FITkit

a principy návrhu obvodů pro FPGA v jazyku VHDL

Přednáška INP
Michal Bidlo

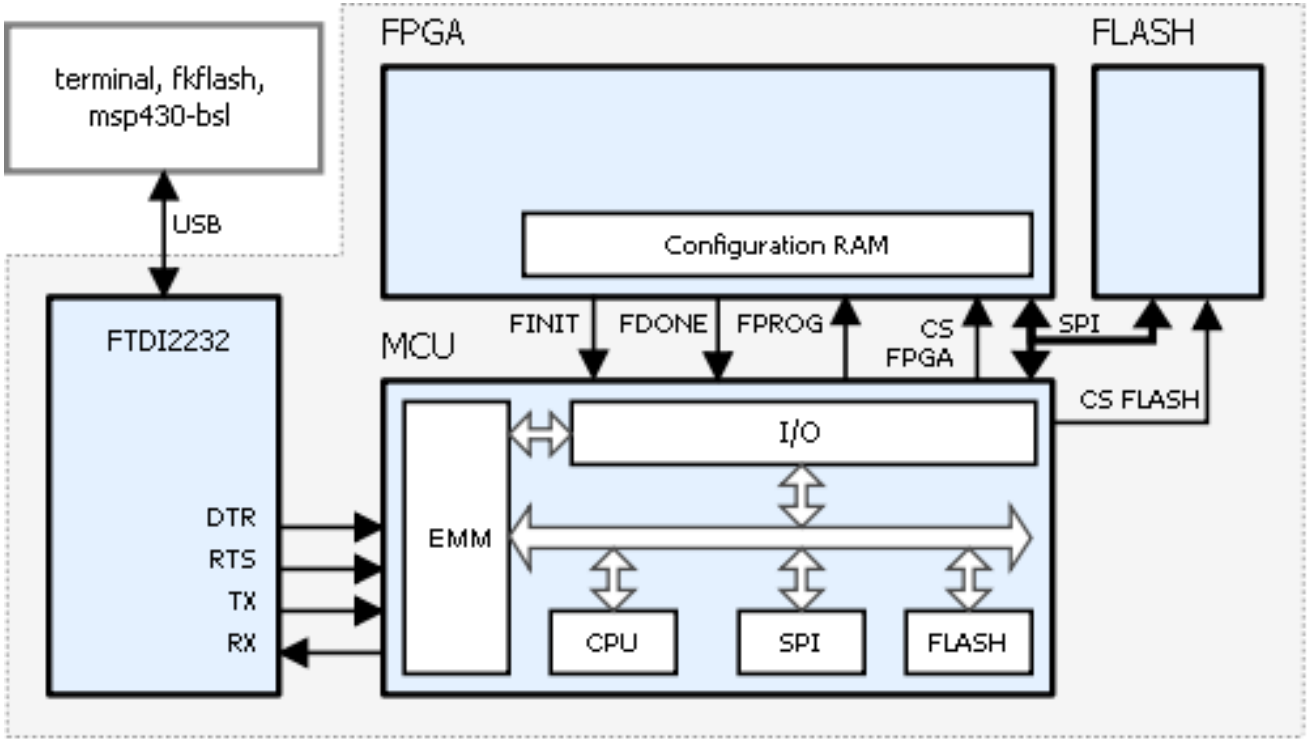
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Obsah

- Úvod: struktura FITkitu, ovladače a pomocné vybavení
- FITkit v příkladech – demonstrace vybraných zařízení
  a jejich HW podpora s využitím jazyka VHDL
  - Návrh jednoduchého čítače
  - Výpis řetězce na LCD displej
  - Klávesnice + LCD: jednoduchý „psací stroj“

Struktura FITkitu

- FITkit = MCU + FPGA + periferie
- Interakce s uživatelem prostřednictvím SW nástrojů speciálně vyvinutých pro FITkit (terminál, překladový systém, fkflash, MSP430-Bootstrap Loader + další knihovny)

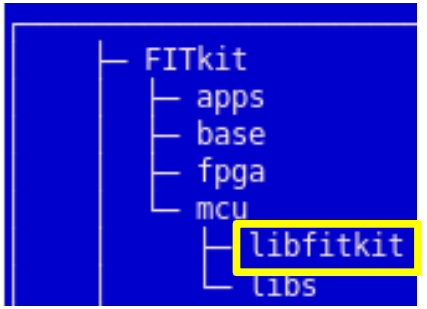


Princip funkce FITkitu

- Centrální řízení je prováděno z MCU. Ten typicky provádí:
  - Přenos dat mezi externí FLASH pamětí, MCU a FPGA
  - Programování FPGA (zápis do konfigurační SRAM paměti FPGA z externí FLASH paměti)
  - Komunikace s FITkitem z PC přes terminál (MCU zpracovává a vykonává příkazy zadané z terminálu uživatelem)
- Většina periferií FITkitu je připojena k FPGA, které též poskytuje řadu IO portů pro všeobecné použití.

