AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

Raport z projektu - Algorytm Quicksort

Systemy Dedykowane w Układach Programowalnych

Autor: *Zofia Wątroba, Maciej Ślęzak*
Kierunek studiów: *Mikroelektronika w Technice i Medycynie*

Kraków, 10-07-2023

Spis treści

1. Wprowadzenie - opis algorytmu

1.1. Opis algorytmu

Sortowanie szybkie (ang. quicksort) – jeden z popularnych algorytmów sortowania działających na zasadzie „dziel i zwyciężaj”. W strategii tej problem dzieli się rekurencyjnie na dwa lub więcej mniejszych podproblemów tego samego (lub podobnego) typu, tak długo, aż fragmenty staną się wystarczająco proste do bezpośredniego rozwiązania.

Zasada działania algorytmu

1. Z tablicy przeznaczonej do sortowania wybiera się element rozdzielający (ang. pivot), po czym tablica jest dzielona na dwa fragmenty: do początkowego przenoszone są wszystkie elementy nie większe od rozdzielającego, do końcowego wszystkie większe.
2. Potem tym samym sposobem sortuje się osobno początkową i końcową część tablicy.
3. Rekursja kończy się, gdy kolejny fragment uzyskany z podziału zawiera pojedynczy element, jako że jednoelementowa tablica nie wymaga sortowania.

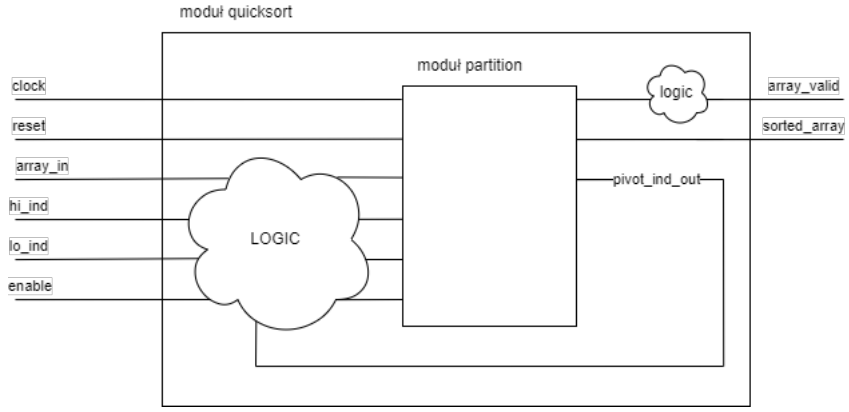
Złożoność algorytmu zależy od wyboru elementu rozdzielającego; jeśli podziały są zrównoważone algorytm jest tak szybki jak sortowanie przez scalanie, czyli $O(n \cdot logn)$, gdzie n to rozmiar tablicy sortowanej.

2. Implementacja algorytmu w języku Verilog

Sprzętowa implementacja algorytmu Quicksort w języku Verilog składa się z dwóch modułów.

Moduł nadrzędny o nazwie *quicksort* przetwarza tablicę wejściową dzieląc ją na mniejsze podzbiory, dobierając element rodzielający (pivot) oraz wywołując podrzędny moduł *partition*, którego zadaniem jest posortowanie elementów podzbioru na mniejsze i większe od pivota. Ważnym zadaniem modułu *quicksort* jest także przechowywanie wartości pivotów oraz indeksów, według których tablica wejściowa jest dzielona na mniejsze podzbiory, w zależności od kolejnych poziomów rekurencji algorytmu.

Wspomniany wyżej drugi moduł *partition* rozpatruje część tablicy wejściowej składającą się na aktualnie przetwarzany podzbiór pod względem aktualnego pivota, przenosząc elementy tablicy mniejsze od pivota na lewą, a większe na prawą stronę podzbioru. Jako pivot domyślnie i odgórnie przyjmowany jest ostatni element podzbioru. Schemat blokowy układu znajduje się na rysunku 2.1.



Rys. 2.1. Schemat blokowy układu realizującego algorytm Quicksort.

Poniżej znajduje się szczegółowy opis działania wymienionych wyżej modułów.

