Týmový projekt do předmětu BPC-DE1 Název projektu: Konzole pro rotoped Členové týmu 1. Smejkal Marek 2. Socha Jakub 3. Soukeník Ondřej 4. Stupka Tomáš 5. Šomšák Martin Link to GitHub project folder Úvod Úkolem tohoto projektu bylo vytvořit konzoli pro měření otáček rotopedu. Toto měření následně vede na rozšíření jako napříkad měření rychlosti, ujeté vzdálenosti, průměrné rychlosti atd. V našem případě jsme se rozhodli pro snímání otáček použít čidlo 44E, který obsahuje halovu sondu. Napájecí napětí čidla je 4,5-24V Výstupem čidla je buď logická 0 nebo 1. Pokud tento sensor umístíme na pevný bod v blízkoti otáčenícího kola, je nutné přidat na otáčejí se kole magnetický puk. Následně při otáčení kola snímač vygeneruje signál pomocí kterého můžeme určit požadované veličiny. Pro následní zobrazení měřené veličiny byl využit 4x7 segmentový displej a tři ledky pro indikaci zobrazovaného výstupu (zapnuto/vypnuto, přepínání zobrazení, reset systému). Popis zapojení Zapojení a PCB jsou navrženy v programu Eagle. V zapojení jsou použity 7 segmentové displeje se společnou anodou. Stejné segmenty jednotlivých digitů jsou pospojovány. Pmod JC slouží k přepínání napájení mezi jednotlivými digity. K Pmod JB jsou připojeny jednotlivé skupiny segmentů a k Pmod JA jsou připojeny LED a hallův senzor. Výhodnější by bylo pouřít jednou součástku obsahující 4 digity (stejně jako na desce Nexys A7), ale Eagle takovou součástku neobsahuje. PADy 1, 2 a 3 slouží k připojení hallova senzoru. PAD1 je určen k připojení +5V protože použitý hallův senzor pracuje od 4,5V, ale na Pmod porty mají 3,3V.

hall sensor digital out <sub>PAD1</sub>+<sub>+</sub>+5V hall sensor PADZ+\_GND hall sensor AD₃+ hall sensor digital PAD4+1N +5V PAD5"\_GND **PCB** Pmod JA Port AN0 4 D12 AN1 3 A11 2 AN2 B11 AN3 1 G13 Pmod JB PCB Port 1 E15 a b 2 E16 3 D15 C

d 4 C15 7 е J17 f 8 J18 9 K15 g 10 5 **GND** 3,3V 6 **PCB Pmod JC** Port LED1 1 U12 LED2 2 V12 3 LED3 V10 LED4 V11 LED5 7 T13 U13 hall sensor Programová část Popis řešení měření času Čidlo vždy vygeneruje klesající hranu, když kolem něj projede magnet umístěný na kole setrvačníku. V top modulu vstup invertujeme, pro snažší pochopení. Tento signál značí jednu otáčku setrvačníku, který simuluje setrvačnost jízdního kola a člověka na něm. Interval mezi klesajícími hranami je tedy doba za kterou virtuální kolo urazí jednu otočku kola. Podle vzorečku v=s/t můžeme vypočítat rychlost, kde s je virtuální vzdálenost jedné otočky setrvačníku (je tedy konstantní) a t je čas mezi signálem ze senzoru. K měření času se dá použít hodinový zdroj z FPGA který má 100 MHz. Rozlišovací schopnost 10 ns je pro toto použití moc přesná a plítvali bychom zbytečně prostředky na např. 32 bitovém čitači, když můžeme použít 16 bitový. Tedy aby pro čas kolem 250 us se musí prodloužit interval mezi náběžnými hranami. Pro tento účel byla použita dělička časových hodin v podobě modulu clock\_enable, který byl využit z cvičení v předmětu DE1. Poté něčím musíme počítat jednotlivé pulzy. Zde byl využit naprogramovaný obousměrný čítač ze cvičení, který pro tento účel zjednodušíme a uděláme z něj jen jednoduchý čítač nahoru. Nasčítaný počet pulzů tedy odpovídá času, který trvala jedna otočka setrvačníku. Pro vyšší přesnost se při každé otočce vyresetuje i modul clock\_enable. Náhled do bloku s instancemi clock\_enable a cnt\_up a procesu p\_etime (elapsed time)

