Efektivní optimalizace kombinačních obvodů

V první kapitole jste se seznámili s teoretickými základy digitální elektroniky: manipulací s binárními daty pomocí šesti různých jednoduchých operací. S těmito znalostmi už máte dostatek informací k vytváření velmi složitých operací, kdy na vstup přivedete množství různých bitů. Zde pravděpodobně narazíte na problém, kdy obvody nebudou optimalizovány ve smyslu minimálního počtu hradel, rychlosti odezvy na vstupy digitálního elektronického obvodu, a také musíte řešit, zda je daný obvod optimalizovaný pro technologii, kterou bude implementován.

Tyto tři parametry tvoří základní měřítka pro určení, zda je obvod efektivně optimalizován. Jasným ukazatelem by měl být počet hradel a měli byste si uvědomit, že s rostoucím počtem hradel roste i počet čipů a cena implementace obvodu, stejně jako složitost instalace vodičů. Teď vidíte propojení logických hradel jen jako černé linky na papíru, ale jestliže se pokusíte instalovat vodiče do vlastních navržených obvodů, zjistíte, že zjednodušením vedení v obvodu často snížíte cenu více, než snížení počtu čipů původně naznačovalo. Malé vylepšení spletitých obvodů může mít překvapivý dopad na celkovou cenu aplikace. Může se stát, že eliminací 1 % hradel v aplikaci ušetříte 10.20 % z celkové ceny produktu. Tyto úspory jsou výsledkem schopnosti postavit obvod na desce plošných spojů, která je menší nebo která má méně vrstev (to může velmi dramaticky snížit výslednou cenu výrobku). Jestliže je aplikace navržena pro použití technologie programovatelné logiky, můžete také zjistit, že optimalizace obvodu vám umožní použít v návrhu levnější čipy. Z menšího počtu hradel v aplikaci vyplývá i menší spotřeba elektrické energie, nižší požadavky na chlazení a menší zdroj elektrické energie.

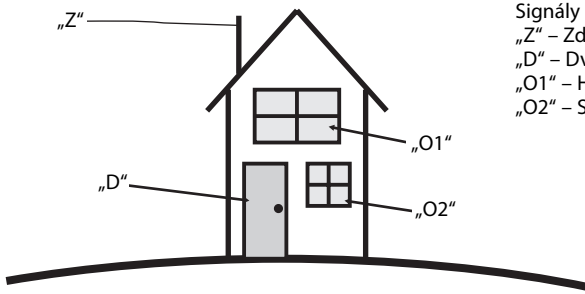
Rychlost průchodu signálu hradlem není nekonečná; standardní TTL logika potřebuje k průchodu přes hradlo NAND 8 biliontin sekundy (nazývaných nanosekundy, zkratka ns). K vyjádření této doby se užívá termín *hradlové zpoždění*. Snížením počtu hradel, přes která musí signál projít, na polovinu (čímž se na polovinu sníží počet hradlových zpoždění) se zdvojnásobí rychlost odezvy na změnu hodnoty vstupu. Jak budete postupně pracovat se stále složitějšími obvody, budete kvůli rychlosti nuceni stále častěji obvody optimalizovat anebo používat rychlejší (a obecně dražší) technologie.

Poslední parametr, nazývaný *technologická optimalizace*, se ve srovnání s předchozími dvěma může zdát na první pohled neurčitý, navíc jeho měření se provádí pomocí dalších dvou parametrů. Nicméně pokud pracujete s fyzickými zařízeními, je to nejdůležitější faktor pro správnou

optimalizaci vaší aplikace. Před prohlášením vašeho obvodu za hotový byste se měli zaměřit na technologii, pomocí níž bude váš obvod implementován, a hledat vhodnou optimalizaci ke snížení počtu hradel a hradlových zpoždění vyžadovaných aplikací.

Zvažte také využití logické optimalizace jako rekurzivní operace k opakované optimalizaci všech různých parametrů a měření. Jakmile jste specifikovali potřebné logické funkce, měli byste se soustředit na jejich implementaci do skutečného obvodu. A jakmile je převedete do skutečného obvodu, vraťte se zpátky a vyhledávejte příležitosti ke snížení počtu hradel, zrychlení průchodu signálu hradlem a znovu zkoumejte technologické optimalizace. Takto pokračujte, dokud nebudete spokojeni s konečným výsledkem.

Pro snazší pochopení tématu této kapitoly se podíváme na praktický příklad, a to jednoduché poplašné domovní zařízení. Na obrázku 2.1 vidíte zjednodušený náčrtek domu s dvěma okny, dveřmi a zdrojem energie k ovládání. Senzory na oknech, dveřích a zdroji zprostředkovávají data poplašného systému. Při návrhu poplašného systému byla vytvořena tabulka (Tabulka 2.1) se všemi kombinacemi, které mohou spustit poplach zvukovou signalizací. Jak je vidno z obrázku 2.1, vstupy zařízení jsou pozitivně aktivní, proto jejich aktivitu vyjádříme pomocí „1".



Signály alarmu jsou pozitivně aktivní:
„Z" – Zdroj je vyřazen
„D" – Dveře jsou otevřeny
„O1" – Horní okno je otevřené
„O2" – Spodní okno je otevřené

Obrázek 2.1: Logika domácího alarmu

Tabulka: 2.1: Pravdivostní logika domácího alarmu

Z	D	O1	O2	Odpověď alarmu
0	0	0	0	
0	0	0	1	
0	0	1	0	Poplach
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	Poplach
0	1	1	1	
1	0	0	0	
1	0	0	1	Poplach
1	0	1	0	Poplach
1	0	1	1	Poplach
1	1	0	0	
1	1	0	1	Poplach
1	1	1	0	Poplach
1	1	1	1	Poplach

V tomto fiktivním domě předpokládáme, že horní okno („O1“) není nikdy otevřené – kdyby bylo otevřeno, spustil by se poplach. Současně, když selže napájení a nejméně jedno z oken je otevřené, potom selhalo poplašné zařízení; tento případ by nastal odstřižením přívodu napájení do domu a násilným otevřením okna. V tabulce 2.1 vidíte případy spuštění poplachu a můžete si všimnout, že spuštění je podmíněno jednou nebo až třemi událostmi dohromady.

Po vytvoření tabulky byste si měli pro funkci sestavit rovnici typu součet součinů:

```
Stav alarmu = (!Z . !D . O1 . !O2)
+(!Z . D . O1 . !O2)
+(Z . !D . !O1 . O2)
+(Z . !D . O1 . !O2)
+(Z . !D . O1 . O2)
+(Z . D . !O1 . O2)
+(Z . D . O1 . !O2)
+(Z . D . O1 . O2)
```

S pomocí symbolů hradel popsaných v první kapitole jste schopni si nakreslit také logický diagram. Ten je ale poměrně složitý a jeho pochopení je velmi náročné. Pokud se ho pokusíte nakreslit sami, měli byste zjistit, že se bude skládat z 12 NOT hradel, 24 dvouvstupových AND hradel (jak jistě víte, jedno čtyřvstupové AND hradlo můžete nahradit třemi dvouvstupovými AND hradly) a sedmi dvouvstupových OR hradel s maximálním hradlovým zpožděním rovným jedenácti (číselný údaj udávající dobu průchodu signálu přes TTL hradlo). Funkce poplašného zařízení se zdá na první pohled docela složitá.

