

# Systemy wbudowane (L1) - rozwiązania

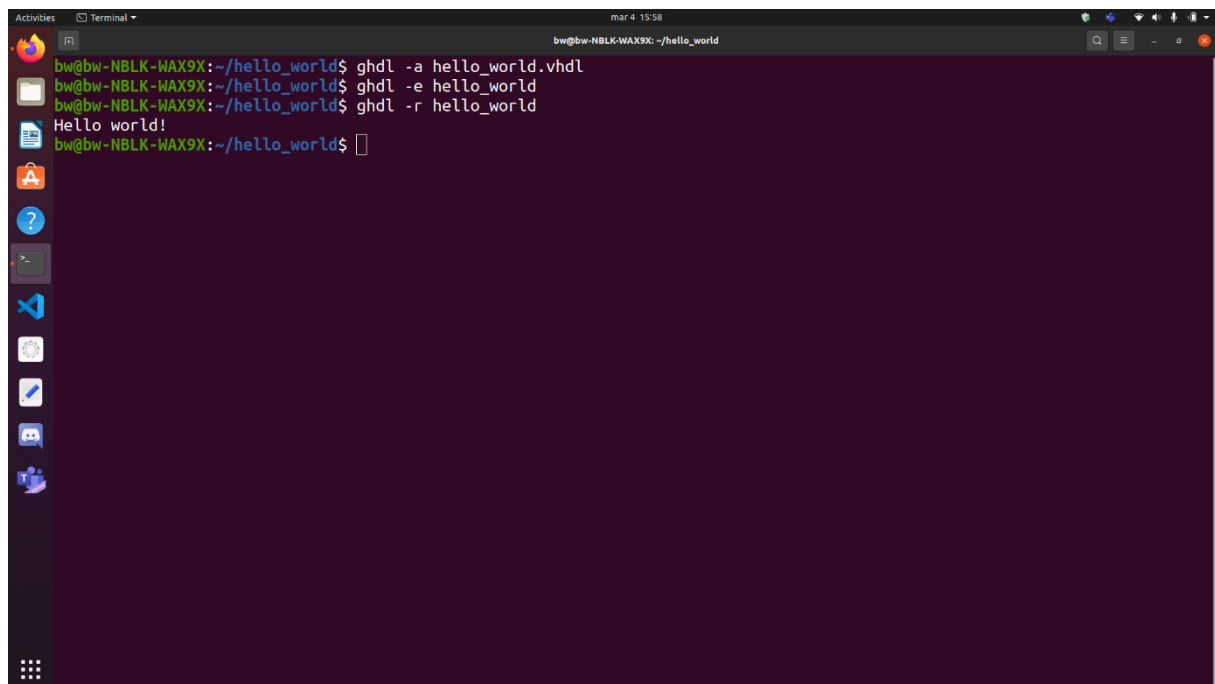
Błażej Wróbel, 250070, 3. rok, W11, informatyka

## Zadanie 1.

W zadaniu 1 należało przeanalizować przykład znajdujący się w dokumentacji narzędzia *GHDL*. W tym przykładzie był podany plik ze źródłami przykładowego programu, który wypisuje *'Hello world!'* na standardowe wyjście i sekwencja kroków, które pozwalają na skompilowanie i uruchomienie programów napisanych w języku *VHDL*. Tymi krokami są:

- Wykonanie polecenia *'ghdl -a hello\_world.vhdl'*. W tym kroku kompilujemy kod źródłowy.
- Wykonanie polecenia *'ghdl -e hello\_world'* (nazwa pliku nie może się kończyć na *.vhdl*). W tym kroku ponownie dokonujemy kompilacji, ale tym razem wszystkich obiektów (encje, architektury itp.), które zdefiniowaliśmy. W tym momencie również dołączamy wszystkie potrzebne biblioteki.
- Wykonanie polecenia *'ghdl -r hello\_world'*. Dzięki temu uruchamiamy plik wykonywalny.