-- Instance (copy) of clock\_enable entity

elsif (unsigned(s\_etime) > g\_ETIME\_ZERO) then s\_etime\_local <= (others => '0');

if (s\_skip = '1') and (reset = '0') then

s\_etime\_local <= unsigned(s\_etime);</pre> s\_distance\_local <= s\_distance\_local + 1;</pre>

s\_rst\_t <= '1'; -- reset the timer and clock divider</pre>

globálním resetu se vyresetuje vše a nastaví se s\_skip, aby to další náběžnou hranu nepočítalo.

otočení setrvačníku násobeného obvodem virtuálního kola ( g\_WHEEL\_CIRCUMFERENCE ).

Náhled na kód procesu p\_calc a přiřazení dat výstupním signálům

Proces p\_etime má za úkol měřit čas, počítat vstupní hodnoty, přiřazovat naměřený čas do signálu s\_etime\_local a posílat signál s\_rst\_t a tím resetovat clk\_en1 a bin\_cnt0 . Reset proběhne vždy když příjde náběžná hrana z hallova senzoru. Také se musí kontrolovat, jestli už neuplynula moc dlouhá doba od posledního vstupu a v případě potřeby vynulovat čas v signálu s\_etime . V tomto případě je potřeba přeskočit další měření, kdeby mohla kvůli jedoucímu čítači přetéct maximální hodnota času (0.1\*2^16 ms = 6.5536 s) a mohlo by to odečíst špatnou hodnotu. Pro tento účel je zde signál s\_skip, který zaručí, že při další náběžné hraně se aktuální hodnota nebude měřit a jen proběhne reset čítače a děliče. Lokální reset s\_rst\_t trvá do dalšího clocku a dokud není vyresetovaný čitač. Při

Rychlost se počítá pomocí vzorečku v=s/t. Tedy  $v[dm/s] = g_{WHEEL\_CIRCUMFERENCE}[cm]/s_{etime}[ms/10]$ . Vzdálenost se počítá podle

Energie mezi otočením se dá rozdělit na rozdíl kinetických energií mezi otečením a aktuálnímu odporu (zátěže) potřebné pro jedno otočení. Kinetická energie se spočítá jako  $E_k = 1/2 \text{ J} \text{ }\omega^2$ . Práce je součet vykonané energie, tedy pokud budeme počítat jednotlivé díly jak

Pro zátěž počítáme s teoretickou, která bude sestávat jen z práce potřebné pro jednu otočku tedy čím vyšší úhlová rychlost, tím vyšší odpor. Výpočet  $W_o = 2\pi F_b$ . V Programu se jedná o g\_RESISTLOAD a o g\_INERTIA\_MOMENT , kde g\_INERTIA\_MOMENT musí být vynásoben

roste, tak dojdeme k celkové energii. Design programu je tímto připraven například na měření okamžitého výkonu.

s skip <= '1':

end if;

end if; end process p\_etime;

Popis kódu

if rising\_edge(hall\_sensor\_i) then

s\_rst\_t <= '1'; s\_skip <= '0'; elsif reset = '0' then

Počítání rychlosti, vzdálenosti a energie

10000π, aby odpovídal jednotkám SI.

