDL, A Wiley-Interscience publication, 2006

[9] Chih-Lung Lin, *Member, IEEE*, Chia-Sheng Li, Yi-Ming Chang, Tsung-Chih Lin, *Senior Member, IEEE*, Jiann-Fuh Chen, and U.-Chen Lin, Pressure Sensitive Stylus and Algorithm for Touchscreen Panel, JOURNAL OF DISPLAY TECHNOLOGY, VOL. 9, NO. 1, JANUARY 2013

[10] V. Haasz, přednášky předmětu A3B38SME, FEL – ČVUT, 2012

[11] Touch screen digitizer AD7843, Analog devices, katalogový list AD převodníku dotykového displeje, přiložen na CD dodávaným s deskou

[12] VHDL Handbook, HARDI Electronics AB, 2000

[13] J. Fischer, přednášky z předmětu A4B38NVS, FEL – ČVUT, 2012

Příloha A

Zdrojový kód bloku „LCD“.

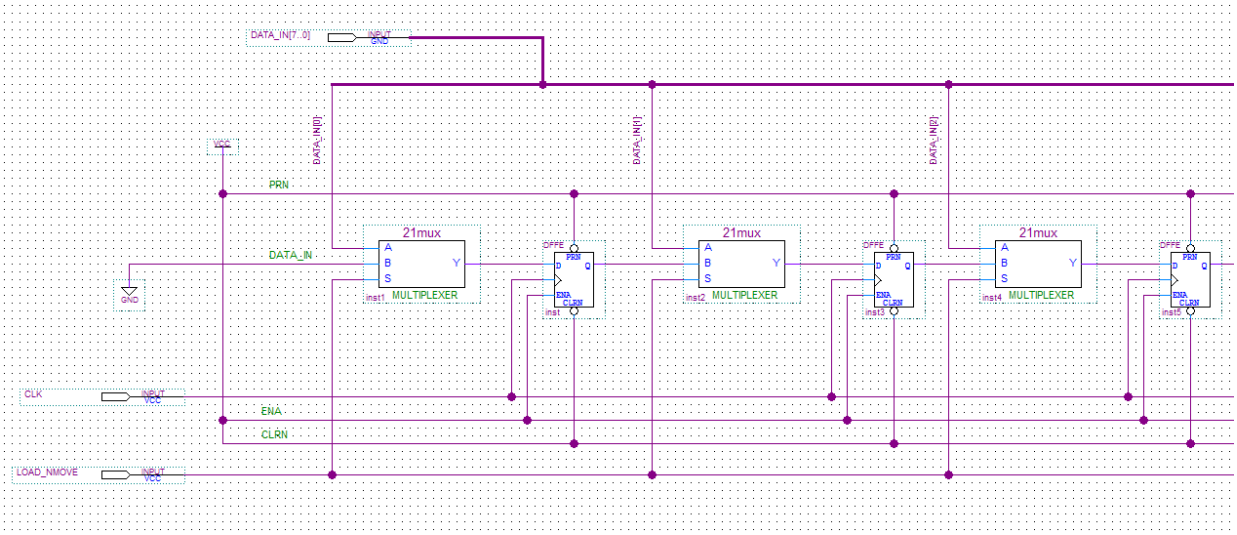
```
--project:      lcd_display
--file:         lcd_display.vhd
--version:      2
--
--author:       Michal Svandrlik
--date:         7.5.2013
--licence:      GNU public
--SW:           Quartus II 11.1 SP2
--HW:           Altera tPad / Cyclone IV E
-- purpose:     education in course A0B35SPS - CTU in Prague - FEE

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity lcd is
port (
        CLK : IN STD_LOGIC; -- INPUT CLOCK SIGNAL
        RED,GRE,BLU : IN STD_LOGIC_VECTOR(5 DOWNTO 0); -- INPUT COLOR DATA

        LCD_DIM, LCD_DEN, LCD_CLK: OUT STD_LOGIC;   -- LCD CONTROLLER SIGNALS
        LCD_R,LCD_G,LCD_B : OUT STD_LOGIC_VECTOR(5 DOWNTO 0);       --    OUTPUT    COLOR
DATA
        HOR_COUNT : OUT UNSIGNED(10 downto 0); -- CURRENTLY DISPLAYED PIXEL - COLUMN
        VER_COUNT : OUT UNSIGNED(9 downto 0); -- CURRENTLY DISPLAYED PIXEL - ROW
        REFRESH : OUT STD_LOGIC                -- END OF FRAME AUXILIARY SIGNAL
);
end lcd;

architecture driver_lcd of lcd is
-- CONSTANTS - ACTIVE SURFACE LIMITS
        constant HORIZONTAL_LIM : unsigned(10 downto 0) := "01100011111"; -- 799