Skutkiem wykonania wszystkich poprzednich kroków jest to:



```
Activities Terminal mar 4 15:58
bw@bw-NBLK-WAX9X: ~/hello_world
bw@bw-NBLK-WAX9X:~/hello_world$ ghdl -a hello_world.vhdl
bw@bw-NBLK-WAX9X:~/hello_world$ ghdl -e hello_world
bw@bw-NBLK-WAX9X:~/hello_world$ ghdl -r hello_world
Hello world!
bw@bw-NBLK-WAX9X:~/hello_world$
```

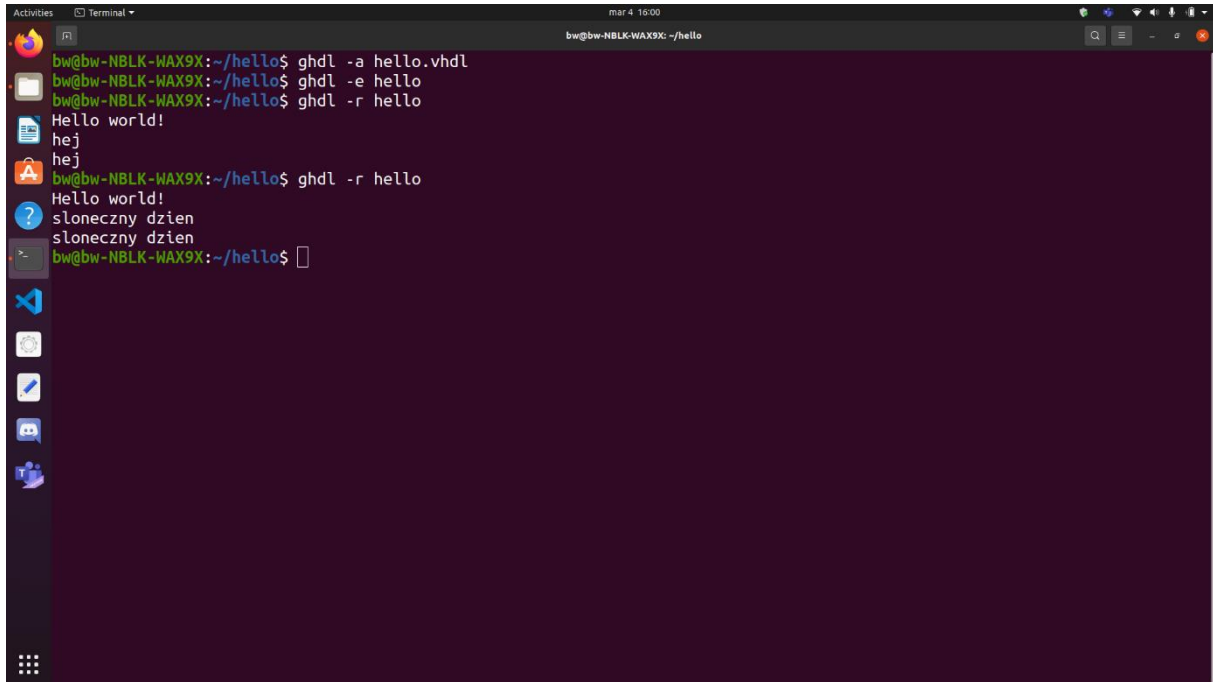
Po przeanalizowaniu źródeł programów można dostrzec, że język *VHDL* jest podobny do języka *Ada*. Oto przykładowe podobieństwa:

- ✓ Oba języki są silnie typowane (wszystkie zmienne muszą mieć określony typ, programista może w wygodny sposób definiować swoje własne typy).
- ✓ Podobnie dołącza się biblioteki zawierające użyteczne dla nas funkcje (słowo kluczowe *use*. W *Adzie* trzeba przed tym użyć słowa kluczowego *with*).
- ✓ Podział struktury programu na część deklaratywną (gdzie deklarujemy zmienne, tablice itd.) i na tą, gdzie wykonywane są obliczenia.
- ✓ Niemal identyczna składnia pętli *for*.

- ✓ Identyczny sposób odwoływania się do atrybutów typów złożonych (operator ' np. *patterns'range* z przykładu *adder*)

## Zadanie 2.

W programie z zadania 1 wystarczy zadeklarować nową zmienną typu **line** i użyć funkcji *getline(file F: text; L: out Line)*. Po skompilowaniu i uruchomieniu skutek jest następujący:



```
bw@bw-NBLK-WAX9X:~/hello$ ghdl -a hello.vhdl
bw@bw-NBLK-WAX9X:~/hello$ ghdl -e hello
bw@bw-NBLK-WAX9X:~/hello$ ghdl -r hello
Hello world!
hej
hej
bw@bw-NBLK-WAX9X:~/hello$ ghdl -r hello
Hello world!
słoneczny dzień
słoneczny dzień
bw@bw-NBLK-WAX9X:~/hello$
```