Popis kódu

Simulace

1000

1081

end if;

clk\_en1 : entity work.clock\_enable

generic map(

g\_MAX => g\_TIME\_UNIT\_LENGHT port map( clk => clk, reset => s\_rst\_t, ce\_o => s\_en ); -- Instance (copy) of cnt\_up entity bin\_cnt0 : entity work.cnt\_up generic map( g\_CNT\_WIDTH => 16 port map(  $clk \Rightarrow clk$ reset => s\_rst\_t, en\_i => s\_en, cnt\_o => s\_etime -- Measure elapsed time using clock divider and counter. p\_etime : process(clk,hall\_sensor\_i,reset) if reset = '1' then s\_rst\_t <= '1'; s\_etime\_local <= (others => '0'); s\_distance\_local <= (others => '0'); elsif (( $s_rst_t = '1'$ ) and ( $s_etime = x"0000"$ )) then s\_rst\_t <= '0'; -- on the next clock take down the reset for timer</pre>

-- Calculate actual speed (average of last four speeds). -- Also monitor maximal speed, calories in total. p\_calc : process(hall\_sensor\_i,reset,s\_etime\_local) begin if falling\_edge(reset) then s\_speed\_local <= (others => '0'); s\_speed1\_local <= (others => '0'); s\_speed2\_local <= (others => '0'); s\_speed3\_local <= (others => '0'); s\_avg\_speed\_local <= (others => '0'); s\_max\_speed\_local <= (others => '0'); s\_inertia\_local <= (others => '0'); s\_inertia1\_local <= (others => '0'); s\_work\_local <= (others => '0'); if  $(s_etime_local = x"0000")$  and (reset = '0') then --flywheel is stopped s\_speed\_local <= (others => '0'); s\_speed1\_local <= (others => '0'); s\_speed2\_local <= (others => '0'); s\_speed3\_local <= (others => '0'); s\_avg\_speed\_local <= (others => '0'); s\_inertia\_local <= (others => '0'); end if; if rising\_edge(hall\_sensor\_i) then if reset = '0' then s\_speed3\_local <= s\_speed2\_local;</pre> s\_speed2\_local <= s\_speed1\_local;</pre> s\_speed1\_local <= s\_speed\_local;</pre> s\_speed\_local <= (g\_WHEEL\_CIRCUMFERENCE\*10000/resize(s\_etime\_local, 22)); -- m\*1e-6/s\*1e-4=cm/s</pre> s\_avg\_speed\_local <= resize(resize(s\_speed\_local+s\_speed1\_local+s\_speed2\_local+s\_speed3\_local, 24)/4, 22); if (s\_max\_speed\_local < s\_avg\_speed\_local) then</pre> s\_max\_speed\_local <= s\_avg\_speed\_local;</pre> s\_inertia1\_local <= s\_inertia\_local;</pre> s\_inertia\_local <= resize(g\_INERTIA\_MOMENT/((s\_etime\_local \* s\_etime\_local)/5000), 16);</pre> s\_work\_local <= s\_work\_local + s\_inertia\_local - s\_inertia1\_local + g\_RESISTLOAD; -- in Joules</pre> end if; end if; end process p\_calc; speed\_o <= std\_logic\_vector(resize(resize(s\_avg\_speed\_local\*36, 26)/10, 22));</pre> max\_speed\_o <= std\_logic\_vector(resize(resize(s\_max\_speed\_local\*36, 26)/10, 22));</pre> distance\_o <= std\_logic\_vector(resize(resize(s\_distance\_local \* g\_WHEEL\_CIRCUMFERENCE, 30) / 100, 22));</pre> calories\_o <= std\_logic\_vector(resize(resize(s\_work\_local\*1000, 26)/4184, 22)); -- 1cal = 4.184 Joules</pre>