        constant VERTICAL_LIM : unsigned(9 downto 0)       := "1001010111"; -- 599
-- STATE REGISTERS FOR COUNTERS
```

```
        signal hor_reg, hor_next : unsigned(10 downto 0); -- HORIZONTAL COUNTER
        signal ver_reg, ver_next : unsigned(9 downto 0); -- VERTICAL COUNTER
-- DATA ENABLE INTERNAL SIGNAL
        signal den : STD_LOGIC;
begin
-- REGISTER --
-- COUNTER REGISTER REFRESH ON EVERY RISING EDGE OF CLOCK SIGNAL
        process(CLK)
        begin
                if (rising_edge(CLK)) then
                        hor_reg <= hor_next;
                        ver_reg <= ver_next;
                end if;
        end process;

-- NEXT STATE LOGIC --
-- COLUMN COUNTER: INCREMENTED ON EVERY CLOCK PULSE
-- SETS TO ZERO WHEN hor_reg = 1055
        hor_next <= hor_reg + 1         when hor_reg < 1055 else
                                        "00000000000";
-- ROW COUNTER: INCREMENTED WHEN (hor_reg >= 1055).
-- SETS TO ZERO WHEN (ver_reg >= 627) and (hor_reg >= 1055) - AT THE END OF FRAME
-- OTHERWISE HOLDS IT'S VALUE
        ver_next <=     ver_reg + 1     when hor_reg >= 1055 else
                        "0000000000"    when (ver_reg >= 627) and (hor_reg >= 1055) else
                        ver_reg;

-- OUTPUT LOGIC --
        LCD_CLK <=      CLK; -- INPUT AND OUTPUT CLOCK CONNECTION

        -- DATA VALID ONLY WHEN COUNTERS ARE ON DISPLAYED SURFACE
        den <= '1' when ((hor_reg < HORIZONTAL_LIM) and (ver_reg < VERTICAL_LIM)) else
                '0';

        -- COLOURS ARE SENT ONLY WHEN den = '1'
        -- OTHERWISE BLACK IS SENT (ALL "000000")
        LCD_R <=        RED when den = '1' else
                        "000000";
        LCD_G <=        GRE when den = '1' else
```

