Teoretický úvod – laboratorní úloha číslo 2

Návrh převodníků kódů v jazyce VHDL, realizace převodníku z kódu BCD do kódu 7segmentového displeje

1. Číselné kódy a převodníky kódů

Číselný kód představuje soubor pravidel pro jednoznačné přiřazení daného znaku (symbolu, čísla) hodnotě daného čísla na určené pozici. Účelem využití různého způsobu kódování je využití jejich vhodných vlastností pro přenos informací, pro kódování a zobrazení určitého znaku na displejích apod. Mezi nejznámější skupiny binárních kódů patří:

- Váhové kódy – každému řádovému místu odpovídá jiný váhový koeficient. Příklady váhových kódů: 5421, 84-2-1, 2421, 3331 aj.
- Binary Coded Decimal, BCD kód – každá desítková číslice je v něm vyjádřena jako čtyřmístné binární číslo. Jedná se o váhový kód s váhami 8421.
- Kód F+3 – kódová složka představující v kódu F+3 nulu odpovídá v binárním zápise hodnotě 3.
- Kód se změnou v *n* řádech – kód, v němž se dvě sousední čísla liší vzájemně od sebe vždy na *n* řádových místech. Nejvýznamnějším kódem je Grayův kód, v němž se dvě sousední čísla vzájemně liší právě v jednom řádovém místě.
- Kód pro zobrazení znaků na 7segmentovém displeji – 7segmentový displej se skládá ze 7 samostatných světelných segmentů vhodně rozmístěných tak, aby pomocí kombinací jeho rozsvícených a zhasnutých segmentů bylo možno zobrazovat dekadické číslice a vybrané znaky abecedy. Pro vyjádření a ovládání zhasnutí/rozsvícení těchto segmentů lze s výhodou použít binární typ kódu – kód 7segmentového displeje[1].

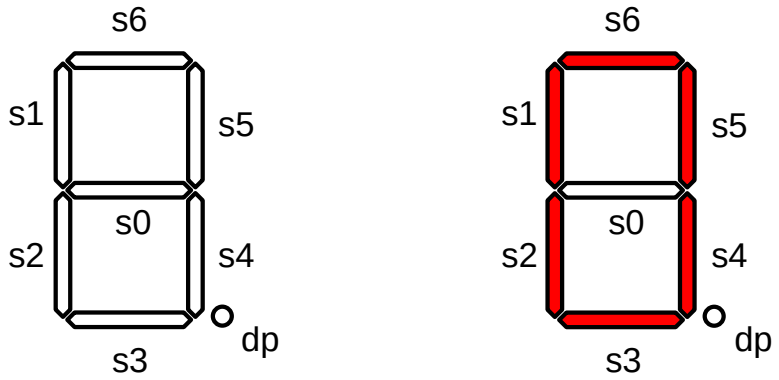
[1] Kromě 7segmentového displeje se lze v praxi setkat i s 9segmentovým, 14segmentovým, 15segmentovým a 16segmentovým displejem. Kromě vlastních segmentů pro zobrazení znaku obsahuje displej často ještě segment pro tečku (obvykle desetinnou), někdy též dvojtečku či další doplňkové znaky.

Tab. č. 1 ukazuje přehled vybraných binárních kódů.

Tab. č. 1: Převodní tabulka vybraných kódů.

$N_{(10)}$	BCD kód	Kód 5421	Kód 84-2-1	Grayův kód	Kód F+3	Kód 7segm. displeje
0	0000	0000	0000	0000	0011	1000000
1	0001	0001	0111	0001	0100	1111001
2	0010	0010	0110	0011	0101	0100100
3	0011	0011	0101	0010	0110	0110000
4	0100	0100	0100	0110	0111	0011001
5	0101	1000	1011	0111	1000	0010010
6	0110	1001	1010	0101	1001	0000010
7	0111	1010	1001	0100	1010	1111000
8	1000	1011	1000	1100	1011	0000000
9	1001	1100	1111	1101	1100	0010000

Zaměřme se nyní na kód a zapojení 7segmentového displeje. Následující obrázek představuje typické rozložení segmentů a jejich zapojení a označení v rámci 7segmentového displeje. Toto zapojení je použito i pro displej na přípravku DE10-Lite.