Proces p\_calc v normálním chodu (není nastaven reset ani s\_etime\_local není vynulovaný) nejprve přesune všechny důležité hodnoty z minulých běhů a uvolní tak místo pro aktuální hodnotu rychlosti s\_speed\_local, poté se vypočítá průměr čtyř předchozích rychlostí s\_avg\_speed\_local a vyzkouší se, jestli to není nová nejvyšší rychlost s\_max\_speed\_local . Poté se uloží minulá kinetická energie a vypočítá se aktuální. s\_work\_local je dosavadní práce, rozdíl kinetických energií setrvačníku a přičtení g\_RESISTLOAD, protože setrvačník udělal jedno otočení. Signál reset vynuluje všechny lokální proměnné používané tímto procesem. V případě že reset není nastaven, ale čas mezi pulsy je na nule, tedymoc dlouho se čekalo na další pulz z hallova senzoru se vynulují rychlosti a kinetická energie, tedy setrvačník se virtuálně zastaví. Tento stav se musí ošetřovat zvlášť, protože dělit nulou nemůžeme a nejvyšší možná hodnota s\_etime se neblíží

Přiřazování hodnot je jen převod na jednotky vhodnější k použití v displeji. Např. z cm/s převádíme do desítek metrů za hodinu (násobíme 3.6). Vzdálenost je násobení počtu otočení setrvačníku a obvodu virtuálního kola, kde výsledek je v metrech. Jouly převádíme na kalorie,

V simulaci je simulován postupné zrychlení na 10 m/s (36 km/h) a poté prudké zastavení, kde se testuje vynulování po dlouhé době bez hrany. Mohlo by se toho testovat více např. po každém pulzu ze senzoru rychlost nebo tak, ale myslím, že vizuální kontrola výsledků je jednodušší a rychlejší. Kontroluje se tedy jen rychlost na začátku po jednom pulzu (test jestli funguje správně s\_skip ), vzdálenost po ujetých deseti metrech a rychlost při konstantní rychlosti. Dělička hodinového signálu je nastavena na tisícinu normální hodnoty, aby

Na Obrázku ze simulace můžeme vidět rozběhnutí. Odpor a moment hybnosti jsem počítal jako rychlost pohybujícího se dospělého a kola

Podle kinetické energie hmotného bodu  $E_k = 1/2 \text{ m v}^2$  (v = o/T; o - obvod kola) si můžeme odvodit jaké parametry musí mít moment setrvačnosti J. Poté můžeme nastavit odpor na nulu a pozorovat, že nám vyjde přesně vypočítaný výkon na konci rozběhu. Délka rozběhu byla volená daleko kratší než by byla v reálných podmínkách, ale účel to splní. Kvůli zaokrouhlování při dělení vždy dolů bude chyba mířit

nekonečnu, hlavně 2500000 > 2^16, tedy nikdy nevýjde nula i s tím, že dělení ve VHDL zaokrouhluje vždy dolů.

které jsou větší a častěji se používají na cvičících strojích, kde jsou v kilokaloriích.

- 80 kg a odpor jsem počítal, aby při 36 km/h byl potřebný výkon dodávaný na šlapátka 80 W.

Převádění vypočtených hodnot do formátu pro driver sedmi segmentového displeje

simulace netrvala extrémně dlouho (místo 20 s je to 20 ms).

také dolů a výsledek bude o trochu menší než očekávání.

3092 \ 3189 \ 3283 \ 3373 \ 3466 \ 3524 \ 3571

950 969 933 950 899

3238 \ 3484 \ 3617

bin\_cnt1 : entity work.cnt\_up

clk => clk, reset => s\_reset,  $en_i \Rightarrow s_en_t$ ,

-- p\_display\_selection:

begin

end if;

end if;

end if:

end if; end if;

end if;

-- p\_display\_control:

s\_reset <= '0';

end process p\_display\_selection;

p\_display\_control : process(clk)

case s\_sel\_display\_local is when  $x"0" \Rightarrow --$  speed

leds\_o <= b"10000";

dp\_o <= b"1101";</pre>

-- (8s) s\_reset sets on and then off.