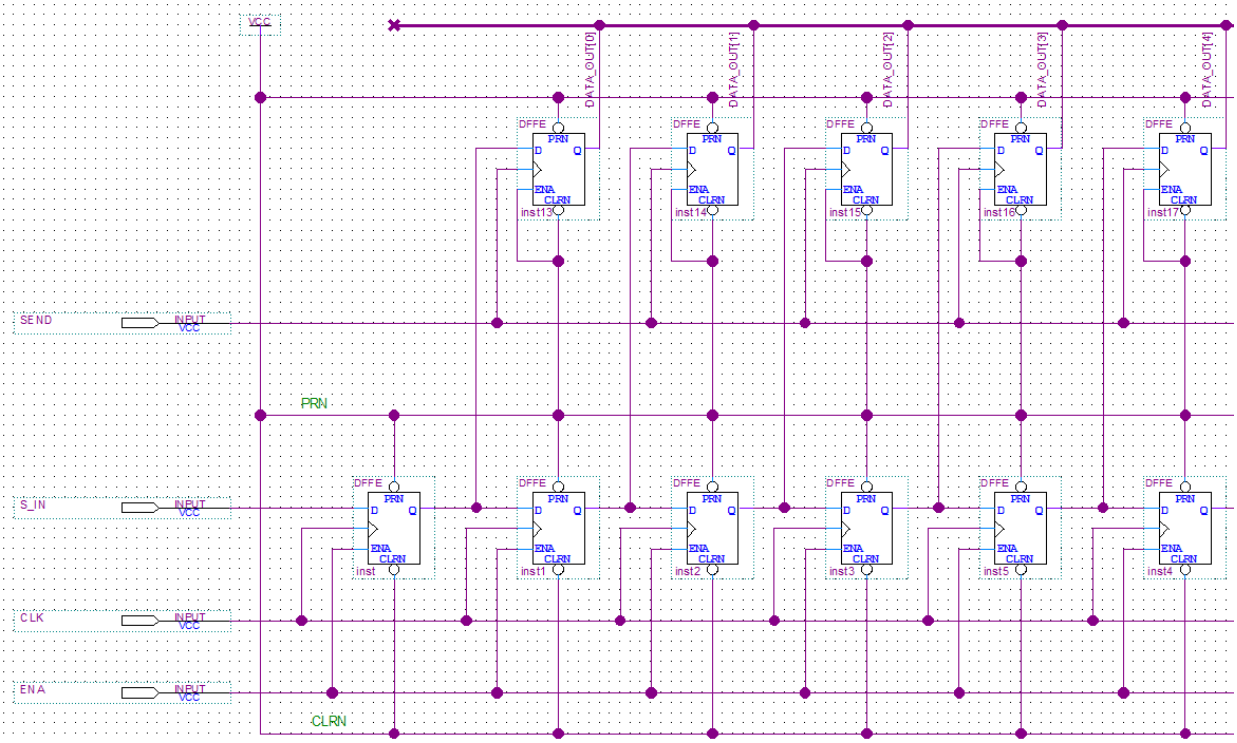
```
                    "000000";
        LCD_B <=    BLU when den = '1' else
                    "000000";

        -- INTERNAL SIGNAL DEN IS CONNECTED TO LCD_DEN OUTPUT
        LCD_DEN <= den;

        -- REFRESH SIGNAL: PULSE IS SENT AT THE END OF FRAME
        REFRESH <=  '1' when (hor_reg >= 1054) and (ver_reg >= 627) else
                    '0';
        -- DISPLAY IS BACKLIT ALL THE TIME
        LCD_DIM <= '1';

        -- VOUNTER VALUES ARE SENT OUT FOR COMMUNICATION WITH OTHER UNITS
        HOR_COUNT <= hor_reg;
        VER_COUNT <=  ver_reg;

end driver_lcd;
```