Ovladače a pomocné vybavení

- Základem je knihovna libfitkit (sada funkcí pro komunikaci MCU s terminálem, FPGA a FLASH pamětí)
- Př.: Hello World! na FITkitu s využitím knihovny libfitkit



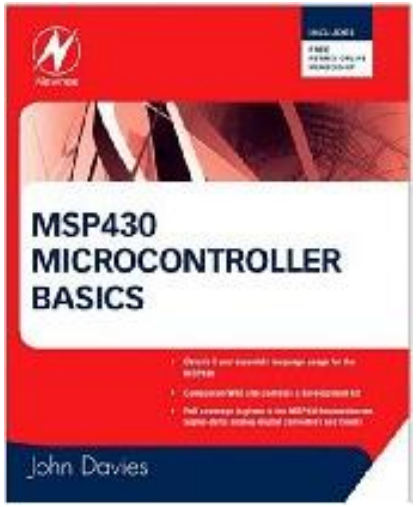
Program pro MCU

```c
#include <fitkitlib.h>

void print_user_help(void) { }

unsigned char decode_user_cmd(char *cmd_ucase, char *cmd)
{
    return (CMD_UNKNOWN);
}

void fpga_initialized() { }

void main(void)
{
    initialize_hardware();
    WDG_stop();                 // zastav watchdog

    term_send_str("Hello World!\n");
    set_led_d5(1);
    set_led_d6(1);

    while (1) {
        terminal_idle();        // obsluha terminalu
    }
}
```

Ukázka kódu pro set_led_d5

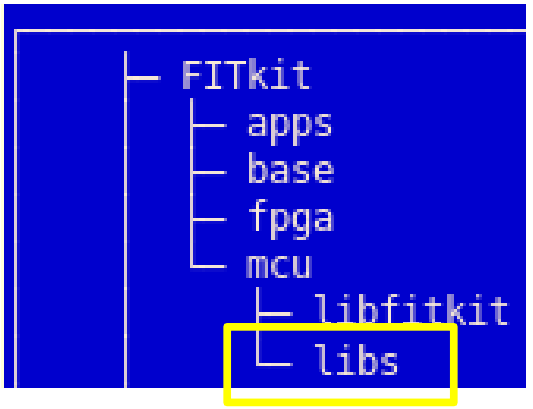
```c
#define set_led_d5(on) {\
  P1DIR |= LED0;      /* nastavit bit jako vystup */ \
  /* vyznam parametru on: 0 - sviti, 1 - nesviti */ \
  P1OUT = (on) ? (P1OUT & ~LED0) : (P1OUT | LED0);  \
}
```

Více o programování MSP430 v C i ASM viz např.



Ovladače a pomocné vybavení

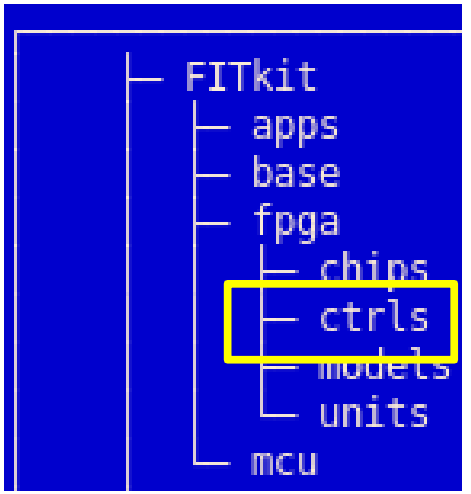
- Sekce MCU: ovladače periferních zařízení v C



Implementace funkcí specifických pro jednotlivé periferie, obecné komunikační operace

- Sekce FPGA: komponenty k zajištění propojení periferních zařízení s MCU

- Návrh systému jako „ze stavebnice“

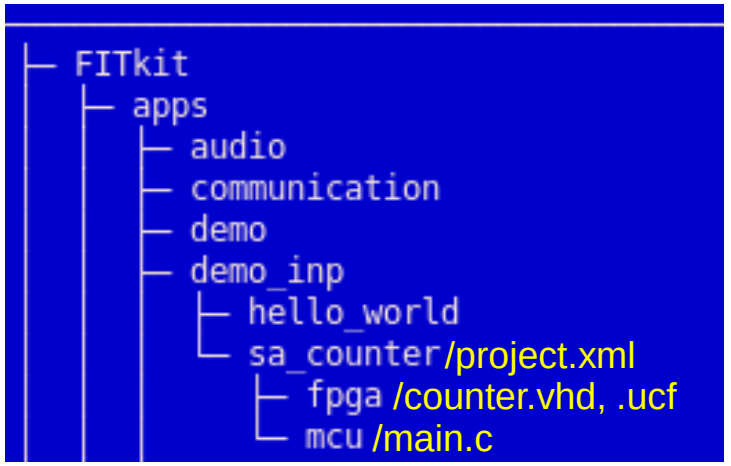


Základy návrhu obvodů na FPGA FITkitu

- Přístup preferující obvodové realizace systémů ve VHDL
- Proč?
  - Aplikačně-specifický HW
  - Možnost optimalizace výkonu a zdrojů při realizaci obvodů v FPGA
  - Plná kontrola návrhu od samého počátku
- Metodika návrhu
  - Návrh probíhá s ohledem na možnosti FPGA, veškeré prvky systému jsou od počátku navrhovány jako HW řešení a popsány ve VHDL
  - Program v MCU realizuje pouze nezbytně nutné funkce pro správnou činnost FITkitu jako celku

Příklad 1 (krok za krokem): návrh jednoduchého čítače pro FPGA

- Samostatný čítač, který bliká s LED D4
- Co budeme potřebovat a muset vytvořit
  - Vytvoříme strukturu projektu s potřebnými soubory



  - Vytvoříme popis v XML pro překladový systém fcmake (soubor project.xml) - viz dokumentace na http://merlin.fit.vutbr.cz/FITkit/docs/navody/kompilacev2.html