Obr. č. 1: Zapojení a značení segmentů 7segmentového displeje a ukázka zobrazení číslice 0.

Jednotlivé segmenty jsou obvykle značeny písmenem „s“ (segment) a číslicí typicky 0 až 6, segment pro desetinnou tečku se typicky označuje jako „dp“ (decimal point). Z hlediska elektrického zapojení v případě LED segmentových displejů se v praxi vyskytují zapojení s tzv. společnou katodou a se společnou anodou. **Zapojení segmentů se společnou anodou, které je použito na přípravku DE10-Lite**, znamená, že jednotlivé anody všech LED segmentů jednoho displeje jsou spojeny do jedné společné anody, kterou pak můžeme připojit ke kladnému napájecímu napětí. Pokud tedy požadujeme rozsvícení daného segmentu, na jeho katodu připojíme zem (GND), tedy hodnotu logická 0, zatímco pokud má být daný segment zhasnutý, na jeho katodu připojíme kladné napětí (Vcc) a tedy hodnotu logická 1. Pro zobrazení číslice 0 na obrázku výše bychom tedy zapsali kód pro buzení 7segmentového displeje: 1000000 v pořadí *s6 s5 s4 s3 s2 s1 s0* (a neuvažujeme desetinnou tečku).
Tab. 1 má tento stav na prvním řádku.

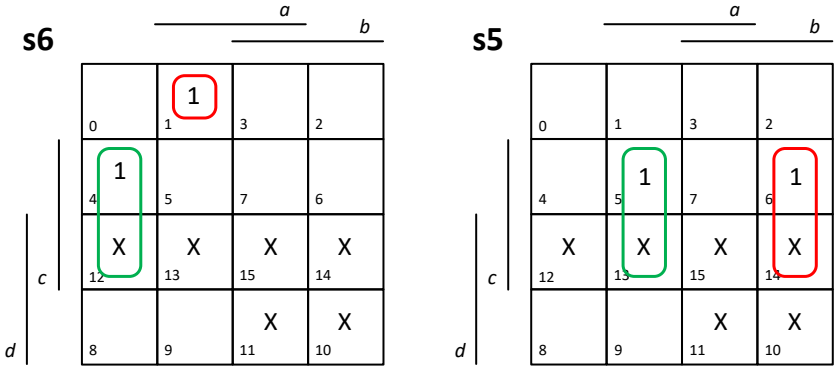
2. Postup návrhu převodníku z kódu BCD na kód 7segmentového displeje

Uveďme si postup návrhu převodníku kódu z BCD kódu do kódu pro buzení 7segmentového displeje. Tab. č. 2, ze které vyjdeme, obsahuje převodní vztahy mezi kódem BCD a kódem 7segmentového displeje se značením výstupních funkcí (segmentů displeje) odpovídající obrázku výše. Obr. č. 1 ukazuje, jak každému řádovému místu kódu BCD přiřadíme vstupní proměnnou *a*, *b*, *c*, *d* a označení výstupů (*s0*, *s1*, *s2*, *s3*, *s4*, *s5* a *s6*). Obvykle je zvykem přiřadit proměnnou *a* na nejnižší řádové místo, tzv. LSB (Least Significant Bit) a v tomto případě proměnnou *d* na nejvyšší řádové místo, tzv. MSB (Most Significant Bit), obdobně pak v kódu pro 7segmentový displej uvažujeme *s6* na pozici nejnižšího řádového místa v kódu a *s0* naopak na pozici nejvyššího řádového místa 7segmentového kódu.

Tab. č. 2: Převodní tabulka mezi BCD kódem a kódem 7segmentového displeje se společnou anodou.