Příloha B

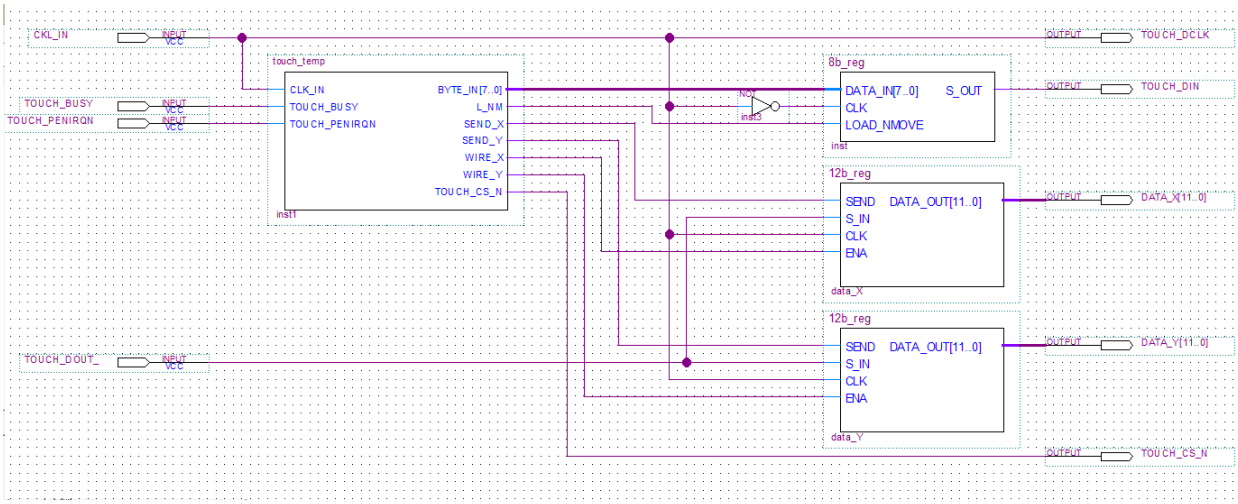
Schéma 8bitového a 12bitového posuvného registru pro komunikaci s AD7843.