2.1. Moduł *quicksort*

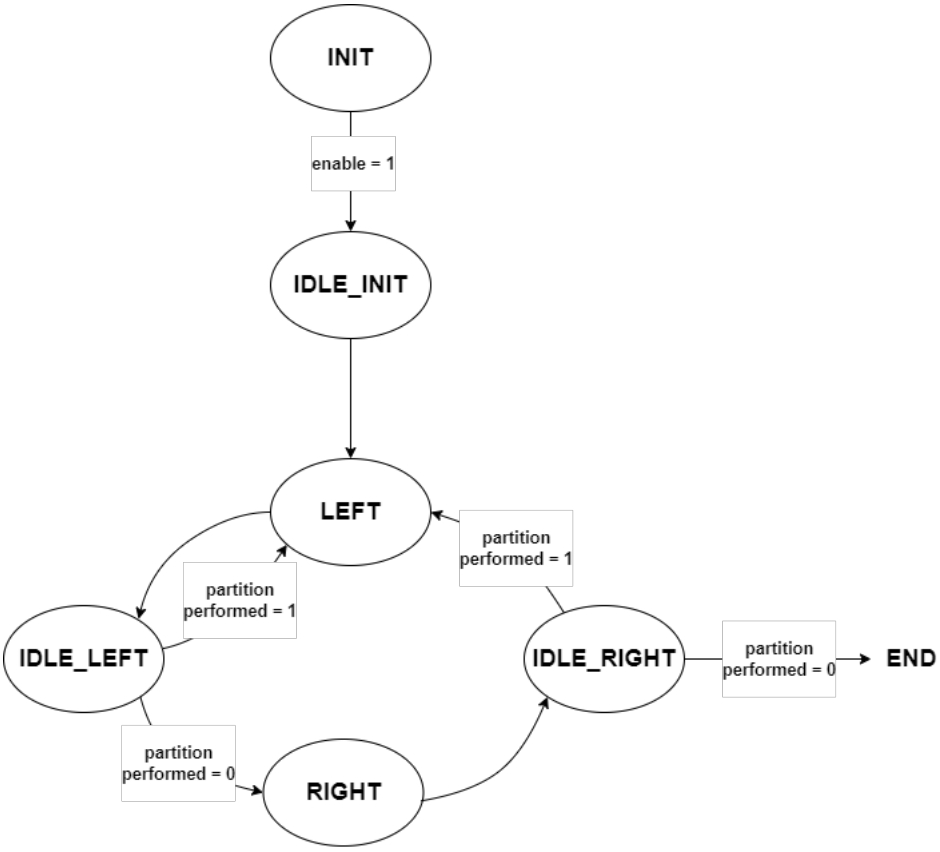
Moduł *quicksort* obsługuje następujące sygnały:

- sygnały wejściowe:
  - sygnał zegara (*clk*),
  - sygnał resetu (*reset*),
  - aktywacja modułu (*enable*),
  - początkowy indeks określający zakres tablicy wejściowej przeznaczony do posortowania (*lo_ind*),
  - końcowy indeks określający zakres tablicy wejściowej przeznaczony do posortowania (*hi_ind*),
  - wejściowa tablica przeznaczona do posortowania (*array_in*),
- sygnały wyjściowe:
  - sygnał oznaczający zakończenie sortowania (*array_valid*),
  - posortowana tablica (*sorted_array*).

Tablica wejściowa modułu jest tablicą **jednowymiarową**, w której wartość jednego elementu jest zapisana na 3 bitach tablicy.

Sygnał *array_valid* jest iloczynem logicznym wewnętrznych sygnałów *left_valid* i *right_valid*, które oznaczają, że w zarówno w stanie **LEFT**, jak i **RIGHT** nie było potrzeby wywoływania modułu *partition* a zatem tablica została już posortowana.

Realizacja algorytmu w module *quicksort* odbywa się w architekrurze maszyny stanów:



Rys. 2.2. Maszyna stanów

Stan **INIT** odpowiada za nadanie zmiennym wartości początkowych oraz oczekiwanie na sygnał *enable* rozpoczynający działanie algorytmu.