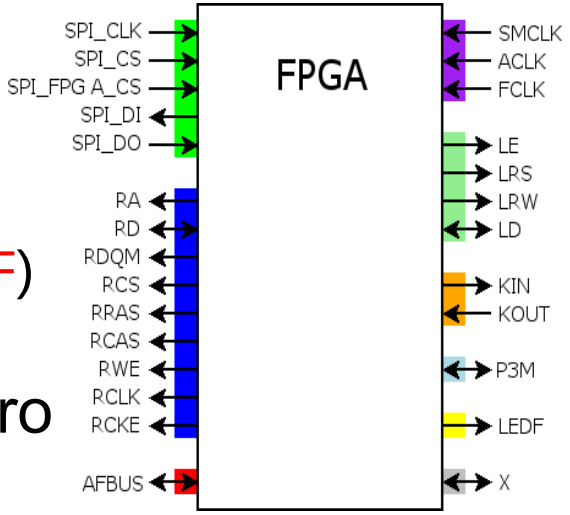
Návrh jednoduchého čítače pro FPGA

- Pro tento projekt potřebujeme specifikovat vlastní tzv. toplevel entitu, user constraits file (.ucf) a skutečnost, že nepoužijeme žádnou předdefinovanou architekturu

```
<?xml version="1.0" encoding="utf-8"?>
<project>

    <!--  Project description -->
    <name>Jednoduchý čítač v FPGA</name>
    <author>Michal Bidlo</author>
    <authoremail>bidlom@fit.vutbr.cz</authoremail>
    <description>Samostatný čítač v FPGA, který bliká s LED D4</description>

    <!--  MCU part -->
    <mcu>
        <file>main.c</file>
    </mcu>

    <!-- FPGA part -->
    <fpga toplevelentity="counter" ucffile="fpga/counter.ucf" architecture="none">
        <file>counter.vhd</file>
    </fpga>

</project>
```

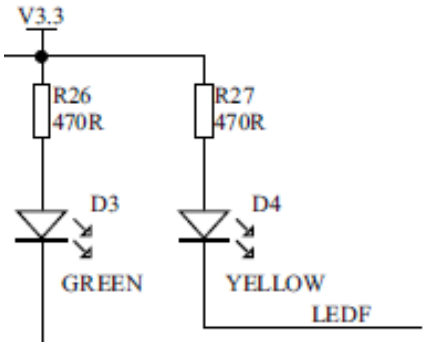
Návrh jednoduchého čítače pro FPGA

- Identifikujeme porty FPGA, které v našem návrhu využijeme. Pro nejjednodušší cyklický čítač potřebujeme:
  - Hodinový signál (SMCLK)
  - Výstup čítače připojený na D4 (LEDF)



- Vytvoříme popis mapování pinů pro syntezátor (soubor counter.ucf)

```
NET "SMCLK"    LOC = "P80"  | PERIOD=8 MHz HIGH 50%;
NET "LEDF"     LOC = "P81"  ; #yellow D4
```



- SMCLK je hodinový signál generovaný z MCU, který má frekvenci 7.3728 Mhz
- LED D4 svítí při LEDF = '0' a nesvítí při LEDF = '1'

Návrh jednoduchého čítače pro FPGA

- Vytvoříme popis obvodu ve VHDL (counter.vhd)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity counter is
port (
   SMCLK   : in std_logic;
   LEDF    : out std_logic
);
end counter;

architecture main of counter is

  signal cnt : std_logic_vector(23 downto 0) := (others => '0');

begin

   LEDF <= cnt(23);
   process(SMCLK)
   begin
      if SMCLK'event and SMCLK = '1' then
         cnt <= cnt + 1;
      end if;
   end process;

end main;
```

Návrh jednoduchého čítače pro FPGA

- Vytvoříme inicializační program pro MCU (nezbytné)

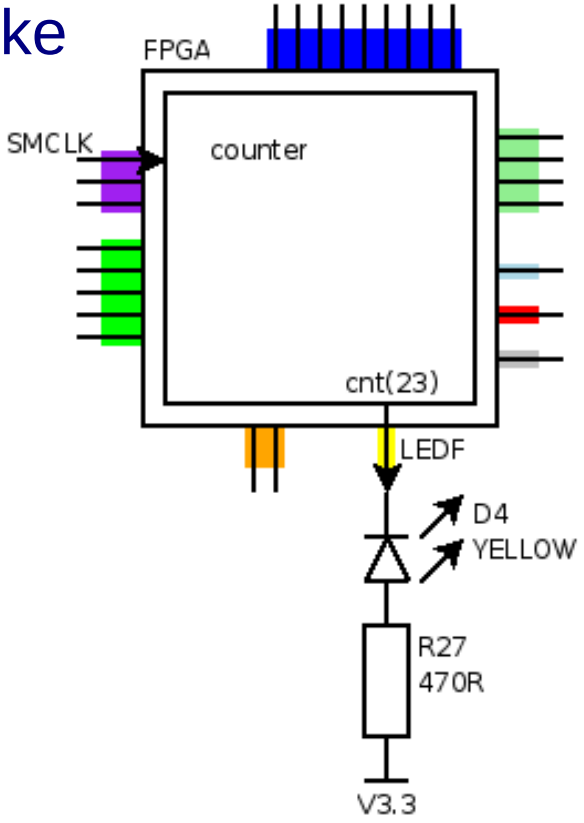
```c
#include <fitkitlib.h>

void print_user_help(void) { }

unsigned char decode_user_cmd(char *cmd_ucase, char *cmd) {

    return (CMD_UNKNOWN);

}

void fpga_initialized() { }

int main(void)
{
    initialize_hardware();

    while (1)
    {
        terminal_idle();  // obsluha terminalu
    }
}
```

Návrh jednoduchého čítače pro FPGA

- Nyní přistoupíme k překladu projektu (lze též přes QDevKit)
  - V adresáři se souborem project.xml spustíme příkaz fcmake; vytvoříme tak soubory pro překlad a syntézu
  - Dále přeložíme projekt příkazem make
  - Některé zajímavé statistiky syntézy (viz build/fpga/project.srp)