Obr. 14: 8bitový registr



Obr. 15: 12bitový registr



Obr. 16: Vnitřní zapojení bloku TOUCH

Příloha C

Zdrojový kód bloku „TOUCH“:

```
--project:      lcd_display
--file:         touch.vhd
--version:      4
--
--author:       Michal Svandrlik
--date:         7.5.2013
--licence:      GNU public
--SW:           Quartus II 11.1 SP2
--HW:           Altera tPad / Cyclone IV E
-- purpose:     education in course A0B35SPS - CTU in Prague - FEE


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity touch is
port (
        CLK_IN : IN STD_LOGIC;          -- input clock signal
        TOUCH_DOUT : IN STD_LOGIC;                      -- input data from ADC
        TOUCH_BUSY : IN STD_LOGIC;              -- converter is busy
        TOUCH_PENIRQN : IN STD_LOGIC;       -- touch "pen interupt"

        TOUCH_DCLK : OUT STD_LOGIC;         -- output clock signal
        TOUCH_DIN : OUT STD_LOGIC;                      -- output control Byte for AD
        TOUCH_CS_N : OUT STD_LOGIC;         -- signal chip select

        DATA_X : OUT  STD_LOGIC_VECTOR(11 DOWNTO 0); -- output data OF X axis
        DATA_Y : OUT  STD_LOGIC_VECTOR(11 DOWNTO 0)  -- output data OF Y axis
);
end touch;

architecture ctrl of touch is

-- COMPONENTS OF TOUCH BLOCK --
------------------------------

-- COMPONENT OF 8b REGISTER FOR ADC CONTROL
        COMPONENT \8b_reg\
                PORT
```

```
		(
			CLK :  IN  STD_LOGIC;
			LOAD_NMOVE :  IN  STD_LOGIC;
			DATA_IN :  IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
			S_OUT :  OUT  STD_LOGIC
		);
	END COMPONENT;

-- COMPONENT OF 12b REGISTERFOR READING DATA FROM ADC
	COMPONENT \12b_reg\
		PORT
		(
			S_IN :  IN  STD_LOGIC;
			CLK :  IN  STD_LOGIC;
			ENA :  IN  STD_LOGIC;
			SEND :  IN  STD_LOGIC;
			DATA_OUT :  OUT  STD_LOGIC_VECTOR(11 DOWNTO 0)
		);
	END COMPONENT;

-- SIGNALS A CONSTANTS OF TOUCH BLOCK --
----------------------------------------

-- STATE MACHINE SIGNALS --

	signal count_reg, count_next : UNSIGNED(4 downto 0);-- main counter for control of ADC
	signal valid_reg, valid_next : UNSIGNED(7 downto 0); -- counter of number of measurement
	signal state_next, state_reg : STD_LOGIC_VECTOR(3 downto 0);-- state register for output
logic

-- SIGNALS FOR COMPONENT AND PROCESS CONNECTION --

signal BYTE_IN : STD_LOGIC_VECTOR(7 downto 0);   -- CONTROL BYTE FOR 8BIT REGISTER DATA
signal WIRE_Y : STD_LOGIC;            -- AUXILIARY SIGNAL ENABLE FOR Y AXIS 12BIT REGISTER
signal WIRE_X : STD_LOGIC;            -- AUXILIARY SIGNAL ENABLE FOR Y AXIS 12BIT REGISTER
signal L_NM : STD_LOGIC;              -- AUXILIARY SIGNAL MODE SELECTION FOR 8BIT REGISTER
-- AUXILIARY SIGNALS FOR OUTPUT LATCHES OF 12BIT REGISTERS
signal SEND_X, SEND_Y : STD_LOGIC;


-- CONSTANTS --
--CONTROL CONSTANT FOR X AXIS MEASUREMENT
constant get_data_X : STD_LOGIC_VECTOR(7 DOWNTO 0) := "11010010";
-- CONTROL CONSTANT FOR Y AXIS MEASUREMENT
constant get_data_Y : STD_LOGIC_VECTOR(7 DOWNTO 0) := "10010010";
constant clock_end : UNSIGNED(4 downto 0) := "11111";      -- LAST COUNTER STATE
```

```
        begin

-- INSTANCES OF COMPONENTS OF TOUCH BLOCK --
-----------------------------------------
-- SHIFT REGISTER FOR SETTING ADC MEASUREMENT MODE --
-- CONNECTING OF INPUTS/OUTPUTS OF REGISTER TO AUXILIARY SIGNALS DEFINED ABOVE
-- WHICH LET US COMMUNICATE WITH THE REGISTER

        b2v_inst : \8b_reg\
        PORT MAP(CLK => not CLK_IN,