## Zadanie 3.

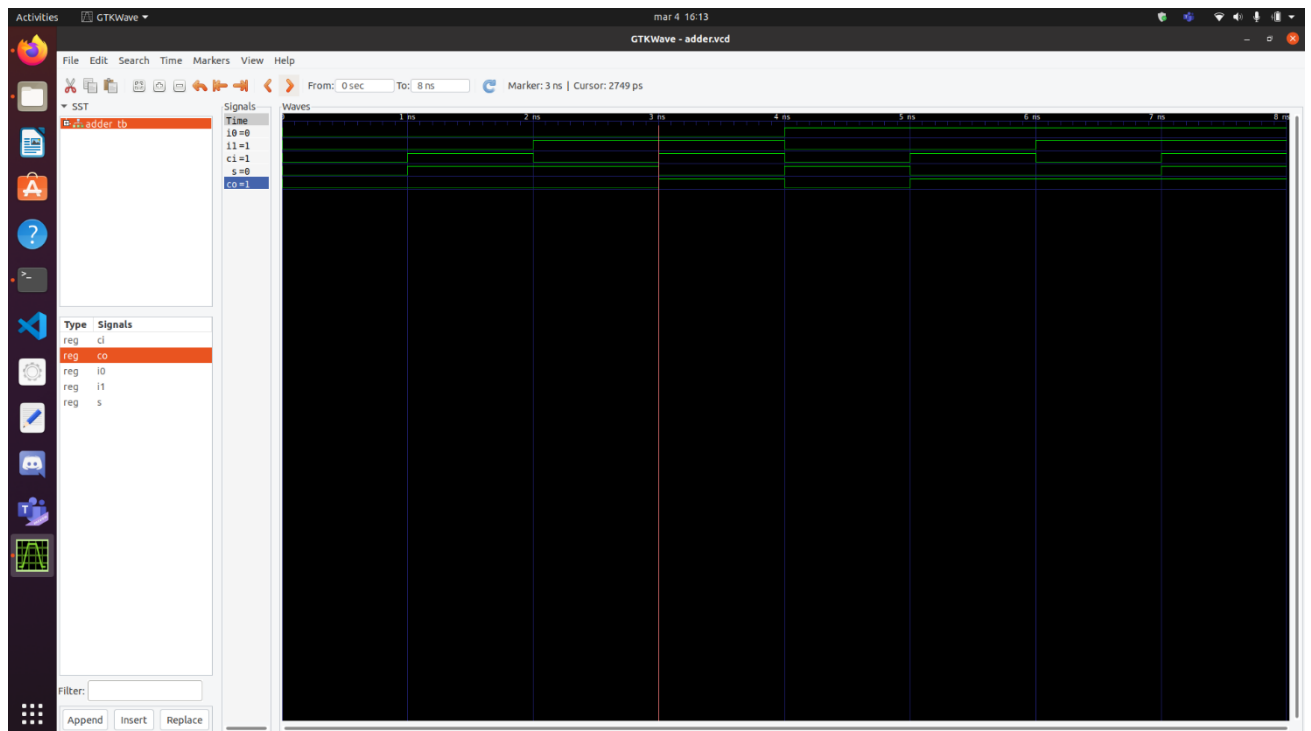
W tym zadaniu należało przeanalizować, przykład z dokumentacji, dotyczący sumatora pełnego. W tym przypadku najpierw zdefiniowaliśmy porty/sygnały, które występują w naszym systemie i określiliśmy ich typ (czy są *in*, *out*, *in out* lub *buffer*) oraz wartości, które na nich mogą się pojawiać (w tym przypadku *bit*). Następnie w ciele konstrukcji **architecture** zdefiniowaliśmy zależności między wejściami, a wyjściami:

$$S = I_0 \oplus I_1 \oplus C_{in}$$
$$C_0 = (I_0 \wedge I_1) \vee (C_{in} \wedge I_0) \vee (I_1 \wedge C_{in})$$

Gdzie  $\oplus$  oznacza xor.

Potem stworzyliśmy test, który sprawdzał poprawność działania stworzonego przez nas modelu. Samo skompilowanie pliku z opisem naszego modelu nic ciekawego nam nie dawało, bo nie mogliśmy zbadać działania/zachowania naszego układu. Dopiero stworzenie, skompilowanie i uruchomienie testu, pozwala nam zbadać i zweryfikować jego działanie.