Při pohledu do tabulky 2.1 a na soustavu rovnic se necháte jen velmi těžko přesvědčit, že tento obvod poplašného zařízení domu lze výrazně optimalizovat. Přesto si v této kapitole ukážeme, jak tyto čtyři vstupy a osm událostí poplašného zařízení zjednodušit tak, aby vyhovovaly většině základních čipů TTL.

Zjednodušování logické funkce pravdivostní tabulky

Dobrou zprávou pro začínající návrháře obvodů je, že optimalizace logického obvodu mohou dosáhnout pouhým pohledem do pravdivostní tabulky. Sice to nevypadá jako použitelný nástroj (zvláště v případě tabulky 2.1), ale tento postup může být stejně efektivní jako ostatní nástroje představené v této kapitole. S výhodou jej užijete také jako kontrolní nástroj pro ověření, že váš optimalizovaný obvod vykoná požadovanou funkci. Nevýhodou zjednodušování logické funkce pravdivostní tabulky jsou nároky na práci ve smyslu velkého množství rutiny, kterou je třeba podstoupit.

Tabulka 2.2: Pravdivostní tabulka domácího alarmu v Grayově kódu

Z	D	O1	O2	Odpověď alarmu
0	0	0	0	
0	0	0	1	
0	0	1	1	
0	0	1	0	Poplach
0	1	1	0	Poplach
0	1	1	1	
0	1	0	1	
0	1	0	0	
1	1	0	0	

Z	D	O1	O2	Odpověď alarmu
1	1	0	1	Poplach
1	1	1	1	Poplach
1	1	1	0	Poplach
1	0	1	0	Poplach
1	0	1	1	Poplach
1	0	0	1	Poplach
1	0	0	0	

Výchozí pravdivostní tabulka z úvodu této kapitoly se nemusí zdát příliš užitečná. Důvod budu zdůrazňovat v celé knize – vytváření seznamu podle logických odezev na změněnou hodnotu binárního vstupu není efektivní, protože najednou se může změnit velké množství stavů. Jestliže se podíváte do tabulky 2.1, uvidíte, že přechod ze stavu Z=0, D=O1=O2=1 do stavu Z=1, D=O1=O2=0 vyvolá změnu čtyř bitů. Přestože je to přirozená posloupnost binárních čísel a pravděpodobně i intuitivní způsob znázornění množství různých vstupních stavů, není to efektivní způsob, jak se dívat na odezvu logického obvodu na měnící se vstupy.

Mnohem lepší metodou je seřadit výstupní odezvy do pravdivostní tabulky podle Grayova kódu, který jsem znázornil v tabulce 2.2. Grayovy kódy představují číslovací systém, ve kterém se mění v čase vždy jen jeden bit: jejich popis a vytváření budou podrobně rozepsány v kapitole 4. Pokud seřazujete data bez ohledu na situaci, vždy implicitně používejte Grayův kód namísto inkrementace binárních vstupů, kterou vidíte v tabulce 2.1.

Vzali jsme radu v úvahu a s využitím Grayova kódu jsme přepracovali pravdivostní tabulku poplašného systému domu do tabulky 2.2. Při nahlédnutí do tabulky 2.2 uvidíte, že nespojitosti z tabulky 2.1 zmizely. Bitové masky znázorňující poplach se pěkně seskupily.

Prozkoumáním jednotlivých hodnot znázorňujících poplach zjistíte, že každý pár má tři bity stejné. Pro představu jsem v tabulce 2.3 zakroužkoval vždy různý bit v každém ze čtyř párů. V každém z těchto párů máme ke spuštění poplachu velmi konkrétní požadavek pro tři bity, ale čtvrtý bit se může nacházet v libovolném stavu.

Tabulka 2.3: Odlišné bity v párech poplach

Z	D	O1	O2	Odpověď alarmu
0	0	0	0	
0	0	0	1	
0	0	1	1	
0	0	1	0	Poplach
0	1	1	0	Poplach
0	1	1	1	
0	1	0	1	
0	1	0	0	
1	1	0	0	
1	1	0	1	Poplach
1	1	1	1	Poplach
1	1	1	0	Poplach
1	0	1	0	Poplach
1	0	1	1	Poplach
1	0	0	1	Poplach
1	0	0	0	

Tabulka 2.4: Pravdivostní tabulka s redundantními bity nahrazenými pomocí „x“

z	D	O1	O2	Odpověď alarmu
0	0	0	0	
0	0	0	1	
0	0	1	1	
0	x	1	0	Poplach
0	1	1	1	
0	1	0	1	
0	1	0	0	
1	1	0	0	
1	1	x	1	Poplach
1	x	1	0	Poplach
1	0	x	1	Poplach
1	0	0	0	

Řečeno jinak: čtvrtý bit je ke spuštění poplachu nepotřebný, a když potom sestavujeme logickou funkci typu součet součinů, můžeme ho ignorovat. K označení *redundantního* bitu jsme v tabulce 2.4 zkombinovali bitové páry a nahradili předchozí zakroužkované bity křížkem „x“. Tento křížek „x“ znázorňuje, že daný bit se může nacházet v libovolném stavu a výstup bude pořád pravdivý. Nahrazením dvou řádků v pravdivostní tabulce za jeden s jedním bitem ve stavu neurčitosti a znázorněným křížkem „x“ uvidíte, že se začalo dít něco až magického.

Tabulka 2.5: Přeskupená optimalizovaná pravdivostní tabulka s redundantními bity u sebe

Z	D	O1	O2	Odpověď alarmu
0	0	0	0	
0	0	0	1	
0	0	1	1	
0	x	1	0	Poplach
1	x	1	0	Poplach
0	1	1	1	
0	1	0	1	
0	1	0	0	
1	1	0	0	
1	1	x	1	Poplach
1	0	x	1	Poplach
1	0	0	0	

Tabulka 2.6: Znovu optimalizovaná pravdivostní tabulka s redundantními bity nahrazenými „x“

Z	D	O1	O2	Odpověď alarmu
0	0	0	0	
0	0	0	1	
0	0	1	1	
x	x	1	0	Poplach
0	1	1	1	

Z	D	O1	O2	Odpověď alarmu
0	1	0	1	
0	1	0	0	
1	1	0	0	
1	x	x	1	Poplach
1	0	0	0	

Nejdříve obvykle zaznamenáte, že tabulka je kratší, ale měli byste si uvědomit, že počet událostí znázorňujících poplach se snížil na polovinu a jsou méně komplikované než osm původních událostí. Rovnice se součtem součinů pro bity z tabulky 2.4 je:

```
Stav alarmu = (!Z . O1 . !O2)
+ (Z . D . O2)
+ (Z . D . !O2)
+ (Z . !D . O2)
```

Tento výraz typu součet součinů vyžaduje čtyři hradla NOT, osm hradel AND a tři hradla OR, přičemž maximální hradlové zpoždění bude devět. Došlo ke snížení celkového počtu hradel na méně než 50 % původního počtu a tato logická rovnice bude pracovat mnohem rychleji než původní.