-- INPUT CLOCK SIGNAL IS NEGATED TO CHANGE REGISTER'S OUTPUT ON OPPOSITE
-- CLOCK EDGE THEN ADC RECIEVES THE DATA
                                LOAD_NMOVE => L_NM,
                                DATA_IN => BYTE_IN,
                                S_OUT => TOUCH_DIN);

-- SHIFT REGISTER FOR RECIEVING X AXIS DATA --
        b2v_instX : \12b_reg\
        PORT MAP
                (
                        S_IN => TOUCH_DOUT,
                        CLK => CLK_IN,
                        ENA => WIRE_X,
                        SEND => SEND_X,
                        DATA_OUT => DATA_X);

-- SHIFT REGISTER FOR RECIEVING Y AXIS DATA --
        b2v_instY : \12b_reg\
        PORT MAP
                (
                        S_IN => TOUCH_DOUT,
                        CLK => CLK_IN,
                        ENA => WIRE_Y,
                        SEND => SEND_Y,
                        DATA_OUT => DATA_Y);

        TOUCH_DCLK <= CLK_IN;        -- INPUT AND OUTPUT CLOCK CONNECTION

-- REGISTER --
-- SYNCHRONIZING PART FOR STATE REGISTER UPDATE
        process(CLK_IN)
        begin
                if(rising_edge(CLK_IN)) then
                        -- CHANGES TAKE EFFECT ON RISING EDGE OF CLK SIGNAL
                        count_reg <= count_next;
                        state_reg <= state_next;
                        valid_reg <= valid_next;
```

```
                end if;
        end process;

-- NEXT STATE LOGIC --
----------------------
        -- CS_N -- CHIP SELECT SIGNAL
        state_next(0) <=        '1' when count_reg = (clock_end - 1) else
                                '0';
        -- REG_ENA -- WHEN '1', 12BIT REGISTER IS RECIEVING MEASURED DATA
        state_next(1) <= '1'    when ((count_reg > 7) and (count_reg < 20)) else
                                '0';
        -- L_NM -- WHEN '0', CONTROL BYTE IS SENT TO ADC
        state_next(2) <= '0'    when ((count_reg >= 0) and (count_reg < 9)) else
                                '1';
        -- SEND_DATA -- CONTROL PULSE FOR 12BIT SHIFT REGISTER OUTPUT LATCHES
        state_next(3) <= '1'    when ((count_reg = 23) ) else
                                '0';
        -- count level 1 -- MAIN COUNTER FOR MEASUREMENT CONTROL
        count_next <=  count_reg + 1;

        -- count level 2 -- COUNTER FOR DATA VALIDATION AND AXIS SELECTION
        valid_next <=   valid_reg + 1 when count_reg = clock_end else
                        valid_reg;

-- OUTPUT LOGIC --
------------------
        -- THIS CONDITION MAKES ONE AXIS BEING MEASURED 128 TIMES AND THEN MEASURE THE
OTHER ONE
        BYTE_IN <=      get_data_X when valid_reg(7) = '0' else
                        get_data_Y;
        -- PASSES ONLY THE LAST MEASURED VALUES TO 12BIT REGISTER
        WIRE_X <=       state_reg(1) when valid_reg = "01111111" else
                        '0';
        WIRE_Y <=       state_reg(1) when valid_reg = "11111111" else
                        '0';
        -- PASSES ONLY THE LAST MEASURED VALUES THROUGH 12BIT REGISTER LATCHES
        SEND_X <=       state_reg(3) when valid_reg = "01111111" else
                        '0';
        SEND_Y <=       state_reg(3) when valid_reg = "11111111" else
                        '0';
        -- STATE REGISTER SIGNALS CONNECTED TO OUTPUTS
        TOUCH_CS_N      <= state_reg(0);
        L_NM            <= state_reg(2);
end ctrl;
```