$N_{(10)}$	Kód 7segmentového displeje se společnou anodou						
	s0	s1	s2	s3	s4	s5	s6
0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	1
2	0	1	0	0	1	0	0
3	0	1	1	0	0	0	0
4	0	0	1	1	0	0	1
5	0	0	1	0	0	1	0
6	0	0	0	0	0	1	0
7	1	1	1	1	0	0	0
8	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0

Podle tabulky můžeme jednoduše zapsat Karnaughovy mapy pro jednotlivé výstupní funkce (segmenty) *s0*, *s1*, *s2*, *s3*, *s4*, *s5* a *s6* a provést jejich minimalizace a získat např. minimální disjunktní formu pro každou z nich. Vzhledem k tomu, že pro zobrazení na 7segmentovém displeji uvažujeme pouze číslice 0–9 na každém jeho řádovém místě, budou v políčkách Karnaughovy mapy pro indexy 10–15 neurčité stavy X. Obr. č. 2 představuje ukázku minimalizace pomocí Karnaughovy mapy pro výstupní funkce (segmenty) *s6* a *s5*.



Obr. č. 2: Karnaughovy mapy pro minimalizaci a realizaci výstupů pro segmenty s6 a s5 7segmentového displeje.

Následně můžeme vyjádřit minimální disjunktní tvary jednotlivých výstupních funkcí pro segmenty displeje a upravit je pro realizaci pomocí hradel NAND, jak ukazuje příklad pro *s5* a *s6*:

$$s6 = \bar{a}\bar{b}c \vee a\bar{b}\bar{c}\bar{d} = \overline{\overline{\bar{a}\bar{b}c} \cdot \overline{a\bar{b}\bar{c}\bar{d}}}\,,$$
$$s5 = a\bar{b}c \vee \bar{a}bc = \overline{\overline{a\bar{b}c} \cdot \overline{\bar{a}bc}}\,. \qquad (1)$$

Stejným způsobem můžeme pokračovat pro zbylé segmenty (výstupy) *s4, s3, s2, s1* a *s0*.

3. Realizace převodníků v jazyce VHDL

V této laboratorní úloze budeme převodník realizovat jednak pomocí schematického editoru, a také přímo v jazyce VHDL pomocí tzv. `Dataflow` či `RTL` (Register Transfer Level) popisu. VHDL patří do rodiny tzv. HDL (Hardware Description Language) jazyků, jak z názvu vyplývá, jeho úkolem je popsat daný obvod, jeho strukturu, chování, operace se vstupními proměnnými. Tento popis v případě VHDL může být tzv. behaviorální, RTL (dataflow) a strukturální, podle nich jsou rovněž pojmenovány metody popisu a syntézy logických obvodů v jazyce VHDL. **Pro realizaci jednoduchých kombinačních obvodů je často výhodné použít právě `RTL` (dataflow) metodu.** Ta je založena na popisu „toku“, jak se vstupní logická proměnná průchodem obvodu a kombinacemi s ostatními vstupními proměnnými dostane na daný výstup. **V praxi tomu obvykle odpovídá algebraický zápis výstupní funkce obvodu pomocí vstupních proměnných za použití pravidel Booleovy algebry.** V jazyce VHDL jsou za tímto účelem definovány základní logické funkce (operátory):

- `NOT` – negace,
- `AND` – logický součin,
- `OR` – logický součet,
- `NAND` – negovaný logický součin,
- `NOR` – negovaný logický součet,
- `XOR` – exkluzivní logický součet,
- `XNOR` – negovaný exkluzivní logický součet.

Odlišností v jazyce VHDL je, že na rozdíl od standardních pravidel algebry, **nejvyšší prioritu při vykonávání uvedených operací má negace `NOT` a všechny zbývající operace mají nižší a stejnou prioritu**, tedy i logické násobení a logické sčítání. Jazyk VHDL vyhodnocuje zapsané výrazy ve směru zleva doprava a při zápisu logické funkce je potřeba rozlišit priority pomocí kulatých závorek. V jazyce VHDL je dále nutné, pokud se v rámci jednoho výrazu vyskytují různé logické operátory s výjimkou negace `NOT`, aby tyto operátory byly vždy odděleny pomocí závorek a bylo tedy na první pohled patrné, která operace má prioritu. Uveďme několik příkladů složených funkcí a jejich interpretace.