if rising\_edge(button\_i) then s\_rst\_r\_local <= '0';

if falling\_edge(button\_i) then

s\_rst\_r\_local <= '1';</pre> s\_rst\_t\_local <= "00";</pre>

if rising\_edge(s\_en\_r) then

if s\_rst\_t\_local = "11" then

s\_rst\_r\_local <= '1';</pre> s\_reset <= '1';

s\_rst\_t\_local <= (others => '0');

g\_CNT\_WIDTH => 19

unsigned(cnt\_o) => s\_total\_time

p\_display\_selection : process(clk, button\_i, s\_en\_r)

if (s\_sel\_display\_local > x"3") then s\_sel\_display\_local <= x"0";</pre>

s\_rst\_t\_local <= (others => '0');

s\_rst\_t\_local <= s\_rst\_t\_local + 1;</pre>

-- Converts input number to format that is compatible with -- driver\_7seg\_4digits and displays only one number.

s\_temp\_local <= unsigned(speed\_i);</pre>

The omitted numbers is rounded down so 155.79 km -> 155.7 km.

if ((s\_temp\_local / 10000) < 1) then -- format DD.DD</pre>

elsif ((s\_temp\_local / 100000) < 1) then -- format DDD.D</pre>

data3\_o <= std\_logic\_vector(resize(s\_temp\_local / 1000, 4));</pre>

data0\_o <= std\_logic\_vector(resize(s\_temp\_local mod 10, 4));</pre>

data3\_o <= std\_logic\_vector(resize(s\_temp\_local / 10000, 4));</pre>

data2\_o <= std\_logic\_vector(resize((s\_temp\_local mod 1000) / 100, 4));</pre> data1\_o <= std\_logic\_vector(resize((s\_temp\_local mod 100) / 10, 4));</pre>

data2\_o <= std\_logic\_vector(resize((s\_temp\_local mod 10000) / 1000, 4));</pre>

if (s\_reset = '1') and rising\_edge(clk) and (s\_total\_time = 0) then

-----

-- Counter reacting for falling edge of button and when long press

s\_sel\_display\_local <= s\_sel\_display\_local + 1;</pre>

generic map(

port map(

42 \ 45 \ 47 \ 58 \ 52 \ 55 \ 57 \ 68 \ 62 \ 65

850 / 913 / 950 | 989 / 1052 / 1057 / 1061 | 1066 / 1071 / 1076

58 3824 3977 4139 4403 4423 4443 4463 4483 4503

Na druhém obrázku vidíme správnou funkci vynulování rychlostí.

Kód ve display\_control by se dal rozdělit na tři části: 1. celkový čas 2. přepínání obrazovek a reset 3. konverze vstupních signálů do formátu zpracovatelného driverem na sedmi segmentový displej Se počítá pomocí instance clk\_en2 a p\_total\_time , každou sekundu se přičte jednička do s\_total\_time , při reset se čas vynuluje. Přepínání obrazovek obsluhuje instance clk\_en3 a bin\_cnt1 a proces p\_display\_selection, který při stisknutí tlačítka button\_i přepne na další obrazovku, pokud je tlačítko stisknuto jen krátce a pokud dlouze více než osm sekund, tak nastaví signál reset na jedničku a po proběhnutí resetu ho zase nastaví na nulu. Doba stisku se počítá pomocí clk\_en3 a bin\_cnt1 . Přepnutí obrazovky nastane až se sestupnou hranou. V procesu p\_display\_control se pomocí operací děleno a modulo rozdělují jednotlivé dekadické hodnoty a přiřazují se dataX\_o, kde je uložená vždy jedna číslice pro displej. datao\_o je číslice na sedmisegmentu nejvíce v pravo data3\_o je číslice nejvíce vlevo. Náhled na kód v architecture display\_control.vhdl -- Instance (copy) of clock enable entity clk\_en2 : entity work.clock\_enable generic map( g\_MAX => g\_clk\_div\_sec ) port map( clk => clk, reset => s\_reset, ce\_o => s\_en\_t Instance (copy) of clock\_enable entity clk\_en3 : entity work.clock\_enable generic map( g\_MAX => g\_time\_for\_reset ) port map( clk => clk, reset => s rst r local,  $ce_o \Rightarrow s_en_r$ ); -- Instance (copy) of cnt\_up entity

