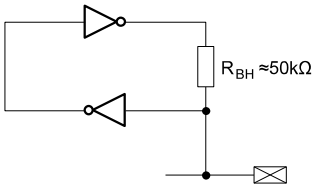
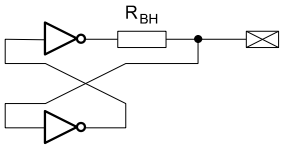
Bloki WE/WY

Na poprzednim wykładzie układ Bus Hold („Pin Keeper") był tylko wspomniany. Dzisiaj nadszedł moment, by przyjrzeć mu się bliżej. Przełącznik na schemacie tego układu wskazuje na dwa stany jego pracy. Jeżeli układ XC 9500XL jest skasowany, odbywa się proces programowania, odbywa się komunikacja przez interfejs JTAG lub jest inicjalizowane napięcie zasilania, to przełącznik jest w pozycji „0". Wszystkie wyjścia są „wyciszone" rezystorami Pull-Up (jest uzyskiwana „słaba jedynka" logiczna na pinach). Podczas procesu włączania układu „słaba jedynka" jest utrzymywana na wyprowadzeniach do chwili, aż wartość napięcia zasilającego V_{CCINT} przekroczy poziom 2,5V.

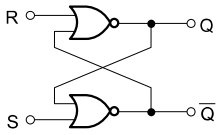
Jeżeli układ scalony pracuje normalnie, to przełącznik znajduje się w dolnej pozycji (zanegowany „PIN"). Wygląda to następująco:



Układ taki powinniśmy kojarzyć z asynchronicznym przerzutnikiem. Jest tu pętla sprzężenia zwrotnego identyczna z pętlami spotykanymi w przerzutnikach:



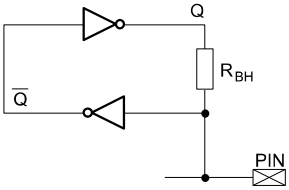
W tej postaci układ jest podobny do asynchronicznego przerzutnika RS. Normalny RS jest wykonany na dwuwejściowych bramkach:

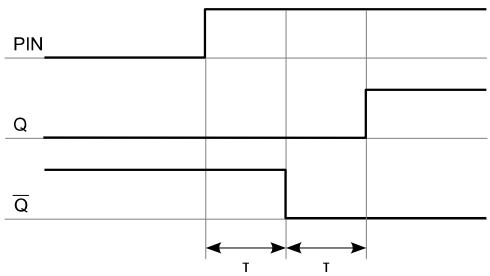


W układzie Bus Hold nie mamy wejść sterujących R i S. Jest tylko pętla podtrzymująca stan przerzutnika. Może on przebywać w jednym z dwóch stanów bistabilnych. Przerzutnik wygląda „partyzancko", bo nie ma wejść sterujących. Jak się to ustrojstwo przełącza? Dzięki umieszczeniu w nim rezystora. By prześledzić jego działanie, musimy na ten „przerzutnik" spojrzeć z „analogowej strony".

Umieszczenie rezystora zapewnia przełączanie się przerzutnika od sygnału wyjściowego. Stan logiczny na pinie przełącza przerzutnik. Podtrzymuje on stan logiczny na linii zanim ta przejdzie w stan wysokiej impedancji. Stanie się to po zdjęciu sterownika, który steruje pinem. Układ wstawiono po to, by linia pracująca w systemie magistralowym w momencie przejścia nadajników w stany wysokiej impedancji zachowała swój stan i nie „wisiała w powietrzu".

Przełączanie układu Bus Hold będziemy rozpatrywać na przykładzie podania zbocza narastającego na pin:

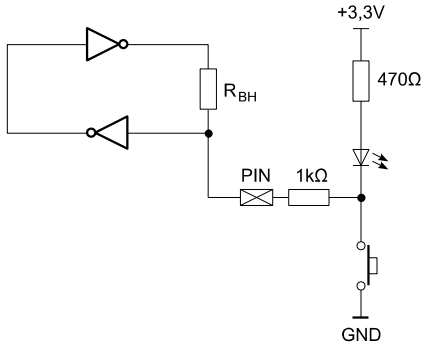




Zakładamy że przed przełączeniem stan Q był równy 0. $\overline{Q}$ wynosi zatem 1. Przyjmujemy, że każda z bramek wnosi opóźnienie τ. Gdy na *PIN* pojawi się „1", to po opóźnieniu τ zmieni się stan $\overline{Q}$ (z wysokiego na niski). Gdy $\overline{Q}$ zmienił swój stan, to stan Q zmieni się z 0 na 1 po czasie τ. Gdy Q zmienił swój stan, przerzutnik znaj-