```
Device utilization summary:
---------------------------

Selected Device : 3s50pq208-4

 Number of Slices:                       13  out of    768     1%
 Number of Slice Flip Flops:             23  out of   1536     1%
 Number of 4 input LUTs:                 23  out of   1536     1%
 Number of IOs:                           2
 Number of bonded IOBs:                   2  out of    124     1%
 Number of GCLKs:                         1  out of      8    12%
```

```
Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 5.298ns (Maximum Frequency: 188.751MHz)
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: 7.241ns
   Maximum combinational path delay: No path found
```



Návrh jednoduchého čítače pro FPGA

- Hlavní výstupy překladu
  - build/project.bin (bitstream pro FPGA)
  - build/project_f1xx.hex
  - build/project_f2xx.hex (program pro MCU)
- Po úspěšném překladu spustíme QDevKit, zadáme naprogramování FITkitu (možno též pomocí make load), připojíme se přes terminál k FITkitu a můžeme sledovat výstup čítače (jeho nejvyšší bit) v podobě blikající LED D4

Příklad 2: ovládání displeje z FPGA

- Jednoduchá aplikace, která na LCD zobrazí textový řetězec
- LCD vyžaduje dodržení předepsaného časování instrukcí, zejména dostatečnou periodu signálu LE. Frekvence SMCLK je pro LCD příliš vysoká.
  - Využijeme čítač jako dělič kmitočtu
- Před zápisem znaků do LCD je třeba provést jeho inicializaci, případně vymazání (přesně definované instrukce v podobě kombinace hodnot signálů LRS, LRW, LD)
  - Použijeme ROM k uchování inicializačních a znakových instrukcí
  - Pomocí čítače vygenerujeme postupně všechny adresy ROM, čímž dosáhneme zaslání požadované sekvence instrukcí do LCD a vypsání řetězce

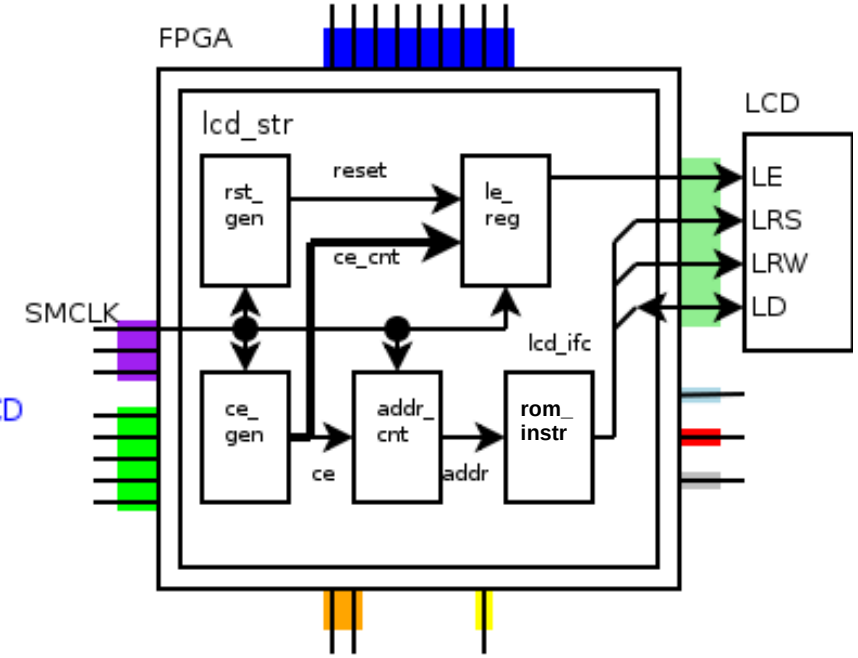
Ovládání displeje z FPGA

- Definice signálu instrukce LCD, paměti ROM a jejího obsahu
  - Pevně zvolená posloupnost instrukcí, která provede inicializaci displeje, vypíše textový řetězec a „skončí“

```
subtype lcd_inst is std_logic_vector(9 downto 0);
signal lcd_ifc: lcd_inst := "0000000000";

type ROM is array(0 to 10) of lcd_inst;
constant rom_instr: ROM := (
    -- LRS, LRW, LD(7 downto 0) <-- toto je instrukcni rozhrani LCD displeje
    "0000111000", -- set to 8bit operation, 2-line display, 5x8 dot font
    "0000001110", -- turn on display and cursor
    "0000000110", -- incr addr mode by 1, shift cursor right, no display shift
    "0000000001", -- clear display, set DDRAM addr to 00H from AC
    "1001000110", -- F
    "1001001001", -- I
    "1001010100", -- T
    "1001101011", -- k
    "1001101001", -- i
    "1001110100", -- t
    "0000000000"  -- output in the idle state
);
```

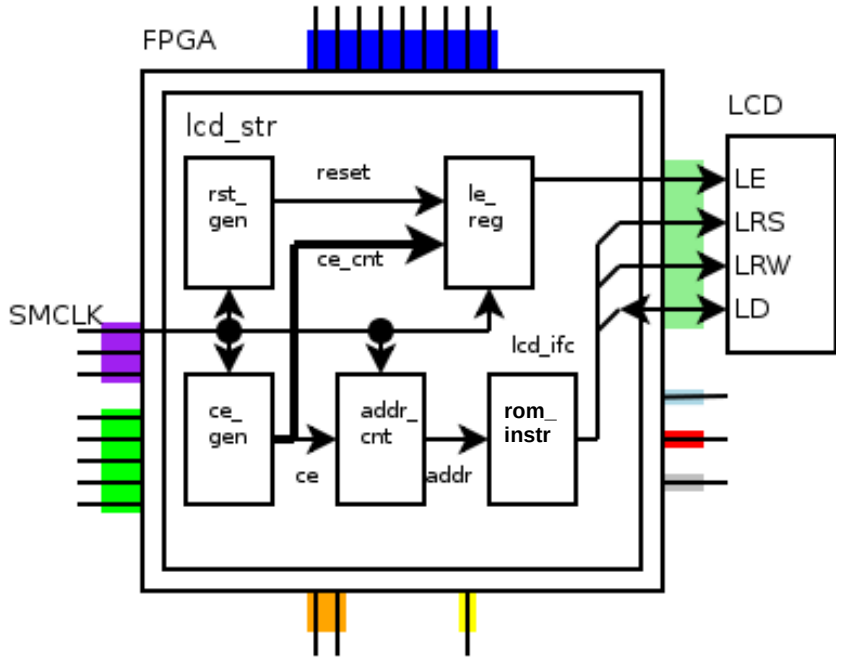
Ovládání displeje z FPGA – popis chování