data1\_o <= std\_logic\_vector(resize((s\_temp\_local mod 1000) / 100, 4));</pre> data0\_o <= std\_logic\_vector(resize((s\_temp\_local mod 100) / 10, 4));</pre> elsif ((s\_temp\_local / 1000000) < 1) then -- format DDDD</pre> dp\_o <= b"1111";</pre> data3\_o <= std\_logic\_vector(resize(s\_temp\_local / 100000, 4));</pre> data2\_o <= std\_logic\_vector(resize((s\_temp\_local mod 100000) / 10000, 4));</pre> data1\_o <= std\_logic\_vector(resize((s\_temp\_local mod 10000) / 1000, 4));</pre> data0\_o <= std\_logic\_vector(resize((s\_temp\_local mod 1000) / 100, 4));</pre> else -- for higher numbers display only 9999 dp\_o <= b"1111";</pre> data3\_o <= x"9"; data2\_o <= x"9"; data1\_o <= x"9"; data0\_o <= x"9"; end if; when x"1" => -- distance s\_temp\_local <= unsigned(distance\_i);</pre> leds\_o <= b"01000"; if ((s\_temp\_local / 10000) < 1) then -- format D.DDD</pre> dp\_o <= b"0111"; data3\_o <= std\_logic\_vector(resize(s\_temp\_local / 1000, 4));</pre> data2\_o <= std\_logic\_vector(resize((s\_temp\_local mod 1000) / 100, 4));</pre> data1\_o <= std\_logic\_vector(resize((s\_temp\_local mod 100) / 10, 4));</pre> data0\_o <= std\_logic\_vector(resize(s\_temp\_local mod 10, 4));</pre> elsif ((s\_temp\_local / 100000) < 1) then -- format DD.DD</pre> dp\_o <= b"1011"; data3\_o <= std\_logic\_vector(resize(s\_temp\_local / 10000, 4));</pre> data2\_o <= std\_logic\_vector(resize((s\_temp\_local mod 10000) / 1000, 4));</pre> data1\_o <= std\_logic\_vector(resize((s\_temp\_local mod 1000) / 100, 4));</pre> data0\_o <= std\_logic\_vector(resize((s\_temp\_local mod 100) / 10, 4));</pre> elsif ((s\_temp\_local / 1000000) < 1) then -- format DDD.D</pre> dp\_o <= b"1101"; data3\_o <= std\_logic\_vector(resize(s\_temp\_local / 100000, 4));</pre> data2\_o <= std\_logic\_vector(resize((s\_temp\_local mod 100000) / 10000, 4));</pre> data1\_o <= std\_logic\_vector(resize((s\_temp\_local mod 10000) / 1000, 4));</pre> data0\_o <= std\_logic\_vector(resize((s\_temp\_local mod 1000) / 100, 4));</pre> elsif ((s\_temp\_local / 10000000) < 1) then -- format DDDD dp\_o <= b"1111"; data3\_o <= std\_logic\_vector(resize(s\_temp\_local / 1000000, 4));</pre> data2\_o <= std\_logic\_vector(resize((s\_temp\_local mod 1000000) / 100000, 4));</pre> data1\_o <= std\_logic\_vector(resize((s\_temp\_local mod 100000) / 10000, 4));</pre> data0\_o <= std\_logic\_vector(resize((s\_temp\_local mod 10000) / 1000, 4));</pre> end if; when x"2" => -- kcalories s\_temp\_local <= unsigned(calories\_i);</pre> leds\_o <= b"00100";</pre> if ((s\_temp\_local / 10000) < 1) then -- format D.DDD</pre> dp\_o <= b"0111";</pre> data3\_o <= std\_logic\_vector(resize(s\_temp\_local / 1000, 4));</pre> data2\_o <= std\_logic\_vector(resize((s\_temp\_local mod 1000) / 100, 4));</pre> data1\_o <= std\_logic\_vector(resize((s\_temp\_local mod 100) / 