Příloha D

Zdrojový kód bloku color_flag

```
--project:      lcd_display
--file:         color_flag.vhd
--version:      1
--
--author:       Michal Svandrlik
--date:         7.5.2013
--licence:      GNU public
--SW:           Quartus II 11.1 SP2
--HW:           Altera tPad / Cyclone IV E
-- purpose:     education in course A0B35SPS - CTU in Prague - FEE


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity color_flag is
port (
        CLK : IN STD_LOGIC;   -- INPUT CLOCK MUST BE SAME AS LCD MODULE
        HOR_C : IN UNSIGNED(10 downto 0); -- HORIZONTAL COUNTER INPUT
        VER_C : IN UNSIGNED(9 downto 0); -- VERTICAL COUNTER INPUT
        TOUCH_X, TOUCH_Y : STD_LOGIC_VECTOR(11 DOWNTO 0); -- TOUCH COORDINATES
        REFRESH : IN STD_LOGIC;       -- END OF FRAME SIGNAL

        LCD_R,LCD_G,LCD_B : OUT STD_LOGIC_VECTOR(5 DOWNTO 0)-- COLOR DATA OUTPUTS
);
end color_flag;

architecture flag_mover of color_flag is

-- SIGNALS OF COLOR_FLAG BLOCK --
        -- STATE MACHINE SIGNALS FOR COLOURS
signal red_reg, blu_reg, gre_reg, red_next, blu_next, gre_next : STD_LOGIC_VECTOR(5 downto 0);
        -- STATE MACHINE COORDINATES FOR TOP LEFT CORNER OF THE FLAG
signal flagx_reg, flagx_next : UNSIGNED(10 downto 0);
signal flagy_reg, flagy_next : UNSIGNED(9 downto 0);
        -- COORDINATES OF FOLLOWING PIXEL
signal HOR : UNSIGNED(10 downto 0);
signal VER : UNSIGNED(9 downto 0);
        -- COORDINATES OF PREVIOUSLY MEASURED TOUCH, CURRENTLY MEASURED TOUCH AND
```

```
SPEED OF FLAG MOVEMENT
signal xprev_reg, xprev_next, xco_reg, xco_next, xmove_reg, xmove_next : UNSIGNED(6 downto 0);
signal yprev_reg, yprev_next, yco_reg, yco_next, ymove_reg, ymove_next : UNSIGNED(6 downto 0);

        -- DIRECTION OF FLAG MOVEMENT, TOUCH DETECTION SIGNALS
signal down_reg, down_next, right_reg, right_next, touch_reg, touch_next : STD_LOGIC;

begin
-- MAKING FUTURE PIXEL FROM PREVIOUS
        VER <=          VER_C + 1 when (HOR_C + 1) = 1055 else
                        VER_C;
        HOR <=          "00000000000" when HOR_C + 1 = 1055 else
                        HOR_C + 1;

-- REGISTER --
-- "COLOUR" PROCESS --
-- UPDATE OF COLOUR REGISTERS AND TOUCH DETECTION
        process(CLK)
        begin
                if falling_edge(CLK) then
                        red_reg <= red_next;
                        gre_reg <= gre_next;
                        blu_reg <= blu_next;
                        touch_reg <= touch_next;
                end if;
        end process;

        -- COLOUR NEXT STATE LOGIC --
        -- RED IS SENT ON ALL THE FLAG EXCEPT THE BLUE WEDGE
        red_next <=     "111111"        when ((hor) >= flagx_reg + (ver + 1 - flagy_reg))
                                        and ((HOR + 1) <= flagx_reg + 400) and
                                        ((VER) >= flagy_reg) and ((VER) <= flagy_reg + 150) else

                        "111111"        when ((HOR) >= flagx_reg + (300 - (VER - flagy_reg)))
                                        and ((HOR) <= flagx_reg + 400) and
                                        ((VER) > flagy_reg + 150) and ((VER) <= flagy_reg + 300)
else
                                        "000000";
        -- GREEN IS SENT ONLY ON WHITE STRIP
        gre_next <=     "111111" when ((HOR) >= flagx_reg +(VER + 1 - flagy_reg))
                                and ((HOR) <= flagx_reg + 400) and
                                ((VER) >= flagy_reg) and ((VER + 1) <= flagy_reg + 150) else
                        "000000";
        -- BLUE IS SENT IN BLUE WEDGE AND WHITE STRIP
        blu_next <=     "111111" when ((HOR) >= flagx_reg) and ((HOR) <= flagx_reg + 400) and
                                ((VER) >= flagy_reg) and ((VER) <= flagy_reg + 150) else
                        "111111" when ((HOR) >= flagx_reg) and
```

```
                        ((HOR) < flagx_reg + (300 - (VER - flagy_reg))) and
                        ((VER) > flagy_reg + 150) and ((VER) <= flagy_reg + 300) else
                "000000";
        -- CONDITION FOR VALID DATA
        touch_next <= '1' when unsigned(TOUCH_X(11 downto 4)) > "00110100" and
                        unsigned(TOUCH_Y(11 downto 4)) > "00110000" else
                '0';

-- "MOVEMENT" REGISTER --
-- COORDINATES OF TOUCH, FLAG POSITION, FLAG SPEED AND DIRECTION ARE UPLOADED HERE
        process(REFRESH)
        begin
                if rising_edge(REFRESH) then