Tab. č. 3 (ve druhém sloupečku) má některé závorky ponechány pro lepší čitelnost. Jazyk VHDL obecně není tzv. „case-sensitive“ jazyk, nerozlišuje se tedy psaní velkých a malých písmen v jednotlivých příkazech ani názvech proměnných, v tomto textu budou pro přehlednost příkazy jazyka zapsány tučným fontem, názvy proměnných, portů, signálů apod. netučným.

Tab. č. 3: Příklady zápisu logických funkcí v jazyce VHDL.

Zápis v jazyce VHDL	Skutečně realizovaná operace
`not a and b`	$\bar{a}\cdot b$
`not a and not b`	$\bar{a}\cdot\bar{b}$
`a or b and c`	Chyba, ve VHDL nelze!
`(a or b) and c`	$(a\vee b)\cdot c$
`a or (b and c)`	$a\vee(b\cdot c)$
`a xor (b and not (c or d))`	$a\oplus\left(b\cdot\overline{(c\vee d)}\right)$
`(a xor b) and not (c or d)`	$(a\oplus b)\cdot\overline{(c\vee d)}$

V této laboratorní úloze se ještě stručně zmíníme o základní struktuře typického modulu v jazyce VHDL o datových typech v jazyce VHDL a portech (vstupech, výstupech).

Jeden samostatný soubor obsahující kód v jazyce VHDL se obvykle označuje jako modul – nebudeme nyní uvažovat samostatné knihovny (libraries) a balíčky (packages). Pro jednoduchost budeme uvažovat, že tento modul obsahuje jednu tzv. entitu (`entity`); ve VHDL lze v rámci jednoho modulu deklarovat více entit. Entitou se v jazyce VHDL označuje samostatný objekt, kterým může být například jedno samostatné logické hradlo, ale i složitý obvod složený z velkého množství různých hradel, klopných obvodů, jiných samostatných obvodů apod. Každá entita obsahuje dvě základní části (deklarace) – porty (`port`) a architekturu (`architecture`). Port představuje rozhraní, kterým daná entita komunikuje navenek, jedná se tedy o vstupy, výstupy obvodu. V jazyce VHDL jsou definovány 4 typy portů, tzv. vstupy (`in`), výstupy (`out`), registrové výstupy (`buffer`) a vstupně-výstupní porty (`inout`) a vždy při deklaraci daného portu je nutné uvést jeho typ (směr)[2]. Pro potřeby této laboratorní úlohy s návrhem převodníku si vystačíme pouze s porty `in` a `out`, jejich funkce vyplývají z jejich názvů, dalšími typy se nyní zabývat nebudeme. Při deklaraci portu je kromě jeho typu (směru) nutné uvést, jakého je datového typu, tedy jaké hodnoty, v jakém tvaru (formátu) může daný port přenášet. V jazyce VHDL je obecně definováno velké množství různých datových typů, namátkou např. číselné typy `integer`, `real`, `natural`, logické typy `std_logic`, `bit`, `boolean`, znakové typy (textové) `string`, `char`, sběrnice a pole `std_logic_vector`, `array` apod., které mají různá využití, uživateli je dokonce umožněno definovat si svůj vlastní datový typ (výčtem či zúžením existujícího). V rámci této laboratorní úlohy využijeme základní logický datový typ, kterým je `std_logic`, tzv. standard logic, definovaný výčtem:

[2] Ve VHDL je definován i tzv. `linkage` port, ten slouží pro propojení entity VHDL s jiným obvodem realizovaným jiným kódem/způsobem a tento typ portu nebudeme dále uvažovat.

```
type std_logic is ('0','1','H','L','W','U','X','Z','-');
```