dzie się w stanie stabilnym. Zajęło to odcinek czasu równy 2τ. Bez rezystora R_{BH} mielibyśmy konflikt w postaci zwarcia dwóch sterowników (nadajników). Rezystor R_{BH} zapewnia nam to, że może występować jakaś różnica napięć między węzłami, ale prąd przepływający między nimi będzie ograniczony. Gdy zasilamy układ napięciem 3,3V to przez rezystor może przepłynąć prąd o wartości 66μA. Jego wartość nie zaszkodzi układowi.

UWAGA: układy Bus Hold należy wyłączać!



Rezystor 1kΩ zabezpiecza przed bezpośrednim zwarciem wyprowadzenia do masy. Zwarcie wyprowadzenia z masą poskutkowałoby uszkodzeniem makrokomórki.

Kłopociki pojawiają się gdy wyprowadzenie będziemy wykorzystywali jako wejście. Naciśnięcie przycisku zestawu laboratoryjnego ma wymuszać „zero”. Naciśnięcie przycisku powoduje pojawienie się na pinie napięcia o wartości $\frac{1}{51}\cdot 3{,}3\,V$. Jeżeli będziemy chcieli podać „jedynkę” na pin, to przełączenie nie wystąpi. Dzieje się tak przez obecność diody LED. Spadek napięcia na tej diodzie jest na poziomie około 2V. Gdy puścimy przycisk, to przełączenie do „jedynki” nie nastąpi z powodu spadku napięcia na diodzie i dzielniku napięcia. Gdyby diody LED nie było, to zwolnienie przycisku łączyłoby pin z „jedynką” logiczną poprzez rezystancję 1,5kΩ (dokładniej $\frac{50}{51{,}5}\cdot 3{,}3\,V$). Daje to „dobrą” jedynkę. Dioda LED i spadek napięcia na niej sprawia, że potencjał na pinie ustali się na poziomie zakresu zabronionego i o przełączeniu się układu nie ma mowy. Zwarcie diody LED sprawiłoby, że układ Bus Hold od razu przełączyłby się do jedynki.

Wyłączanie układu Bus Hold jest ważne, ponieważ załączony układ zwyczajnie będzie przeszkadzał w pracy. Podawanie zera na pin będzie zachodziło bezproblemowo, natomiast będą kłopoty z podawaniem jedynki na wyprowadzenie układu. Wyłączania Bus Holda dokonuje się poprzez wejście w opcje implementacji układu (w środowisku ISE) i ustawieniu „Float” dla właściwości „I/O Pins”.

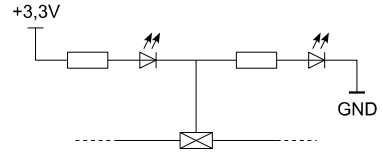
Schemat logiczny zestawu laboratoryjnego

Umieszczono w nim jeden układ logiczny (9536XL) i 3 generatory sygnałów zegarowych (Rodzina 9500 posiada 3 wejścia zegarowe globalne i każdemu wejściu odpowiada jeden generator). Częstotliwość przebiegu uzyskiwanego z generatorów można zmieniać poprzez przełączanie kondensatorów przyłączonych do wyprowadzenia TRIGGER a masą układu 555. Przełączania dokonuje się przy użyciu dip switcha. Wyjście generatora steruje diodą LED, co ułatwia obserwację sygnału przy jego małej częstotliwości. Sygnał z generatorów nie jest podpięty na stałe do wyprowadzeń układu XC 9536XL. Dołączania dokonuje się przy pomocy zworek. Umieszczenie zworek umożliwia alternatywne podawanie sygnału zegarowego „z guzika”. Takie rozwiązanie nie jest zalecane, ponieważ w przyciskach istnieje zjawisko drgania styków. Jedno naciśnięcie przycisku może wygenerować szereg impulsów, które kilkakrotnie przełączą układ sekwencyjny sterowany ręcznie wygenerowanym sygnałem zegarowym.

W zestawie umieszczono dwie grupy układów we/wy, które mogą działać dwukierunkowo (przyciski i diody LED). Nie wszystkie wyprowadzenia układu 9536 są używane. Ten fragment schematu jest istotny, ponieważ na jego podstawie będzie trzeba zdefiniować w pliku UCF przypisania sygnałów do fizycznych wyprowadzeń układu. Na płytce przy diodach i przyciskach są wytrawione numery, lecz są to numery kolejnych makrokomórek w bloku funkcyjnym a nie wyprowadzeń układu.