```
-- vygenerujeme vnitrni reset pro nstaveni pocatecni '1' na LE
-- (tim se s jeho prvni sestupnu hranou vykona prvni instrukce LCD)
reset <= '0' when reset_cnt = X"F" else '1';
rst_gen: process(SMCLK)
begin
    if SMCLK'event and SMCLK = '1' then
        if reset = '1' then
            reset_cnt <= reset_cnt + 1;
        end if;
    end if;
end process rst_gen;

-- citac pro snizeni frekvence zasilani instrukci do LCD
ce_gen: process(SMCLK)          -- (odvozena od SMCLK)
begin
    if SMCLK'event and SMCLK = '1' then
        ce_cnt <= ce_cnt + 1;
    end if;
end process ce_gen;

ce <= '1' when ce_cnt = X"3FFF" else '0';
```



Ovládání displeje z FPGA – popis chování

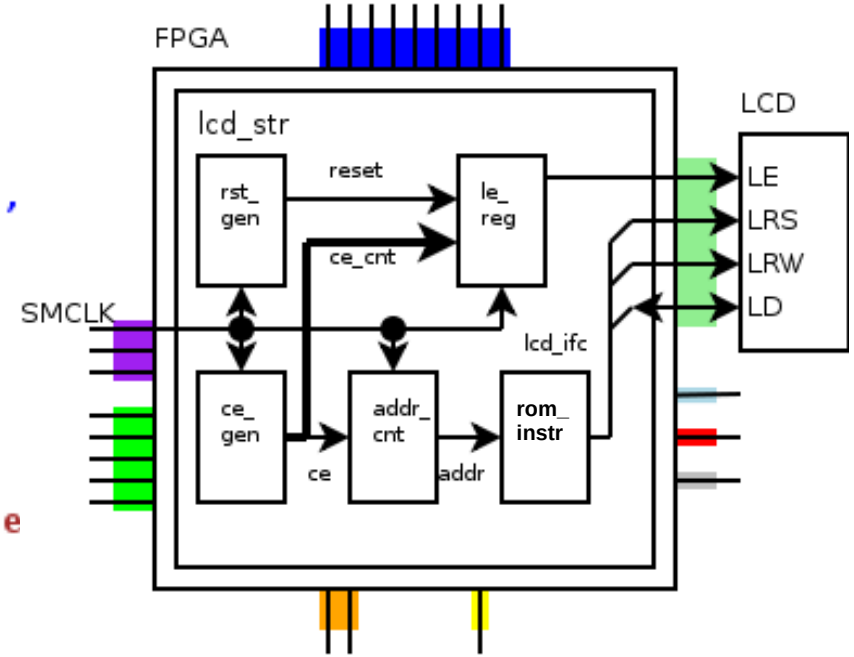
```
-- citac adresy ROM, ze ktere se ctou instrukce pro LCD
addr_cnt: process(SMCLK)
begin
    if SMCLK'event and SMCLK = '1' then
        if ce = '1' then
            if addr < 10 then
                addr <= addr + 1;
            end if;
        end if;
    end if;
end process addr_cnt;