To je opravdu skvělé vylepšení logického obvodu. Zeptejte se sami sebe, jestli to umíte ještě lépe. Abychom se podívali, jestli to půjde ještě lépe, přeskládáme data do tabulky 2.4 tak, aby události vyjadřující poplach a se stejným bitem ve stavu neurčitosti byly seskupeny dohromady a vznikla tabulka 2.5.

Když seskupíme události vyjadřující poplach a zároveň mající stejné redundantní bity, zjistíme, že v každém z těchto případů jsou dva zbývající bity ve stejném stavu a jeden bit v odlišném (což jsem zakroužkoval v tabulce 2.5).

V tabulce 2.5 si můžete všimnout, že změna jednoho bitu z původního Grayova kódu se ztratila; což však není problém. Uspořádání podle Grayova kódu posloužilo svému účelu – pomohlo znázornit podobné počáteční vstupní vzory jako sousedy. Ve složitějších pravdivostních tabulkách můžete přeskupit bitové vzory několikrát, dokud nenaleznete podobnosti. Pokud to děláte, nedbejte ztráty dat; důležité bitové vzory jsou stále uloženy v aktivních vzorech.

V tabulce 2.6 vidíte, co se stane, když znázorníme druhý redundantní bit. Pokud už dvě události znázorňující poplach nemají další společné redundantní bity, nemůžeme tento proces už dále opakovat. Dvě události z tabulky 2.6 mohou být popsány rovnicí typu součet součinů:

```
Stav alarmu = (O1 . !O2) + (Z . O2)
```

Takto optimalizovaná pravdivostní tabulka pro výstup stav alarmu má zredukovaný počet komponent na jedno hradlo NOT, dvě hradla AND a jedno hradlo OR, přičemž se vykoná za pět hradlových zpoždění – slušné zlepšení oproti původním 43 hradlům a 11 hradlovým zpožděním!

Jste-li cynik, můžete mít dojem, že jsem kvůli demonstraci dramatického zlepšení tento příklad uvařil z vody. Ve skutečnosti je zde zmíněná aplikace vůbec mým prvním pokusem s logickým obvodem pro ukázku optimalizačních operací logických obvodů; k podobným výsledkům můžete dojít sami, když začnete se základním logickým obvodem a budete ho zkoušet zjednodušit.

Karnaughovy mapy

Použití pravdivostních tabulek je efektivní, ale není to nejúčinnější metoda pro optimalizaci logických digitálních obvodů. Jeden velmi chytrý francouzský matematik, Maurice Karnaugh (vyslo-

vováno Karno) přišel se způsobem, jak zjednodušit proces optimalizace pravdivostních tabulek, tím, že rozdělil vstupy pravdivostních tabulek od středu dolů a tyto dvě poloviny uspořádal svisle vedle sebe, aby přehledněji znázorňovaly vztahy mezi bity. Takto přetvořené pravdivostní tabulky se nazývají *Karnaughovy mapy* a nejlépe se hodí pro funkce s jedním výstupním bitem a třemi až šesti vstupními bity.

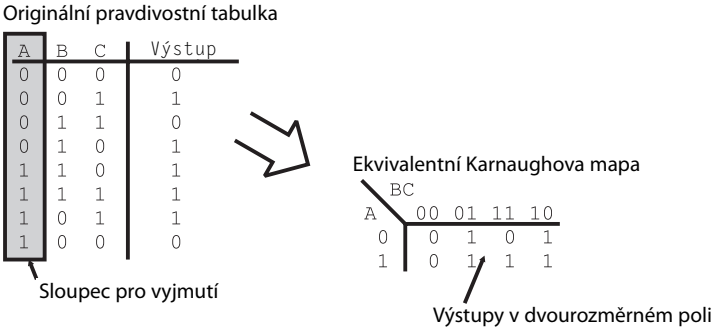
Můj popis Karnaughových map může znít povrchně, ale ve skutečnosti je velice přesný. Běžnou pravdivostní tabulku můžeme považovat za jednorozměrné zobrazení logické funkce, a když ji správně upravíme, můžeme zachovat vztah mezi aktivními výstupy, jak jsme si ukázali v předchozí části. Problém této metody je v tom, že je poměrně namáhavá a potřebujete velké množství papíru. Karnaughovy mapy představují data v dvojrozměrném poli, což dovoluje rychlejší prohledávání aktivních výstupních bitů v závislosti na vstupech a nalezení základních vztahů mezi nimi.

Příklad převedení třívstupové logické funkce z pravdivostní tabulky do Karnaughovy mapy je znázorněn na obrázku 2.2. Původní logická funkce byla:

```
Výstup = (!A + !B + C) + (!A . B . C) + (A .B . !C) + (A . B . C) + (A . !B . C)
```

K vytvoření Karnaughovy mapy vytvoříme matici dva krát čtyři, v níž jsou řádkům přiřazeny rozdílné hodnoty pro A a sloupce obsahují čtyři rozdílné hodnoty pro B a C. Všimněte si, že řádky jsou vypsány v dvoubitovém Grayově kódu – to je velmi důležitá vlastnost Karnaughových map, a jak musíme zdůraznit, i důležitý nástroj umožňující optimalizovat funkci.

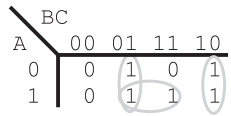
Jakmile si vyberete dvě osy Karnaughovy mapy, pečlivě převeďte výstupy z pravdivostní tabulky do Karnaughovy mapy. Při převádění výstupů pracujte s mapou jako s dvojrozměrným polem, v němž rozměr X představuje vstupy, které nebyly vyjmuty, a rozměr Y naopak vstupy, které byly vyjmuty z pravdivostní tabulky. Pokud tuto činnost provádíte poprvé, budete při ní snadno chybovat, poněvadž vyžaduje určitou praxi. Abyste se ujistili, že jste tomuto postupu porozuměli, je vhodné provést zpětnou kontrolu tím, že převedete svou Karnaughovu mapu do pravdivostní tabulky a porovnáte ji s původní tabulkou.