Plik *.vcd* wygenerowałem następująco: *ghdl -r adder\_tb --vcd=adder.vcd*. Rezultat działania programu **gtkwave** na tym pliku:



A oto wynik działania samego testu:

```
Activities Terminal mar 4 16:02 bw@bw-NBLK-WAX9X: ~/adder
bw@bw-NBLK-WAX9X:~/adder$ ghdl -a adder.vhdl
bw@bw-NBLK-WAX9X:~/adder$ ghdl -a adder_tb.vhdl
bw@bw-NBLK-WAX9X:~/adder$ ghdl -e adder_tb.vhdl
/usr/bin/ghdl-mcode:error: bad unit name 'adder_tb.vhdl'
/usr/bin/ghdl-mcode:error: (a unit name is required instead of a filename)
bw@bw-NBLK-WAX9X:~/adder$ ghdl -e adder_tb
bw@bw-NBLK-WAX9X:~/adder$ ghdl -r adder_tb
adder_tb.vhdl:52:5:@8ns:(assertion note): end of test
bw@bw-NBLK-WAX9X:~/adder$ ghdl -r adder_tb --vcd=adder.vcd
adder_tb.vhdl:52:5:@8ns:(assertion note): end of test
bw@bw-NBLK-WAX9X:~/adder$
```

Należało również odpowiedzieć na następujące pytania:

### Co to jest adder i jak działa ?:

Sumator pełny jest układem logicznym służącym do dodawania bitów (z tego powodu zazwyczaj jest stosowany w kaskadzie z innymi sumatorami, co pozwala nam na dodawanie 8, 16, 32, 64 bitowych liczb binarnych). Zawiera trzy wejścia : na pierwszy i drugi bit, oraz przeniesienie (które być może powstało jako wynik dodawania na poprzednim sumatorze) oraz dwa wyjścia: na wynik i przeniesienie, które mogło powstać podczas dodawania. Wynik jest 'xorem' wszystkich trzech wejść

(co ma sens, bo jeśli mamy np. dwie wartości 1 i jedną wartość 0, to  $1 \text{ xor } 1 \text{ xor } 0 = 0$ , co zgadza się arytmetyką w systemie dwójkowym), a przeniesienie jest klauzulą:

$$C_{\text{out}} = (In_1 \wedge In_2) \vee (In_1 \wedge C_{\text{in}}) \vee (In_2 \wedge C_{\text{in}})$$

Gdzie:  $In_1$  – pierwszy bit,  $In_2$  – drugi bit,  $C_{\text{out}}$  – przeniesienie na wyjściu,  $C_{\text{in}}$  – przeniesienie na wejściu.

Widać, że to wyrażenie również będzie poprawnie ‘wyrażało’ przeniesienie, bo np. gdy mamy dwie jedynki na wejściu to wyrażenie przyjmie wartość 1.

### Czym są słowa **entity**, **architecture**, **port**, **component**, **process** ?:

#### Słowo **entity**:

Używając tego słowa tworzymy model/układ, który potem będziemy mogli opisać, uruchomić i przetestować. W ciele tej konstrukcji możemy wyspecyfikować porty/sygnały (tj. określić ich typy (*in*, *out*, *in out*, *buffer*)) oraz wartości, które będą się na nich pojawiać (*np. bit*).

#### Słowo **architecture**:

Za pomocą tej konstrukcji składniowej możemy określić sposób zachowania naszego modelu i podać zależności między poszczególnymi sygnałami (np. między sygnałami wejściowymi i wyjściowymi).

#### Słowo **port**:

Za pomocą tego słowa kluczowego możemy wyspecyfikować i opisać porty/sygnały występujące w naszym modelu (zazwyczaj stosujemy to słowo w ciele konstrukcji **entity**, przed słowem kluczowym **map** lub przy tworzeniu komponentu).

#### Słowo **component**:

Dzięki temu słowu kluczowemu możemy stworzyć ‘kopię’ naszego układu i potem wykorzystać ją w jakimś celu podczas symulacji.

#### Słowo **process**:

Dzięki tej konstrukcji składniowej możemy uruchomić symulację działania naszego układu (i np. przetestować go). Podczas wykonywania takiej symulacji, każda instrukcja w ciele tej konstrukcji jest wykonywana sekwencyjnie (po kolei) i po wykonaniu ostatniej instrukcji symulacja zaczyna się od początku (chyba, że użyjemy słowa kluczowego **wait**, które zatrzyma symulację po wykonaniu ostatniej instrukcji).