W stanie **IDLE_INIT** następuje pierwsze wywołanie modułu *partition* dla całej tablicy wejściowej. Na podstawie wyniku działania modułu *partition*, a dokładniej na podstawie zwróconego przez niego indeksu, na którym znajduje się pivot po wstępnym przesortowaniu tablicy, nastąpi podział tablicy wejściowej na mniejsze podzbiory, które będą sortowane rekurencyjnie w podobny sposób.

Stany **LEFT** oraz **RIGHT** odpowiadają za podział tablicy wejściowej na coraz mniejsze podzbiory zawierające elementy o wartościach odpowiednio mniejszych oraz większych od wartości pivota rozpatrywanego w poprzednim etapie algorytmu.

Stany **IDLE_LEFT** oraz **IDLE_RIGHT** sprawdzają, czy podzbiory przygotowane w stanach odpowiednio **LEFT** i **RIGHT** wymagają dalszego sortowania. Jeśli tak, następuje wywołanie modułu *partition*, oczekiwanie na jego zakończenie i powrót do odpowiadającego stanu (**LEFT** lub **RIGHT**). W przeciwnym wypadku następuje skok do następnego stanu w kierunku określonym przez pokazany

powyżej diagram.

W tak przyjętej architekturze maszyny stanów zakończenie działania algorytmu następuje w przypadku, gdy zarówno w stanie **IDLE_LEFT** oraz **IDLE_RIGHT** okaże się, że aktualny podzbiór nie wymaga sortowania. Oznacza to, że cała tablica została posortowana, więc następuje zakończenie działania algorytmu oraz powrót do stanu **INIT**.

2.2. Moduł *partition*

Moduł *partition* obsługuje następujące sygnały:

- sygnały wejściowe:
  - sygnał zegara (*clk*),
  - sygnał resetu (*reset*),
  - aktywacja modułu (*start*),
  - początkowy indeks określający zakres tablicy wejściowej przeznaczony do posortowania (*lo_ind*),
  - końcowy indeks określający zakres tablicy wejściowej przeznaczony do posortowania (*hi_ind*),
  - wejściowa tablica przeznaczona do posortowania (*array_in*)
- sygnały wyjściowe:
  - sygnał oznaczający zakończenie sortowania (*part_valid*),
  - posortowany fragment tablicy wejściowej (*sorted_array*),
  - indeks elementu rozdzielającego w tablicy po posortowaniu (*pivot_ind_out*) .

Moduł ten również oparty jest o (prostą) architekturę maszyny stanów:

W stanie **IDLE** moduł oczekuje na aktywację sygnałem *start*, po czym konwertuje tablicę wejściową na tablicę dwuwymiarową.

Następnie w stanie **PARTITION_PENDING** każdy z rozpatrywanego podzbioru tablicy wejściowej zostaje porównany do wartości pivota i na tej podstawie następuje podmiana kolejności elementów w taki sposób, aby finalnie elementy mniejsze od pivota znajdowały się po lewej, a większe po jego prawej stronie.

Po podmianie, w stanie **DONE** wystawiany jest sygnał *part_valid* sygnalizujący, że wartości na wyjściu modułu są gotowe do dalszego przetwarzania. Obecność dodatkowego stanu "wyjściowego"jest

potrzebna ze względu na potrzebę odczekania jednego cyklu po to, aby przetwarzana tablica mogła zostać przekonwertowana ponownie na tablicę jednowymiarową.

2.3. Testbench

Działanie algorytmu zostało sprawdzone za pomocą testbenchu napisanego również w języku Verilog. Jego zadaniem było rozpatrzenie wszystkich możliwych przypadków tablic wejściowych. Testbench kolejno generował tablicę, wykonwał na niej algorytm oraz sprawdzał czy wartości zostały posortowane w poprawny sposób.

3. Podsumowanie

Zaimplementowany przez nas algorytm służący do sortowania wartości po przetestowaniu działa prawidłowo. Jest to funkcjonalny prototyp, który w przyszłości może zostać rozszerzony zarówno dla większych liczb (na przykład 16-bitowych), bądź też dla większych rozmiarów tablic.