lcd_ifc <= rom_instr(addr);
```

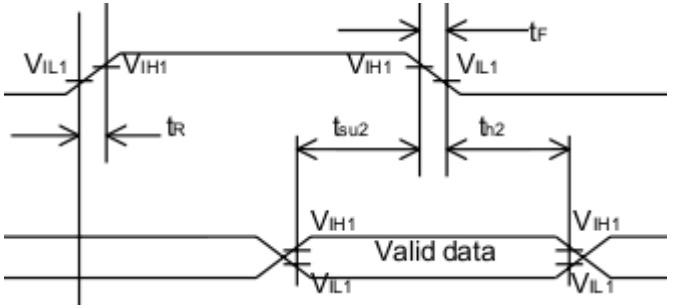


rom_instr je definován jako pole konstant reprezentujících instrukce pro LCD.

Ovládání displeje z FPGA – popis chování

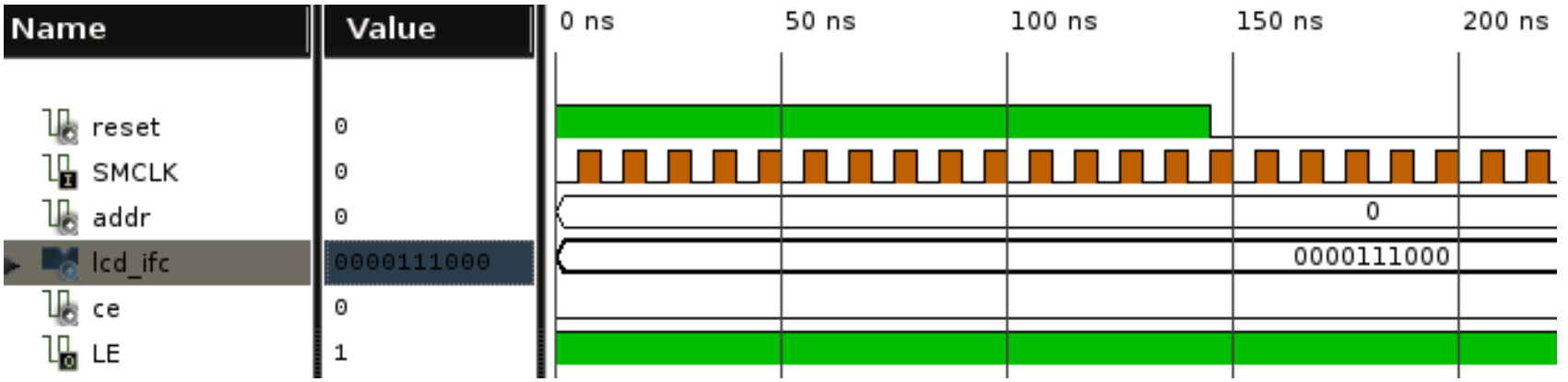
```
-- tento registr ridi prechody signalu LE displeje,
-- jehoz setupne hrany vzorkuji dodavane instrukce
le_register: process(SMCLK, reset)
begin
    if reset = '1' then
        le_reg <= '1';
    elsif SMCLK'event and SMCLK = '1' then
        if ce_cnt = X"3000" or ce_cnt = X"0FFF" the
            le_reg <= not le_reg;
        end if;
    end if;
end process le_register;

LE <= le_reg;
LRS <= lcd_ifc(9);
LRW <= lcd_ifc(8);
LD <= lcd_ifc(7 downto 0);
```



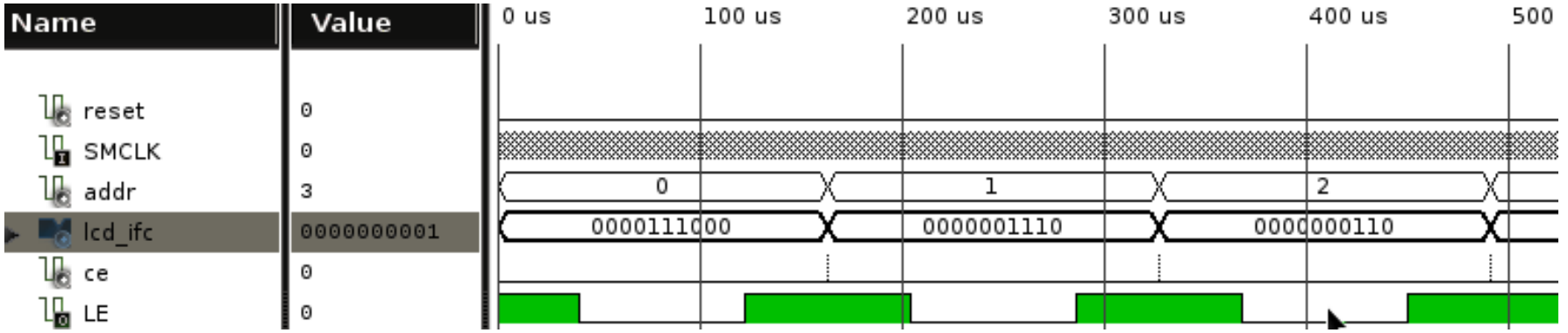


Ovládání displeje z FPGA – ukázka simulace

- Inicializace obvodu (reset)

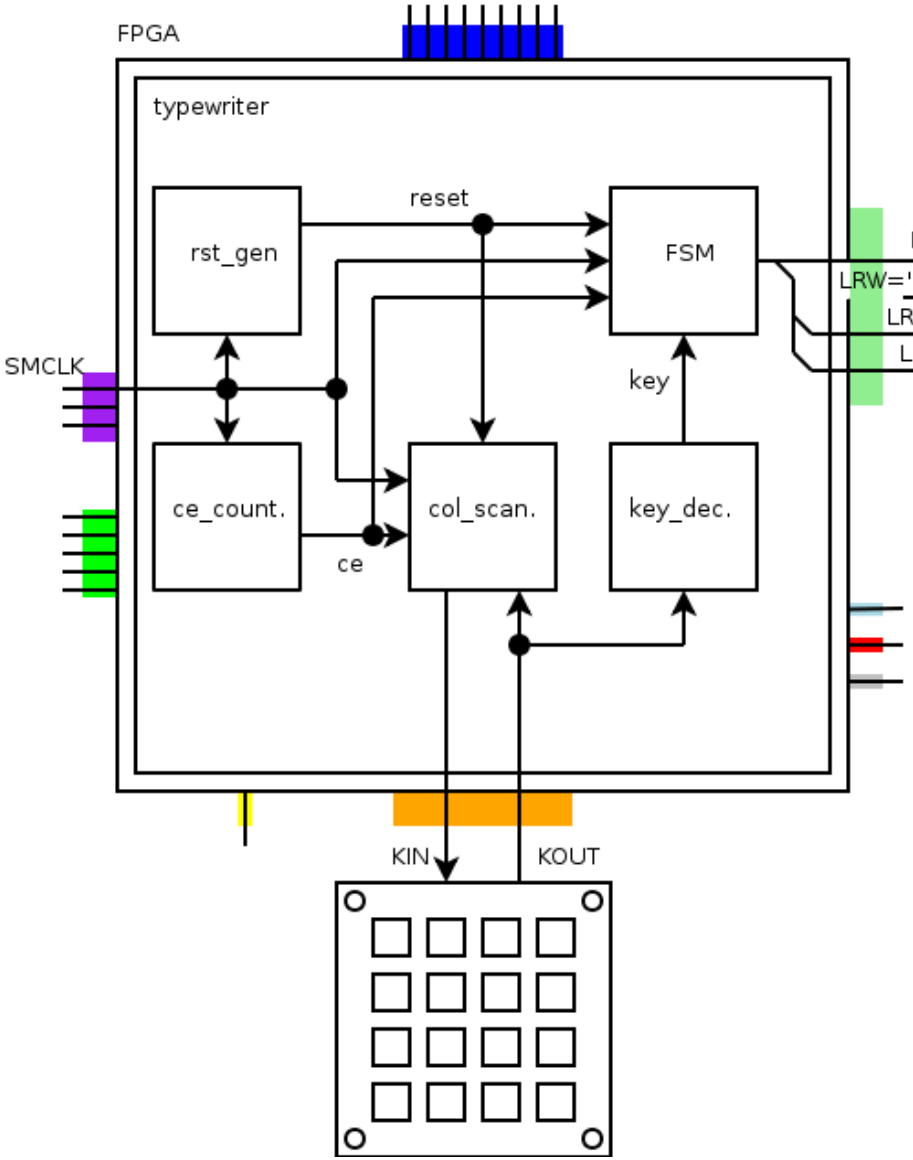


- Hlavní činnost obvodu



Příklad 3: interakce s klávesnicí FITkitu

- Jednoduchý „psací stroj“ využívající klávesnici a LCD



Algoritmus scanování klávesnice:

Čtveřice bitů KIN udává aktivní sloupce (log. 0), které budou citlivé na stisk klávesy.

Na vstup KIN přivádíme cyklicky kombinace bitů, které aktivují vždy jediný sloupec.

Je-li v aktivním sloupci stisknuta klávesa, výstup KOUT určuje řádek (bit v log. 0), ve kterém se nachází stisknutá klávesa.

Jelikož víme, který sloupec je právě aktivní, můžeme na základě pozice nulového bitu v KOUT určit stisknutou klávesu.

Interakce s klávesnicí FITkitu

Scan kláves a dekodér znaků