### W jaki sposób jest testowany sumator ?:

Sumator jest układem logicznym, więc posiada swoją tabelę prawdy i w oparciu o nią układ jest testowany. W programie testującym została stworzona tablica zawierająca krotki (rekordy) wartości wejść i odpowiadających im wartości wyjść. Następnie na wejście *kopii* (komponentu) są podawane wartości logiczne i za pomocą asercji sprawdzamy, czy dla danych wartości wejściowych, wartości wyjściowe zgadzają się z poprawnymi odpowiedziami (tj. tabelą prawdy). Jeśli któraś odpowiedź jest błędna to zwracamy odpowiedni komunikat błędu, a jeśli wszystko przeszło bez problemu to zwracamy komunikat o pozytywnym wyniku testu.

#### Zadanie 4.

W zadaniu 4 należało napisać kod jednostki i testujący dla podanego układu logicznego. Opis jednostki znajduje się w pliku *system.vhdl* , a kod testujący w pliku *system\_tb.vhdl* . Zależność między wejściem, a wyjściem przedstawia się następująco:

$$X = A \vee B \vee C$$

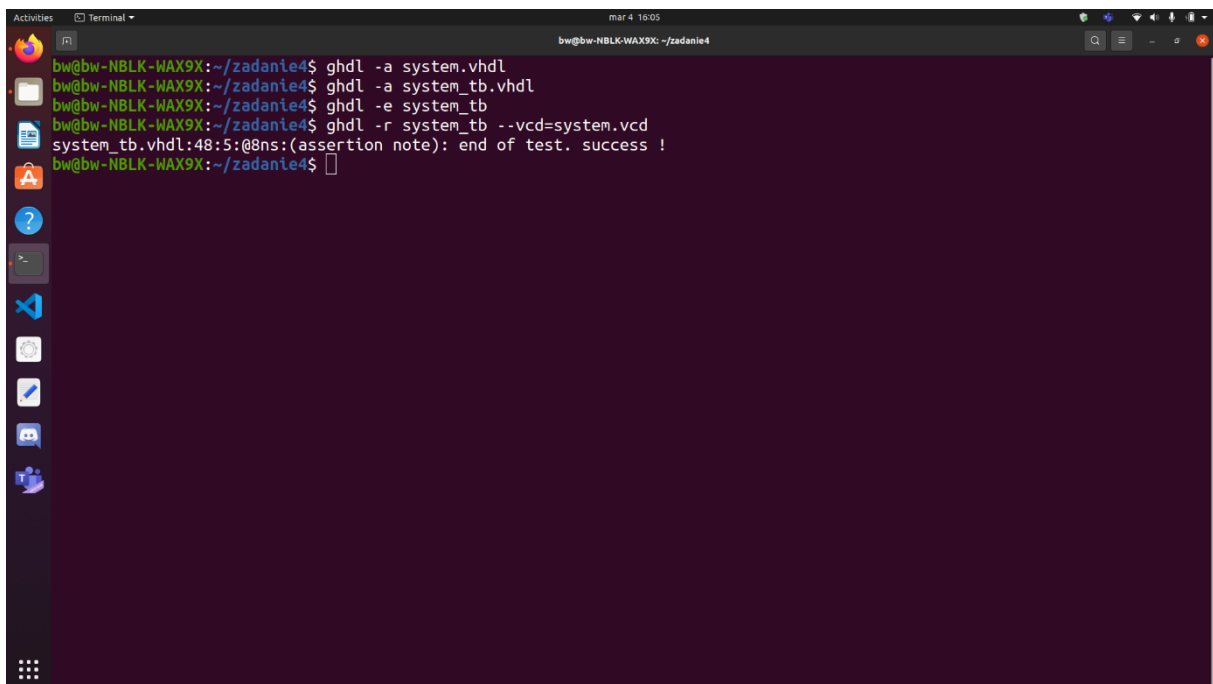
$$Y = (A \oplus C) \wedge (B \vee C)$$

Gdzie  $\oplus$  - oznacza xor.