Ten obsahuje základní sadu logických hodnot – logická 0 `'0'`, logická 1 `'1'`, neznámý stav `'X'`, nedefinovaný stav `'U'`, neurčitý stav `'-'`, stav vysoké impedance `'Z'` a tzv. slabé (weak) hodnoty, které se jen s určitou pravděpodobností blíží některé logické hodnotě (`'H','L','W'`). Řadu těchto hodnot lze využít pouze pro simulace a nikoliv pro syntézu vlastních logických obvodů. Pro odlišení se logické hodnoty v jazyce VHDL zapisují do apostrofů, tedy logická 1 je `'1'`, zatímco číslice (`integer`) 1 je pouze 1. Pokud sdružujeme několik logických hodnot do sběrnice a vytváříme tzv. vektor (`std_logic_vector`), zapisuje se takový vektor logických hodnot pomocí uvozovek, např. `"1111"`. Kromě portů je i v případě deklarace tzv. signálů, proměnných a konstant v jazyce VHDL nutné uvést jejich datový typ (ty ale v této laboratorní úloze nepotřebujeme). Kromě deklarace portů pak entita vždy obsahuje i tzv. architekturu (`architecture`). To je vlastní část VHDL modulu, která popisuje právě chování entity (`Behavioral`), strukturu entity (`Structural`) či operace se vstupy na výstup entity (`Dataflow`, `RTL`). V jazyce VHDL je navíc povoleno, aby jedna entita obsahovala klidně několik rozdílných architektur, vždy však musí být specifikováno, která architektura se má při jaké podmínce použít (pokud není specifikováno, vybere se automaticky poslední architektura v entitě). Obvykle je totiž možné popsat jeden obvod či jeho funkci (chování, strukturu) více různými způsoby. To je výhodné například v situaci, kdy jeden popis je vhodnější pro přímou implementaci obvodu do FPGA pole, jiný je třeba výhodnější pro simulaci funkce obvodu a jeho testování apod.

Na závěr této kapitoly se vrátíme k původnímu úkolu realizovat převodník z BCD kódu do kódu 7segmentového displeje pomocí dataflow (RTL) popisu v jazyce VHDL. V ukázkovém VHDL kódu (viz níže) se omezíme na zápis pouze výstupů (segmentů) *s6* a *s5*, zbylé výstupy (segmenty) *s4*, *s3*, *s2*, *s1* a *s0* bychom zapsali obdobným způsobem. S výhodou použijeme připravené minimální disjunktní tvary podle vztahů (1). Entita převodníku obsahuje tedy 4 vstupy: `a`, `b`, `c`, `d` a 7 výstupů: `s0`, `s1`, `s2`, `s3`, `s4`, `s5`, `s6` všechny vstupy a výstupy jsou základního logického typu `std_logic`.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity BCD7segmentVHDL is
port (a,b,c,d : in std_logic;
      s0,s1,s2,s3,s4,s5,s6 : out std_logic);
end BCD7segmentVHDL;

architecture RTL of BCD7segmentVHDL is
begin
s6 <= (not a and not b and c) or (a and not b and not c and not
d);
s5 <= (a and not b and c) or (not a and b and c);
s4 <=…;
s3 <=…;
s2 <=…;
s1 <=…;
s0 <=…;
end RTL;
```

Po deklaraci nezbytných knihoven (použitá knihovna obsahuje jádro jazyka VHDL) následuje deklarace názvu entity (`BCD7segmentVHDL`) a dále sekce `port` s deklarací portů entity převodníku, deklarace entity je ukončena klíčovým slovem `end` s názvem entity. Dále je deklarována architektura s názvem `RTL` (pojmenovat lze architekturu samozřejmě libovolně bez použití diakritiky, mezer, speciálních znaků), která je přiřazena dané entitě (`architecture RTL of BCD7segmentVHDL is`). V architektuře jsou pak pouze přepsány původní minimalizované disjunktní tvary jednotlivých výstupů (segmentů) dle vztahů (1), pro přiřazení hodnoty do výstupního portu slouží v jazyce VHDL symbol <=, směr šipky určuje směr portu. Na konci je již architektura zakončena klíčovým slovem `end` a názvem architektury. Mezery a odřádkování při zápisu kódu v jazyce VHDL obvykle nehrají roli, jsou použity pouze pro zlepšení čitelnosti kódu.