2019 MSOC Assignment 2

Student: 傅子興
Student ID: B04901015

0. Generator / Coroutine
   a. Complete program execution flow

   [Python execution flow]

f0	g0	clk	f1	g1	clk	f2	g-2	clk	f3	g-fin	clk	f4	g-fill	clk	f-fin

   [Concept level execution flow]

f0		clk	f1		clk	f2		clk	f3		clk	f4		clk	f-fin
g0			g1			g2			g-fin			--			--

   b. Explain how *zip_iterator* works

   As shown in the code, for each of the passed in iterables, *zip_iterator* will make an iterator that aggregates their elements. For *zip_longest* function used in the code, it will fill a *fillvalue* for iterables with shorter length (in this case, no outputs will be generated if the iterable is shorter).

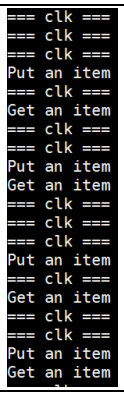
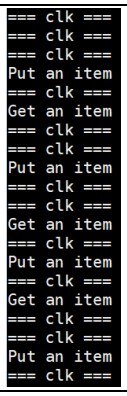
1. Producer / Consumer Revisit
   a. What if we use Consumer(11) ?

   The process will never stop, as the Consumer can never get the 11th item due to the limited item (in this case, 10) produced by the producer.

   b. What if we swap the order in *main_loop()* ?

   i. Is there any difference ?

   Yes, swapping results in different results. From the table above we can see that after swapping, the printed pattern shows different routine. A notable fact is that it never put and get the item in the same cycle.

Before swap		After swap	

ii. Is it reasonable ?

Yes. After swapping, the execution of request changes so that the *Consumer* will always check *n_item* first, causing the increase of *n_item* (push item) and reduce of *n_item* (get item) occur in different cycles.

2. Event (Implement all TODOs in 2_2_event_fifo.py)
3. dont_initialize
   a. What does *dont_initialize* mean in SystemC ?

      For simple process construction in SystemC, the declared process will always run once during the construction. If a *dont_initialize()* function is set, the process will not be executed until the clock signal starts.

   b. How do you implement it ?

      The core concept of *dont_initialize()* is that every process should not be waiting any events other than **INIT_EVENT** before the clock first triggered, i.e. the first process handled by the triggered **CLOCK_EVENT** should always be the *Clk()* process. Thus, I implement *dont_initialize* by simpling adding a yield **INIT_EVENT** in every process.

   c. Give an example about the difference with/without dont_initialize.

      The following table shows a simple difference of the two scenarios. Without *dont_initialize*, something strange like putting and getting item in the same cycle may occur, while this never happened with *dont_initialize*. This is because at the fisrt triggered **CLOCK_EVENT**, the first handled process is the *Producer()* process, causing the **WRITE_EVENT** triggering will be pushed to the *event_pending list* earlier than the next **CLOCK_EVENT**, resulting in the handling of get item (“Get an item”) process before the next clock edge (“handling 7”). On contrast, the with *dont_initialize* scenario will always trigger the **CLOCK_EVENT** earlier than any process triggering this cycle, causing the result that there must always exist a clock edge (“handling 7”) between putting and getting an item.

Without dont_initialize	With dont_initialize
handling 5 handling 7 handling 7 handling 7 Put an item handling 11 Get an item handling 7 handling 7	handling 5 handling 7 handling 7 handling 7 Put an item handling 7 handling 11 Get an item handling 7