Pamiętać trzeba, że naciśnięcie przycisku powoduje podane nie wyprowadzenie układu 9536 niskiego stanu logicznego. Diody będą zapalać się od podania stanu niskiego na wyprowadzenie. Wynika to z przesłanek histo-

rycznych. Diodę LED możemy podłączyć do wyprowadzenia układu cyfrowego na dwa sposoby:



Dioda po lewej stronie będzie się zapalać przy podaniu zera logicznego na pinie. Dioda prawa zapali się od podania jedynki na pinie. Czemu łączy się diody tak, by zapalały się od stanu niskiego na pinie? Ponieważ dioda zapalana „jedynką" jest zasilana wprost z układu scalonego (napięcie pobierane z V_{CCINT}). Jeżeli mamy do zasilenia kilkanaście diod naraz, to źródło V_{CCINT} będzie obciążone, a to może spowodować niepożądane wahania napięcia. Lepsze jest rozwiązanie z diodą zasilaną z zewnętrznego źródła. Nie musimy martwić się o stabilność napięcia V_{CCINT}.

Na płytce znajdują się dodatkowe 4 wyprowadzenia portu JTAG (komunikacja podczas programowania i testowania układu programowalnego). Nie mogą one być współdzielone ze „zwykłymi" wyprowadzeniami we/wy, ponieważ komunikacja przez JTAG odbywać się musi w sposób nadrzędny. Musi być ona dostępna podczas pracy układu.

Inne cechy układu XC 9500 XL

Wspomnieć należy o możliwości konfiguracji makrokomórek do pracy w trybie obniżonego poboru mocy. Jest to konfiguracja indywidualna dla każdej makrokomórki. Wykorzystuje się atrybut:

```
PWR_MODE = LOW|STD
```

Atrybut należy dołączyć do przerzutnika, który umieścimy na schemacie. Przerzutnik zostanie umieszczony w którejś z makrokomórek i będzie ona pracowała w trybie obniżonego poboru mocy. Jeżeli będziemy wykorzystywać makrokomórkę do pracy kombinacyjnej, to atrybut „PWR_MODE" dołączamy do bramki logicznej. Obniżenie poboru mocy spowalnia pracę układu. Gdy normalnie t_{LOGI}=1ns, to dla obniżonego poboru mocy czas zwiększa się do t_{LOGILP}=5ns. Oszacowanie zysku jest trudne, ale można w przybliżeniu określić, że wartość prądu pobieranego przez makrokomórkę pracującą w trybie obniżonego poboru mocy stanowi 50% prądu pobieranego podczas normalnej pracy. Rozwiązanie to stosuje się, gdy nie zależy nam na szybkości pracy układu. Częstotliwość pracy układu może być obniżona z 170MHz do 50MHz.

Inną ciekawostką jest ochrona konfiguracji układu. Jest to istotne zagadnienie w układach programowalnych. Przydaje się to wówczas, gdy chcemy chronić nasz projekt przed jego skopiowaniem. Układy z rodziny XC 9500XL mają pamięć konfiguracji opartą o technikę Flash. W układach FPGA jest problem, bo pamięć konfiguracji jest oparta o pamięci RAM. Strumień konfigurujący jest pobierany z zewnętrznej pamięci, a to umożliwia podsłuchanie strumienia danych. W układach FPGA zaszyty jest klucz, przy pomocy którego jest szyfrowana zawartość pliku konfiguracyjnego. Utrudnia to konfigurację układów, bo trzeba szyfrować pliki konfiguracyjne osobno dla każdego układu.

W układach PLD mamy dwa bity sterujące zabezpieczeniami. Bit „Read Security" zabezpiecza przed odczytem. Drugi bit („Write Security") zabezpiecza przed omyłkowym nadpisaniem konfiguracji układu. Ustawienia tych bitów dokonuje się dzięki aplikacji programującej „Impact". Ustawienie bitu „Write Security" nie oznacza że układ jest stracony i nie będzie można go zaprogramować. Bit ten można wyzerować poprzez przesłanie odpowiedniego polecenia interfejsem JTAG. Oba bity są skutecznymi zabezpieczeniami na poziomie sprzętowym.