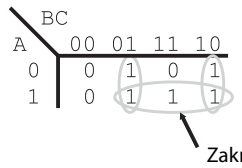
Obrázek 2.2: Převádění pravdivostní tabulky na Karnaughovu mapu

Pokud jste vytvořili Karnaughovu mapu pro funkci, je docela dobrý nápad si ji okopírovat nebo vypsat perem před tím, než budete pokračovat. Doporučili jsme to proto, že u pravdivostních tabulek budeme dávat do kroužků výstupy, které mají stejné, nezměněné skupiny bitů. Až budeme kroužkovat vstupy, budeme měnit skupiny bitů, které se vám budou zdát příliš neefektivní k zakroužkování, nebo zjistíte, že jste udělali chybu v zakroužkování bitů. Fotokopie nebo tabulka psaná perem vám to dovoluje zkoušet znovu a znovu bez nutnosti přepisovat Karnaughovu mapu.

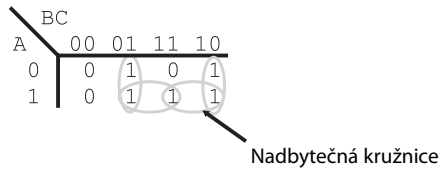
Pro příklad zobrazený na obrázku 2.2 má na sobě Karnauhgova mapa umístěné tři kroužky, jak je znázorněno na obrázku 2.3. Každý kruh by měl vést k sjednocení dvou skupin vstupů a přetvoření alespoň jednoho bitu na redundantní. Správné zakroužkování bitů může být obtížné k pochopení, ale existuje několik pravidel, která vám pomohou. Každý kruh musí být násobkem dvou bitů – nemůžete označit tři bity (jako v příkladu na obrázku 2.4). Za druhé není problém, když se kruhy v jednotlivých bitech překrývají. Měli bychom poukázat na případ pro nadbytečný kruh (obr. 2.5). Pokud je nakreslená kružnice a všechny zakroužkované bity jsou uzavřeny v jiných kruzích, pak je uzavřená kružnice nadbytečná. Za třetí, pamatujte, že když kroužkujete bity, můžete je kroužkovat v násobcích dvou, a nejenom po dvou. Na obrázku 2.6 jsme změnili tříbitovou Karnaughovu mapu s výstupy v „1" pro A=0 a B=C=1 a A=1 a B=C=0 a objevili jsme, že můžeme zakroužkovat dvě skupiny čtyř bitů. V každém z těchto případů vytvoříme dva redundantní bity.



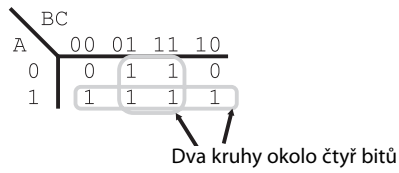
Obrázek 2.3: Zakroužkování bitů v příkladu Karnaughovy mapy



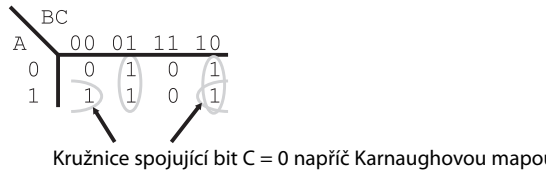
Obrázek 2.4: Nesprávné zakroužkování lichého počtu bitů v příkladu Karnaughovy mapy



Obrázek 2.5: Nadbytečné kroužky v příkladu Karnaughovy mapy



Obrázek 2.6: Karnaughova mapa zobrazující zakroužkované více než dva bity současně



Obrázek 2.7: Kružnice rozprostírající se mimo zdánlivé hranice Karnaughovy mapy

Nakonec je potřeba říci, že považovat Kanaughovu mapu za dvojrozměrné pole je nepřesné – je to vlastně uzavřená smyčka s propojenými vrcholy a stranami. Když vypisujete Karnaughovu mapu, možná zjistíte, že bity, které lze zakroužkovat (myšleno se společným vzorem), leží na protějších koncích mapy. To není problém, pokud se jedná o shodné bity.

Jakmile máte jednou zakroužkované výstupy, můžete začít vypisovat optimalizovanou funkci. Pro ukázku se můžete podívat na Karnaughovy mapy z obrázků 2.3, 2.6 a 2.7. Výstupní rovnice pro tyto obrazce jsou:

$Výstup_{2.03}$ = (!B . C) + (B . !C) + (A . C)

$Výstup_{2.06}$ = A .C

$Výstup_{2.07}$ = (!B . C) + (B . !C) + (A . !C)

Z	D	O1	O2	Odpověď alarmu
0	0	0	0	
0	0	0	1	
0	0	1	1	
0	0	1	0	Poplach
0	1	1	0	Poplach
0	1	1	1	
0	1	0	1	
0	1	0	0	
1	1	0	0	
1	1	0	1	Poplach
1	1	1	1	Poplach
1	1	1	0	Poplach
1	0	1	0	Poplach
1	0	1	1	Poplach
1	0	0	1	Poplach
1	0	0	0	

Karnaughova mapa alarmu

ZD \ O1 O2	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	0	1	1	1
10	0	1	1	1

Obrázek 2.8: Pravdivostní tabulka domácího alarmu v Karnaughově mapě

V této kapitole jsem chtěl ukázat, jaké lze použít různé nástroje pro optimalizaci systému domácího alarmu prezentovaného v úvodní kapitole. Funkci systému alarmu lze optimalizovat použitím Karnaughovy mapy, jak je znázorněno na obrázku 2.8. Na obrázku 2.8 jsme nakreslili kruhy okolo skupin čtyř aktivních výstupních bitů, které jsou společné, a výsledkem je logická funkce:

```
Odpověď alarmu = (Z .O2) + (O1 . !O2)
```

která je shodná s rovnicí vytvořenou zjednodušením pomocí pravdivostní tabulky a mnohem méně pracná.

Než budeme pokračovat, chtěli bychom říct, že pokud se jednou seznámíte s Karnaughovými mapami, objevíte v nich rychlou a účinnou metodu pro optimalizaci jednoduchých logických funkcí. Dobře si osvojit a být schopný bezchybně převádět informaci z pravdivostní tabulky do Karnaughovy mapy zabere čas. Je nutné naučit se správně kroužkovat aktivní výstupy k vytvoření optimalizovaného obvodu typu součet součinů. Pokud ale tuto dovednost jednou ovládnete, zjistíte, že budete moci převádět požadavky přímo do Karnaughových map bez toho, abyste museli jako úvodní krok vypisovat pravdivostní tabulku.

Zákony Booleovy aritmetiky

Jedním ze způsobů optimalizace obvodů je prohlédnout jejich výstupní rovnice a zkusit najít vztahy, díky nimž můžete s výhodou použít pravidla a zákony z tabulky 2.7. Tato pravidla byste si měli

uložit do paměti tak rychle, jak je to jen možné (nebo alespoň opsat na list papíru a používat jako pomůcku). Pomohou vám při optimalizaci logických rovnic bez potřeby pravdivostních tabulek nebo Karnaugových map. Mnohá z těchto pravidel se mohou zdát zcela zřejmá, ale je neuvěřitelné, na co všechno zapomenete, nebo se vám nebude zdát jasné, pokud pracujete na optimalizaci logické rovnice v nějaké úloze.