10, 4));</pre> data0\_o <= std\_logic\_vector(resize(s\_temp\_local mod 10, 4));</pre> elsif ((s\_temp\_local / 100000) < 1) then -- format DD.DD</pre> dp\_o <= b"1011";</pre> data3\_o <= std\_logic\_vector(resize(s\_temp\_local / 10000, 4));</pre> data2\_o <= std\_logic\_vector(resize((s\_temp\_local mod 10000) / 1000, 4));</pre> data1\_o <= std\_logic\_vector(resize((s\_temp\_local mod 1000) / 100, 4));</pre> data0\_o <= std\_logic\_vector(resize((s\_temp\_local mod 100) / 10, 4));</pre> elsif ((s\_temp\_local / 1000000) < 1) then -- format DDD.D</pre> dp\_o <= b"1101";</pre> data3\_o <= std\_logic\_vector(resize(s\_temp\_local / 100000, 4));</pre> data2\_o <= std\_logic\_vector(resize((s\_temp\_local mod 100000) / 10000, 4));</pre> data1\_o <= std\_logic\_vector(resize((s\_temp\_local mod 10000) / 1000, 4));</pre> data0\_o <= std\_logic\_vector(resize((s\_temp\_local mod 1000) / 100, 4));</pre> elsif ((s\_temp\_local / 10000000) < 1) then -- format DDDD</pre> dp\_o <= b"1111";</pre> data3\_o <= std\_logic\_vector(resize(s\_temp\_local / 1000000, 4));</pre> data2\_o <= std\_logic\_vector(resize((s\_temp\_local mod 1000000) / 100000, 4));</pre> data1\_o <= std\_logic\_vector(resize((s\_temp\_local mod 100000) / 10000, 4));</pre> data0\_o <= std\_logic\_vector(resize((s\_temp\_local mod 10000) / 1000, 4));</pre> when  $x"3" \Rightarrow -- \text{ total time}$ leds\_o <= b"00010"; if (s\_total\_time < 3600) then -- MM.SS</pre> dp\_o <= b"1011";</pre> data3\_o <= std\_logic\_vector(resize((s\_total\_time / 60) / 10, 4));</pre> data2\_o <= std\_logic\_vector(resize((s\_total\_time / 60) mod 10, 4));</pre> data1\_o <= std\_logic\_vector(resize((s\_total\_time mod 60) / 10, 4));</pre> data0\_o <= std\_logic\_vector(resize((s\_total\_time mod 60) mod 10, 4));</pre> else -- HH.MM dp\_o <= b"1011";</pre> data3\_o <= std\_logic\_vector(resize((s\_total\_time / 3600) / 10, 4));</pre> data2\_o <= std\_logic\_vector(resize((s\_total\_time / 3600) mod 10, 4));</pre> data1\_o <= std\_logic\_vector(resize(((s\_total\_time mod 3600) / 60) / 10, 4));</pre> data0\_o <= std\_logic\_vector(resize(((s\_total\_time mod 3600) / 60) mod 10, 4));</pre> end if; when  $x"4" \Rightarrow -- maximal speed$ s\_temp\_local <= unsigned(max\_speed\_i);</pre> leds\_o <= b"00001"; if ((s\_temp\_local / 10000) < 1) then -- format DD.DD</pre> dp\_o <= b"1011";</pre> data3\_o <= std\_logic\_vector(resize(s\_temp\_local / 1000, 4));</pre> data2\_o <= std\_logic\_vector(resize((s\_temp\_local mod 1000) / 100, 4));</pre> data1\_o <= std\_logic\_vector(resize((s\_temp\_local mod 100) / 10, 4));</pre> data0\_o <= std\_logic\_vector(resize(s\_temp\_local mod 10, 4));</pre> elsif ((s\_temp\_local / 100000) < 1) then -- format DDD.D</pre> dp\_o <= b"1101";</pre> data3\_o <= std\_logic\_vector(resize(s\_temp\_local / 10000, 4));</pre>