Język VHDL

Język ten należy do kategorii języków opisu sprzętu (Hardware Description Language). Historia języka sięga roku 1980. Pojawiła się wtedy potrzeba stworzenia profesjonalnego języka opisu sprzętu. Pojawiły się wtedy języki do opisu układów PAL (Palasm, inne zależne od producentów sprzętu). Ministerstwo obrony USA zaczęło finansować wieloletni projekt związany z projektowaniem układów cyfrowych wielkiej skali integracji i pracujących z dużymi prędkościami VHSIC (Very High Speed Integrated Circuits). Celem projektu było zagospodarowanie możliwości, które stały się realne dzięki rozwojowi technologii.

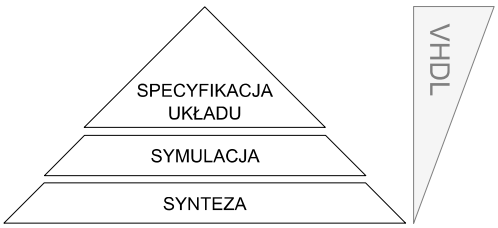
Technologia osiągnęła taki poziom, że praktycznie każdy mógł skonfigurować sobie programowalny układ cyfrowy. Potrzebny był język, który umożliwiałby efektywną pracę kilku grupom ludzi nad projektem jakiegoś układu. Język musiał być bardziej zaawansowany niż Palasm czy CUPL. W ten sposób powstał VHDL. Prace

nad językiem były zakrojone na szeroką skalę, co sprawiło że konieczne było opracowanie standardu opisującego ten język. VHDL doczekał się opracowania IEEE 1076.

Język bazował na koncepcjach zaczerpniętych z języka ADA. Język ADA był również opracowany na zlecenie ministerstwa. ADA opierał się na koncepcjach, uważanych w tamtych czasach za nowatorskie, pozwalał na opisywanie procesów wykonywanych współbieżnie, a to jest ważne w językach opisu sprzętu.

Kolejne standardy opisujące język VHDL pojawiały się w latach 1993 oraz 2000. W 1993 ustanowiono finalną wersję składni języka. Rok 2000 przyniósł ze sobą drobne poprawki.

VHDL można wykorzystywać do różnych rzeczy:



Pierwszym zastosowaniem języka był opis specyfikacji układu. Trzeba było określić jakie są sygnały wejściowe i wyjściowe, z jakich modułów będzie składał się układ i co te moduły będą robić.

Kolejnym zastosowaniem była symulacja. Jeżeli mamy opis działania jakiegoś układu, to chcielibyśmy dokonać symulacji układu. Symulacja w pełni automatyczna była nieosiągalna do przełomu lat 80-tych i 90-tych, kiedy to powstały pierwsze softwareowe symulatory.

Obecnie nas i projektantów najbardziej interesuje synteza. Opisujemy jakiś układ po to, by uruchomić automatyczne narzędzie, które wygeneruje nam plik konfiguracyjny. Plik przesyłamy do układu programowalnego, który działa zgodnie z opisem.

Należy sobie zdawać sprawę z tego, że VHDL nie nadaje się „do wszystkiego” Język ten został stworzony by umożliwić dokładny opis specyfikacji układu i potencjalnej symulacji. Łatwo jest napisać coś, co będzie składniowo poprawne, będzie się kiepsko symulowało i w ogóle nie będzie się syntezowało. Sama znajomość języka nie wystarcza. Trzeba wiedzieć jak języka używać by opis się dobrze syntezował lub symulował. Z tej to okazji skrót VHDL rozwijany był jako Very Hard Description Language. Pewnych rzeczy nie powinno się pisać w VHDL'u mimo że składniowo są one poprawne.

W pierwszej połowie lat 90-tych pojawiły się pierwsze automatyczne narzędzia syntezy. Rozróżniało się wtedy kod syntezowalny i niesyntezowalny. Każdy producent narzędzia syntezy specyfikował jakie kategorie opisu są dozwolone, a jakie nie są. Dzisiaj każdy producent chce być „jak najbardziej postępowy” wobec czego dziś jest coraz mniej konstrukcji niesyntezowalnych. Narzędzie będzie usiłowało zsyntezować nawet najgłupszy zapis i uda mu się to. Efekt takiej syntezy będzie koszmarny pod względem zajętości zasobów i własności czasowych. Dzisiaj jest jeszcze gorzej niż dawniej, ponieważ jakość syntezowanego projektu może być nieciekawa.