```
col_scan: process(SMCLK, reset)
begin
    if reset = '1' then
        col_reg <= "1110";
    elsif SMCLK'event and SMCLK = '1' then
        if ce = '1' then
            -- neni-li stisknuta klavesa, provadime sken sloupcu
            if KOUT = "1111" then
                col_reg <= col_reg(2 downto 0) & col_reg(3);
            end if;
            -- (jinak podrzime aktivni sloupec se stisknutou klavesou)
        end if;
    end if;
end process col_scan;
KIN <= col_reg;

-- dekoder znaku ve sloupci aktivovanem signalem col_reg
key_decoder: process(col_reg, KOUT)
begin
    case (col_reg & KOUT) is
        -- 1. sloupec
        when "11101110" => key <= key_1;
        when "11101101" => key <= key_4;
        when "11101011" => key <= key_7;
        when "11100111" => key <= key_krat;
        -- 2. sloupec
        when "11011110" => key <= key_2;
                    .
                    .
                    .
        when others => key <= (others => '0');
    end case;
end process key_decoder;
```

Interakce s klávesnicí FITkitu

Kombinační logika FSM:

inicializace LCD

detekce stisku klávesy, ošetření držení klávesy

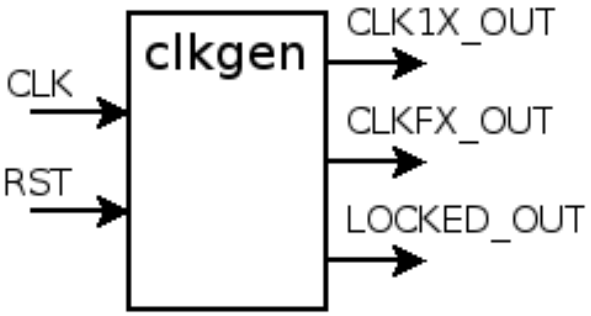
časování vystavení znaku na LCD