                        flagx_reg <= flagx_next;
                        flagy_reg <= flagy_next;
                        right_reg <= right_next;
                        down_reg <= down_next;

                        xco_reg <= xco_next;
                        yco_reg <= yco_next;

                        xprev_reg <= xprev_next;
                        yprev_reg <= yprev_next;

                        xmove_reg <= xmove_next;
                        ymove_reg <= ymove_next;

                        xmove_reg(0) <= '0';
                        ymove_reg(0) <= '0';
                end if;
        end process;

-- MOVEMENT NEXT STATE LOGIC --
-- POSITION --
-- POSITION OF FLAG AT X AXIS - IF X-COORD. IS LOWER THAN NEXT STEP MOVEMENT AND DIRECTION IS TO THE LEFT
-- IT GIVES FLAG [1,Y] COORDINATES
-- IF right = 0, THEN FLAG MOVES xmove PIXELS TO THE RIGHT , IF right = 1 FLAG MOVES TO THE LEFT
        flagx_next <=  "00000000001" when (right_reg = '1') and (flagx_reg < xmove_reg) else
                        flagx_reg + xmove_reg  when right_reg = '0' else
                        flagx_reg - xmove_reg;
-- POSITION OF FLAG AT Y AXIS - IF Y-COORD. IS LOWER THAN NEXT STEP MOVEMENT AND DIRECTION IS UP
-- IT GIVES FLAG [X,1] COORDINATES
-- IF down = 0, THEN FLAG MOVES ymove PIXELS DOWN , IF down = 1 FLAG MOVES UP
        flagy_next <=  "0000000001"   when (down_reg = '1') and (flagy_reg < ymove_reg) else
```

```
                    flagy_reg + ymove_reg when down_reg = '0' else
                    flagy_reg - ymove_reg;
-- DIRECTION --
-- IF AXIS X OF FLAG IS HIGHER THAN 400 AND MOVES TO THE RIGHT AND DISPLAY IS NOT TOUCHED OR
-- MEASURED COORDINATES X IS LOWER THEN THE PREVIOUS AND DISPLAY IS TOUCHED, CHANGE VALUE TO '1'
        right_next <=  '1' when ((flagx_reg > 400) and (right_reg = '0') and (touch_reg = '0')) or
                            ((xco_reg < xprev_reg) and (touch_reg = '1')) else
-- IF AXIS X OF FLAG IS LOWER THAN 2 AND MOVES TO THE LEFT AND DISPLAY IS NOT TOUCHED OR
-- MEASURED COORDINATES X IS HIGHER THEN THE PREVIOUS AND DISPLAY IS TOUCHED, CHANGE VALUE TO '0'
                    '0' when ((flagx_reg < 2) and (right_reg = '1') and (touch_reg = '0')) or
                            ((xco_reg >= xprev_reg) and (touch_reg = '1')) else
                    right_reg;
-- IF AXIS Y OF FLAG IS HIGHER THAN 300 AND MOVES DOWN AND DISPLAY IS NOT TOUCHED OR
-- MEASURED COORDINATES Y IS HIGHER THEN THE PREVIOUS AND DISPLAY IS TOUCHED, CHANGE VALUE TO '1'
        down_next <=  '1' when ((flagy_reg > 300) and (down_reg = '0') and (touch_reg = '0')) or
                            ((yco_reg >= yprev_reg) and (touch_reg = '1')) else
        -- IF AXIS Y OF FLAG IS LOWER THAN 2 AND MOVES UP AND DISPLAY IS NOT TOUCHED OR
        -- MEASURED COORDINATES Y IS LOWER THEN THE PREVIOUS AND DISPLAY IS TOUCHED, CHANGE VALUE TO '0'
                    '0' when ((flagy_reg < 2) and (down_reg = '1') and (touch_reg = '0')) or
                            ((yco_reg < yprev_reg) and (touch_reg = '1')) else
                            down_reg;

        -- PREVIOUS MEASUREMENT
        -- IF DISPLAY IS TOUCHED, LAST VALUE IS MOVED TO THE PREVOIUS ONE
        xprev_next <=  xco_reg when (touch_reg = '1') else
                    xprev_reg;
        yprev_next <=  yco_reg when (touch_reg = '1') else
                    yprev_reg;

        -- THIS MEASUREMENT
        -- IF DISPLAY IS TOUCHED, MEASURED VALUE IS MOVED FROM INPUT TO REGISTERS
        xco_next <=    unsigned(TOUCH_X(11 downto 5))     when (touch_reg = '1') else
                    xco_reg;
        yco_next <=    unsigned(TOUCH_Y(11 downto 5))     when (touch_reg = '1') else
                    yco_reg;

        -- SPEED
        -- COUNTING "SPEED" OF FLAG
        -- NUMBERS ARE UNSIGNED VALUES SO WE MUST COMPARE THEM BEFORE SUBTRACTING
        xmove_next <=xco_reg - xprev_reg     when xco_reg >= xprev_reg else
                    xprev_reg - xco_reg;
        ymove_next <=yco_reg - yprev_reg     when yco_reg >= yprev_reg else
                    yprev_reg - yco_reg;
```

```
-- OUTPUT LOGIC --
------------------

        LCD_R <= red_reg;
        LCD_G <= gre_reg;
        LCD_B <= blu_reg;

end flag_mover;
```

Příloha E

Obsah přiloženého CD:

DISPL – Složka s projektem
Obrázky – Složka s obrázky užitými v této práci
svandmi2.pdf – Tato práce ve formátu .pdf