Tabulka 2.7: Zákony a pravidla Booleovy aritmetiky

Pravidlo/Zákon	Příklad Booleovy aritmetiky
Zákon neutrálnosti hodnot AND	A . 1 = A
Zákon neutrálnosti hodnot OR	A + 0 = A
Zákon agresivity nuly (Vynulování výstupu)	A . 0 = 0
Zákon agresivity jedničky (Nastavení výstupu)	A + 1 = 1
Zákon totožnosti	A = A
Zákon vyloučení třetího	A . !A = 0
Zákon vyloučení třetího	A + !A=1
Zákon idempotence AND	A . A = A
Zákon idempotence OR	A + A = A
Komutativní zákon AND	A . B = B . A
Komutativní zákon OR	A + B = B + A
Asociativní zákon AND	(A . B) . C = A . (B . C) = A . B . C
Asociativní zákon OR	(A + B) + C = A + (B + C) = A + B + C
Distributivní zákon AND	A . (A + B) = (A . B) + (A . C)
Distributivní zákon OR	A + (B . C) = (A + B) . (A + C)
De Morganovo pravidlo pro NOR	!(A + B) = !A . B
De Morganovo pravidlo pro NAND	!(A . B) = !A + !B

Když mluvíme o používání zákonů a pravidel z tabulky 2.7 k zjednodušování logické rovnice, obyčejně používáme termín *snižování* namísto *optimalizace*. Důvodem, proč uvažujeme o těchto operacích jako o snižování, je, že jak logickou operaci postupně procházíme, tak ji zmenšujeme, snažíce se najít nejefektivnější výraz typu součet součinů.

Dvě shodné funkce jsou použity k zobrazení stejných stavů, kterými může procházet nezměněná vstupní hodnota hradlem AND nebo OR. Zákony idempotence lze shrnout prohlášením, že pokud vstup prochází přes neinvertující hradlo, jeho hodnota se nemění.

Zbývající zákony – komutativní, asociativní a distributivní – a De Morganova pravidla nejsou tak jednoduché a extrémně silné nástroje pro optimalizaci logické rovnice. Komutativní zákony vyjadřují, že vstupy do AND a OR hradel lze obrátit, což se může zdát zřejmé, ale pokud máte dlouhou logickou rovnici napsanou v libovolném tvaru (ne bezpodmínečně ve formátu součet součinů), můžete být lehce zmateni. Je užitečné mít v záloze tento zákon k přeměně rovnice v něco, s čím se dá mnohem lépe manipulovat.

Při předvedení těchto zákonů se můžeme vrátit k nějakému logickému obvodu popsanému v příkladech Karnaughových map z předchozí sekce. Podíváme-li se na obrázek 2.3, pak úvodní logická rovnice typu součet součinů bude:

```
Výstup = (!A . !B . C) + (!A . B . !C) + (A . !B . C) + (A . B . C) + (A . B . !C)
```

Užitím asociativního zákona AND můžeme tuto rovnici přepsat s výrazem A odděleným od výrazů B a C. Tak uvidíme, jestli existují nějaké případy, kde jsou výrazy B a C identické.

```
Výstup = !A . (!B . C) + !A . (B . !C) + A . (!B . C) + A . (B . C) + A . (B . !C)
```

Pokud to uděláte, uvidíte, že vnitřní výrazy prvního a třetího součinu jsou shodné. Zároveň s tím si všimněte, že druhý a pátý součin je také shodný. Použitím distribučního zákona OR můžeme první a třetí výraz sloučit jako:

```
(!A . !B . C) + A . (!B . C) = 1 . (!B .C)
```

Jednička s funkcí AND s !B AND C může být dále zjednodušena použitím zákona totožnosti AND (1 AND A se rovná A):

```
!A . (!B . C) + A . (!B . C) = (!B . C)
```

Toto můžeme zopakovat pro druhý a pátý výraz.

```
(!A . B . !C) + (A . B . !C) = (B .C)
```

Pokud se vrátíte zpět k originální funkci, uvidíte, že čtvrtý výraz (A . B . C) nemůžeme zjednodušit spojením s dalším výrazem. Ve skutečnosti může být spárován se třetím výrazem (A . !B . C) přeskupením dvou výrazů (použitím komutativního zákona AND) tak, že část výrazů pracující se dvěma bity je společná (A . C). Pokud je toto splněno, třetí a čtvrtý výraz můžeme zredukovat na:

```
(A . !B .C) + (A . B . !C) = (A . !C)
```

Po dokončení této práce máme optimalizovanou nebo zjednodušenou logickou rovnici pro tuto funkci, která je shodná s tím, k čemu jsme dospěli použitím Karnaughových map.

```
Výstup = (!B . C) + (B . !C) + (A . C)
```

Pohledem na zjednodušenou logickou rovnici byste měli zjistit, že jsou tu dva výrazy, které budou mít výstup „1“ ve stejnou dobu (!B . C) a (A . C), kde A = 1, B = 0 a C=1). To není problém, protože hradlo OR (ačkoli symbol, který používáme je „+“) bude mít výstup 1, navzdory tomu, že má hodně pravdivých vstupů. Toto jsme zmínili při výkladu Karnaughových map, ale chci zdůraznit, že stejný problém nastává, pokud zjednodušujeme logické rovnice.

Než budeme pokračovat, dovolte mi vrátit se k rovnici domácího alarmu. Podívejme se, jestli ji můžeme vyřešit stejným způsobem jako předchozí rovnici. Začneme logickou rovnicí typu součet součinů:

```
Stav alarmu = (!Z . !D . O1 . !O2)
+(!Z . D . O1 . !O2)
+(Z . !D . !O1 . O2)
+(Z . !D . O1 . !O2)
+(Z . !D . O1 . O2)
+(Z . D . !O1 . O2)
+(Z . D . O1 . !O2)
+(Z . D . O1 . O2)
```

Můžeme vynést hodnoty „Z“ ze součinů a hledat podobnosti ve zbývajících závorkovaných výrazech a jejich slučováním pomocí asociativních, distributivních, komplementárních zákonů a zákonů totožnosti AND. Můžeme vidět, že první a čtvrtý výraz a druhý a sedmý výraz lze sloučit v logickou rovnici:

```
Stav alarmu = (!D . O1 . !O2)
+ (D . O1 . !O2)
+(Z . !D . !O1 . O2)
+(Z . !D . O1 . O2)
+(Z . D . !O1 . O2)
+(Z . D . O1 . O2)
```

Přenesení „O1“ do popředí dovolí sloučení třetího a čtvrtého a pátého a šestého výrazu předchozí logické rovnice, výsledkem je nová rovnice:

```
Stav alarmu = (!D . O1 . !O2)
+ (D . O1 . !O2)
+ (Z . !D . O2)
+(Z . D . O2)
```

Zrušili jsme polovinu výrazů a ty, které zůstaly, jsou o 25 % menší. Když se podíváte na novou logickou rovnici, vidíte, že sloučením prvního a druhého výrazu (tvořících „D“ redundantní bit v procesu) a sloučením třetího a čtvrtého výrazu („D“ je znovu redundantní bit) skončíme s rovnicí:

```
Stav alarmu = (O1 . !O2) + (Z . O2)
```

která je znovu rovnicí, k níž jsme došli zjednodušením funkce užitím pravdivostních tabulek nebo Karnaughových map.