Tabela prawdy dla podanego układu:

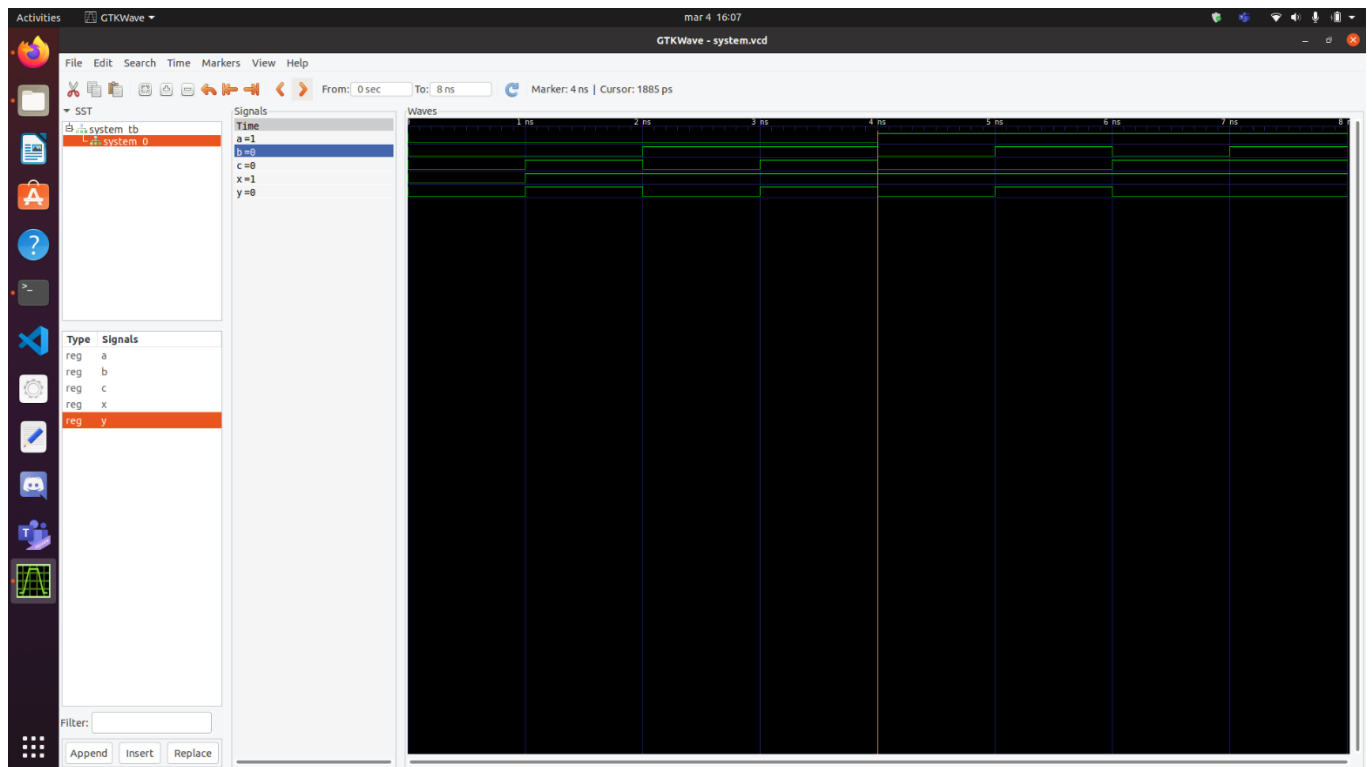
A	B	C	X	Y
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	1	0	1	1
1	0	1	1	0
1	1	1	1	0

Oto wynik działania testu:



```
bw@bw-NBLK-WAX9X:~/zadanie4$ ghdl -a system.vhdl
bw@bw-NBLK-WAX9X:~/zadanie4$ ghdl -a system_tb.vhdl
bw@bw-NBLK-WAX9X:~/zadanie4$ ghdl -e system_tb
bw@bw-NBLK-WAX9X:~/zadanie4$ ghdl -r system_tb --vcd=system.vcd
system_tb.vhdl:48:5:@8ns:(assertion note): end of test. success !
bw@bw-NBLK-WAX9X:~/zadanie4$
```

Oto wynik działania programu **gtkwave** dla wygenerowanego pliku *.vcd*:



W celu możliwości przetestowania **gtkwave**, dołączam w pliku *zip* pliki *adder.vcd* (przebiegi z przykładu *adder*) oraz *system.vcd* (przebiegi z bieżącego zadania).