₩ s\_leds[4:0] **₩** s\_dp[3:0] **⊌** s\_data2[3:0] **₩** s\_data1[3:0] s\_en\_t s\_en\_r

BTN 0

) U13

CLK 100MHz

Simulace

s\_buttor s\_reset ed[21:0]

₩ s distance[21:0]

😽 s\_calories[21:0]

₩ s\_sel\_display\_local[3: ♥ s\_temp\_local[21:0] Simulace kompletního projektu je zbytečná, protože výstupní hodnoty pro sedmisegmentový display jsou špatně čitelné a vyzkoušení veškeré funkcionality dohromady velmi náročné. Blokové schéma top.vhdl

display\_control\_o

leds\_o[4:0]

data3 o[3:0]

data2\_o[3:0]

data1\_o[3:0]

data0 o[3:0

dp\_o[3:0]

s\_dp

button i

speed\_i[21:0]

distance i[21:0]

calories\_i[21:0]

max\_speed\_i[21:0]

speed\_o[21:0

distance o[21:0]

calories\_o[21:0

max\_speed\_o[21:0

reset

data2\_o <= std\_logic\_vector(resize((s\_temp\_local mod 10000) / 1000, 4));</pre> data1\_o <= std\_logic\_vector(resize((s\_temp\_local mod 1000) / 100, 4));</pre> data0\_o <= std\_logic\_vector(resize((s\_temp\_local mod 100) / 10, 4));</pre>

data2\_o <= std\_logic\_vector(resize((s\_temp\_local mod 100000) / 10000, 4));</pre> data1\_o <= std\_logic\_vector(resize((s\_temp\_local mod 10000) / 1000, 4));</pre> data0\_o <= std\_logic\_vector(resize((s\_temp\_local mod 1000) / 100, 4));</pre>

Simulace probíhá tím, že se testuje zobrazení různě velkých čísel a potom se pomocí assertů kontroluje výstupní hodnota. Postupně se

1571558

driver\_7seg\_4digits\_0

dp\_c

seg\_o[6:0]

dig\_o[3:0]

dp\_i[3:0]

data3 i[3:0]

data2\_i[3:0]

data1\_i[3:0]

ata0\_i[3:0]

U12,V12,V10,V11,T13

D12,A11,B11,G13

E15,E16,D15,C15 J17,J18,K15

) J15

Poznámka: Simulace pracuje v mnohem kratším časovém intervalu a tedy výstupní signály nejsou naprosto čisté, ale normálně se

zkouší také různé režimy zobrazení a na závěr se testuje dlouhé podržení pro nastavení reset a potom jeho vypnutí.

data3\_o <= std\_logic\_vector(resize(s\_temp\_local / 100000, 4));</pre>

elsif ((s\_temp\_local / 1000000) < 1) then -- format DDDD</pre>

else -- for higher numbers display only 9999

neproklikává displej takovou rychlostí, přebliknutí hodnot z minulého displeje není kritické

when others => -- there are no other states.

dp\_o <= b"1111";</pre>

dp\_o <= b"1111"; data3\_o <= x"9"; data2\_o <= x"9"; data1\_o <= x"9";</pre> data0 o <= x"9";

end case;

reset <= s\_reset;

157155

end process p\_display\_control;

Video Link na video prezentace Reference 1. https://store.digilentinc.com/arty-a7-artix-7-fpga-development-board/ 2. https://reference.digilentinc.com/reference/programmable-logic/arty-a7/reference-manual

**TODO** 

3. arty\_a7\_sch.pdf 4. 44E datasheet