Tabulka 2.8: Testování optimalizované logické funkce domácího alarmu

Z	D	O1	O2	Poplach	O1 ● !O2	Z ● O2	OR
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1
0	1	0	0	1	1	0	1
0	1	0	1	0	0	0	0
0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	1	0	1	1
1	0	1	0	1	0	1	1
1	0	1	1	1	1	0	1
1	1	0	0	1	1	0	1
1	1	0	1	1	0	1	0
1	1	1	0	1	0	1	0
1	1	1	1	0	0	0	0

Osobně se přikláníme k optimalizaci logických obvodů použitím zákonů a pravidel Booleovy aritmetiky vypsaných v tabulce 2.7. Pokaždé, když vytvoříme zjednodušenou logickou rovnici typu součet součinů, vrátíme se zpět a porovnáme její výstupy v pravdivostní tabulce s požadovanými výstupy. Když to děláme, zobrazujeme hodnoty pro každý součin (AND) a pro každý konečný součet (OR) v oddělených řádcích, jak je znázorněno v tabulce 2.8.

Optimalizace pro technologii

Když se podíváme na zákony v tabulce 2.7 a srovnáme je s textem v předchozí části, zjistíme, že jsme poslední dva zákony (De Morganovy poučky) pominuli. Tyto dva zákony se pro zjednodušení základních logických obvodů běžně nepoužívají, ale využití najdou při převodu částí rovnice do hradel NAND nebo OR, což je důležité pro závěrečnou implementaci logické funkce do skutečné elektroniky. Dalším důležitým hlediskem optimalizace pro technologii je připojení funkce ze zbylých hradel do obvodu; při prozkoumání implementace konkrétního logického obvodu mu často můžete přidat funkce, aniž byste zvýšili jeho cenu.

Tabulka 2.9: Pravdivostní tabulka hradla XOR

A	B	A ^ B
0	0	0
0	1	1
1	1	0
1	0	1

Doposud jsme v knize neprobírali hradlo Exkluzivní OR (XOR) do větších detailů, ale pro implementaci binárních adres je podstatné, jak si ukážeme v knize později. V první kapitole jsme si ukázali hradlo OR s pravdivostní tabulkou znázorněnou v tabulce 2.9.

Měli byste být schopni vytvořit logickou rovnici pro tabulku XOR ve tvaru:

```
Výstup = (!A.B)+(A.!B)
```

která se nemusí jevit jako jednoduchý případ pro optimalizaci. Stejně tak pro vás bude pravděpodobně těžké uvěřit, že následující logická operace představuje stejnou funkci:

```
Výstup = !((A.B)+!(A+B))
```

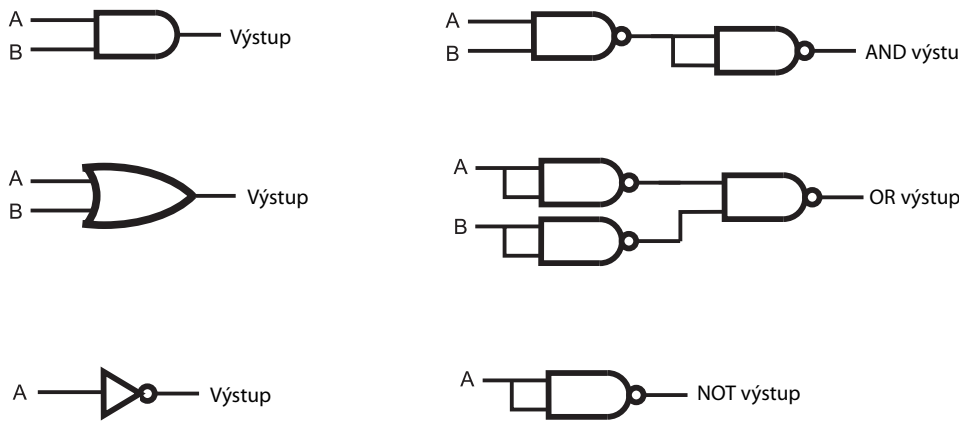
ale použitím De Morganových pravidel a také dalších pravidel a zákonů z tabulky 2.7 můžeme manipulacemi zobrazenými v tabulce 2.10 dojít ke zjištění, že obě rovnice jsou totožné a také přepočítaná hradla vyžadovaná mezikroky vám dají seznam rozdílných použití hradla XOR. Pro každý přechodný krok v tabulce 2.10 existuje implementace hradla XOR, kterou můžete realizovat s použitím počtu hradel vypsaných napravo od výrazů.

Základním hradlem používaným v TTL je hradlo NAND. To znamená, že tři základní hradla (AND, OR a NOT) jsou sestavena z jeho násobků, jak jsme znázornili na obrázku 2.9. Základním logickým obvodem pro CMOS je hradlo NOR a obrázek 2.10 ukazuje jeho použití pro realizaci tří základních hradel. Třetí hradlo NAND a záměny NOR pro hradla OR a AND jsou výbornými příklady využití De Morganových zákonů v praxi. Tyto úpravy můžete porovnat s De Morganovými výroky, pravidly a zákony v tabulce 2.7.

Tabulka 2.10: Rozdílné implementace hradla XOR

Výrazy	Počet NOT	Počet AND	Počet OR	Počet NAND	Počet NOR
(!A . B)+(A . !B)	2	2	1	0	0
!(!(!A . B) . !(A . !B))	2	0	0	3	0
!((A+!B) . (!A + B)	2	0	2	1	0

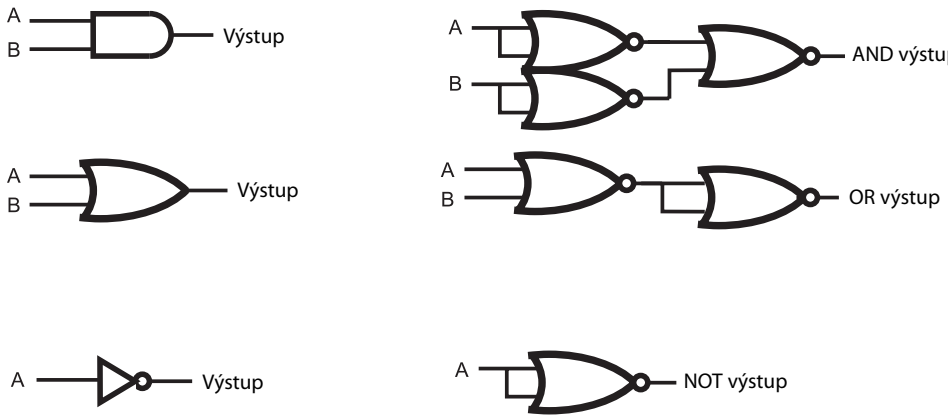
Výrazy	Počet NOT	Počet AND	Počet OR	Počet NAND	Počet NOR
!((A . B) + (!A . !B))	2	2	0	0	1
!((A . B) + (A + B))	0	1	0	0	2