```
LE <= '1';
LRS <= '0';
LRW <= '0'; -- nevyuzito => trvale do '0'
LD <= "00000000";
case cstate is
    when SInit => nstate <= SI1phase1;
            LD <= "00111000"; -- 8-bit, 2-line, 5x8-font
    when SI1phase1 => nstate <= SI1phase2;
            LE <= '0';
    when SI1phase2 => nstate <= SI2phase1;
            LD <= "00001110"; -- turn on display and cursor
    when SI2phase1 => nstate <= SI2phase2;
            LE <= '0';
    when SI2phase2 => nstate <= SI3phase1;
            LD <= "00000110"; -- incr addr. mode 1, shift right
    when SI3phase1 => nstate <= SI3phase2;
            LE <= '0';
    when SI3phase2 => nstate <= SI4phase1;
            LD <= "00000001"; -- clear, DDRAM addr. 00H
    when SI4phase1 => nstate <= SI4phase2;
            LE <= '0';
    when SI4phase2 => nstate <= SIdle;
    when SIdle => -- KOUT 1111 = zadna klavesa neni stisknuta
        if KOUT /= "1111" then nstate <= SKeyDown;
            LRS <= '1'; LD <= key;
        else nstate <= SIdle;  end if;
    when SKeyDown => nstate <= SHold;
            LE <= '0';
    when SHold => -- osetrime, aby se znak nevypisoval opakovane
        if KOUT /= "1111" then nstate <= SHold;
        else nstate <= SIdle; end if;
    when others => nstate <= SIdle;
end case;
```

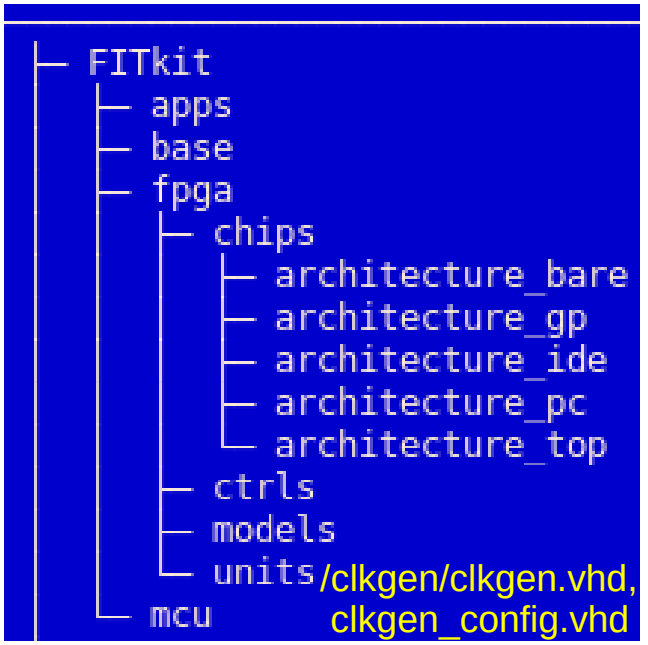
Konstrukce grafu FSM dle výše uvedené VHDL specifikace je ponechána na samostatné cvičení.

Taktování obvodů s využitím DCM vestavěného v FPGA

- Příklad 4: Čítač blikající s LED D4, který využívá hodinový signál o vyšší frekvenci z generátoru hodin



- Využijeme komponentu Xilinx clkgen
  - Vstup: hodiny SMCLK
  - Výstup: hodiny CLKFX_OUT (násobené či dělené SMCLK dle specifikace)
  - CLK1X_OUT jsou původní SMCLK zarovnané vůči CLKFX_OUT
  - LOCKED_OUT je aktivní (v log. 0) před náběhem generátoru, kdy ještě nejsou hodinové výstupy k dispozici
  - Balíček clkgen_cfg definuje možné konstanty pro specifikaci frekvence



Čítač pracující na vyšší frekvenci

- K popisu projektu (soubor project.xml) přidáme informaci o generátoru hodin pro FPGA

```
<?xml version="1.0" encoding="utf-8"?>
<project>

    <!--  Project description -->
    <name>Zrychlený čítač v FPGA</name>
    <author>Michal Bidlo</author>
    <authoremail>bidlom@fit.vutbr.cz</authoremail>
    <description>Čítač, který je taktován zrychlenými hodinami z clkgen a bliká s D4</description>

    <!--  MCU part -->
    <mcu>
        <file>main.c</file>
    </mcu>

    <!-- FPGA part -->
    <fpga toplevelentity="counter" ucffile="fpga/counter.ucf" architecture="none">
        <include>fpga/units/clkgen/package.xml</include>
        <file>counter.vhd</file>
    </fpga>

</project>
```

Čítač pracující na vyšší frekvenci

- VHDL popis modifikovaného obvodu

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.clkgen_cfg.all;

entity counter is
port (
   SMCLK    : in std_logic;
   LEDF     : out std_logic
);
end counter;

architecture main of counter is

    component clkgen is
        generic (
            FREQ        : dcm_freq
        );
        port (
            CLK         : in    std_logic;
            RST         : in    std_logic;

            CLK1X_OUT   : out   std_logic;
            CLKFX_OUT   : out   std_logic;
            LOCKED_OUT  : out   std_logic
        );
    end component;

    signal cnt : std_logic_vector(23 downto 0) := (others => '0');
    signal miclk : std_logic;

begin

    clkgen_inst : entity work.clkgen
    generic map (
        FREQ => DCM_40MHz
    )
    port map (
        CLK => SMCLK,
        RST => '0',

        CLK1X_OUT => open,
        CLKFX_OUT => miclk,
        LOCKED_OUT => open
    );

    LEDF <= cnt(23);

    process(miclk)
    begin
        if miclk'event and miclk = '1' then
            cnt <= cnt + 1;
        end if;
    end process;

end main;
```

Čítač pracující na vyšší frekvenci

- Projekt přeložíme a spustíme stejným způsobem jako v předcházejícím příkladu
- Po naprogramování FITkitu můžeme pozorovat LED D4, jejíž blikání vykazuje vyšší frekvenci
- Výsledek syntézy

```
Device utilization summary:
---------------------------

Selected Device : 3s50pq208-4

 Number of Slices:                       15  out of    768     1%
 Number of Slice Flip Flops:             24  out of   1536     1%
 Number of 4 input LUTs:                 24  out of   1536     1%
 Number of IOs:                           2
 Number of bonded IOBs:                   2  out of    124     1%
 Number of GCLKs:                         1  out of      8    12%
 Number of DCMs:                          1  out of      2    50%
```

```
Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 28.955ns (Maximum Frequency: 34.537MHz)
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: 7.241ns
   Maximum combinational path delay: No path found
```