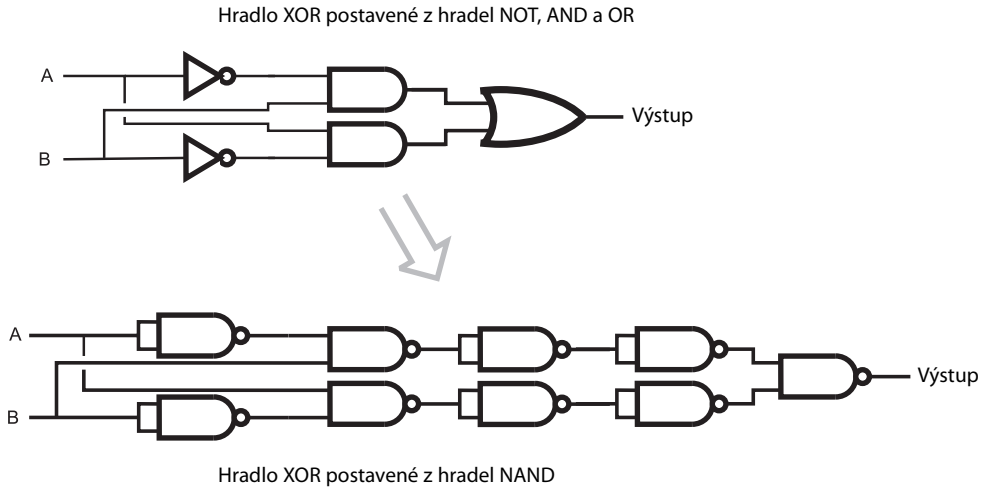
Obrázek 2.9: Implementace tří základních hradel použitím hradel NAND

Pochopením toho, jak jsou hradla realizována v čipu, získáme nový pohled na optimalizaci hradla k provádění co nejrychlejších operací s logickými funkcemi. Použitý příklad hradla XOR můžeme graficky znázornit jako hradlo, které je vytvořeno užitím hradel AND, OR a NOT a tím, jak jsou tato hradla realizována hradly NAND v čipech TTL (obrázek 2.11).

Pohlédnete-li na spodní část obrázku 2.11, uvidíte, že jsou tu dvě skupiny hradel NAND zapojených jako invertory. Pohledem zpět do tabulky 2.7 můžeme vidět, že dvakrát invertovaný signál je stejný signál, a proto tyto dvě skupiny hradel můžeme odstranit, jak je znázorněno na obrázku 2.12. Výsledný obvod XOR bude zpracovávat signál pomocí tří hradel NAND, která se počítají jako tři hradlová zpoždění. Zde vidíte výsledek toho, co nazýváme technologií optimalizace: logický obvod byl zjednodušen na základní minimum použitím výhody operace základních logických hradel, která tvoří implementovanou technologii.



Obrázek 2.10: Realizace tří základních hradel použitím hradel NOR

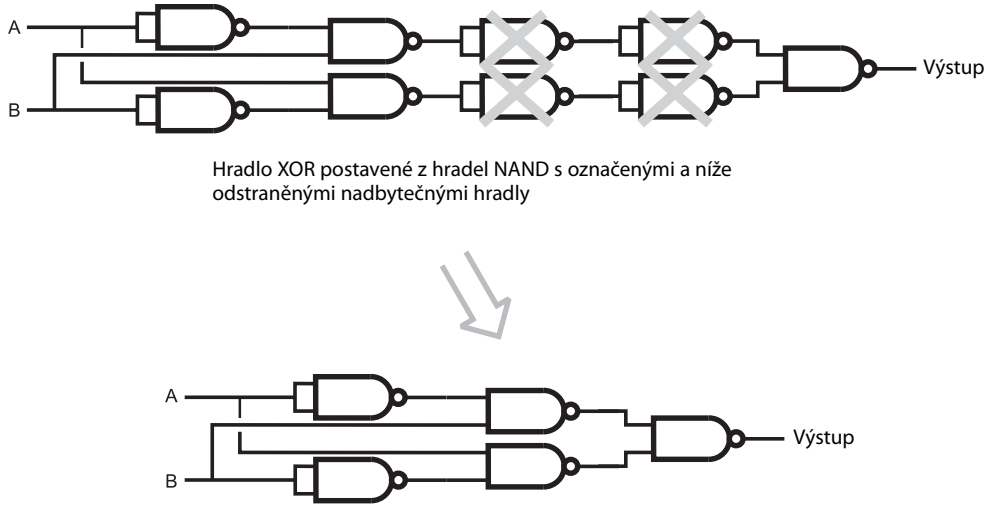


Obrázek 2.11: Hradlo XOR postavené jako rovnice typu součet součinů a převedené do hradel NAND

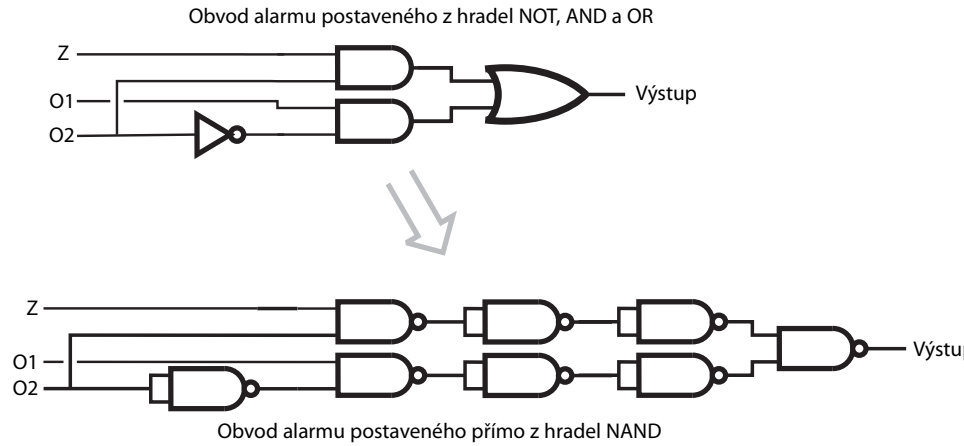
Než budeme pokračovat, měli bychom se blíže podívat na obvod domácího alarmu, který jsme probírali v průběhu této kapitoly. Na začátku kapitoly jsme použili docela odvážné prohlášení, že můžeme obvod zjednodušit do nejzákladnějšího dostupného čipu TTL – vidíte, že jsme se nemýlili. Optimalizovaná logická rovnice pro systém domácího alarmu byla ve tvaru:

```
Stav alarmu = (O1.!O2)+(Z.O2)
```

Můžeme ji nejprve implementovat dvěma hradly AND, jedním hradlem OR a jedním hradlem NOT zobrazenými na obrázku 2.13 a převést ji do hradel NAND. Všimněte si nápadné podobnosti mezi diagramem domácího alarmu na obrázku 2.13 a diagramem logického XOR. Jak je vidět z obrázku 2.14, logická funkce je zjednodušena na čtyři hradla NAND (o jedno méně, než je potřeba na hradlo XOR realizované pomocí hradel NAND).

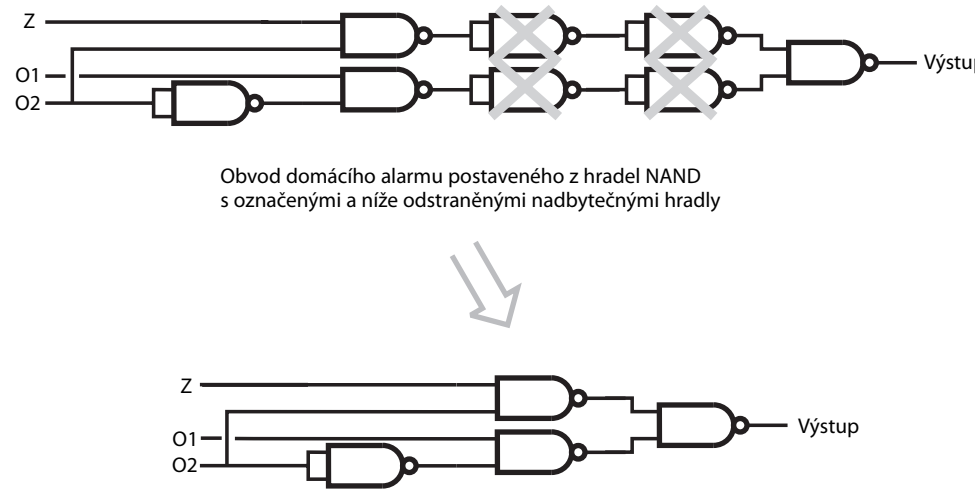


Obrázek 2.12: Optimalizované hradlo XOR sestavené z hradel NAND



Obrázek 2.13: Logický obvod domácího alarmu postavený s použitím hradel AND, OR a NOT a převedený do hradel NAND

Výsledná funkce domácího alarmu vyžaduje čtyři dvouvstupová hradla NAND, která jsou obsažena v nejzákladnějším čipu TTL 7400. Všechny TTL čipy s výjimkou tohoto a odvozených verzí obsahují více než čtyři hradla, protože zprostředkovávají dodatečné funkce vyžadující mnohonásobná hradla NAND. Nepřeháněli jsme, když jsme řekli, že logickou funkci domácího alarmu lze zjednodušit do nejjednoduššího dostupného čipu TTL. V další kapitole vám představíme způsob, jakým čip TTL zprostředkovává základní logické funkce.



Obrázek 2.14: Optimalizovaný obvod alarmu postavený z hradel NAND

Test

1. Pro měření úrovně optimalizace digitálních elektronických obvodů se používají tři parametry:
   (a) Cena, rychlost a složitost
   (b) Zpoždění hradla, počet hradel a technologie optimalizace
   (c) Počet hradel, počet hradlových zpoždění, kterými musí signál projít, a technologie optimalizace
   (d) Počet hradel, počet spojů, kterými musí signál projít, a technologie optimalizace
2. Pokud je hradlové zpoždění TTL logiky 8 ns, signál prochází skrz 9 hradel a nejkratší cesta je pět hradlových zpoždění, čas potřebný k projití signálu skrz hradla je:
   (a) 40 ns
   (b) 8 ns
   (c) nekonečný
   (d) 24 ns
3. Pokud vypisujete pravdivostní tabulku, vstupy by měly být seřazeny:
   (a) pomocí Grayova kódu
   (b) pomocí binárního růstu
   (c) v abecedním pořadí
   (d) podle důležitosti
4. Redundantní bit se v pravdivostní tabulce:
   (a) označuje jako „dc“ a nahrazuje společné bity ve dvou pravdivých skupinách vstupů
   (b) označuje jako „x“ a nahrazuje společné bity ve dvou pravdivých skupinách vstupů
   (c) označuje jako „dc“ a nahrazuje rozdílné bity ve dvou pravdivých skupinách vstupů
   (d) označuje jako „x“ a nahrazuje rozdílné bity ve dvou pravdivých skupinách vstupů
5. Pokud optimalizujete logickou funkci, můžete očekávat, že:
   (a) počet čipů potřebných na počátku návrhu se zmenší
   (b) optimalizovaná funkce poběží rychleji než původní návrh
   (c) můžeme použít levnější čipy než v původním návrhu
   (d) Odpovědi (a) až (c) jsou všechny možné a nemusí být možné optimalizovat původní rovnici typu součet součinů.
6. Karnaughovy mapy jsou:
   (a) nástroje navržené pro nalezení cesty digitálním elektronickým obvodem
   (b) nástroje, které pomohou optimalizovat logickou funkci
   (c) nejúčinnější metody pro optimalizaci logických funkcí
   (d) těžké na zapamatování a musí být použity v každém návrhu logické funkce
7. Logická funkce typu součet součinů
   Výstup = (A . !B . C) + (!A . !B . C)
   může být zjednodušena na:

(a) A . C
(b) !A . !B
(c) C . !B
(d) C

8. Které z následujících párů zákonů Booleovy aritmetiky nemohou být použity společně:
   (a) totožnosti a De Morganovo pravidlo
   (b) asociativní a idempotentní
   (c) komplementární a komutativní
   (d) všechny zákony a pravidla se mohou používat společně
9. Náhrada hradla AND pomocí NAND:
   (a) je sestavena ze dvou hradel NAND a vyžaduje dvě hradlové prodlevy pro průchod signálu
   (b) je sestavena ze tří hradel NAND a vyžaduje dvě hradlové prodlevy pro průchod signálu
   (c) je sestavena ze tří hradel NAND a vyžaduje tři hradlové prodlevy pro průchod signálu
   (d) je sestavena z jednoho hradla NAND a vyžaduje dvě hradlové prodlevy pro průchod signálu
10. Technologie optimalizace je definována jako:
   (a) navrhování obvodu s co nejmenším počtem čipů a co nejrychlejším průchodem signálu
   (b) implementace logických funkcí za účelem získání výhody základní logiky využívající také zbylá hradla
   (c) hledání nejlepší technologie digitální elektroniky pro použití v aplikacích
   (d) navrhování soustavy obvodů, která vyzáří nejmenší množství tepla při provádění požadované operace