VLSI – Dyskretna transformata Z w VHDL AMS

Zadanie 1.

Wyznacz wzór na odpowiedź **y[n]** integratora o podanych transmitancjach:

a) Backward Euler Integrator $H(Z) = \frac{z}{z-1}$

b) Forward Euler Integrator $H(Z) = \frac{1}{z-1}$

c) Bilinear Integrator $H(Z) = \frac{z+1}{z-1}$

Na podstawie wyznaczonych wzorów narysuj schemat blokowy każdego z układów. Wyznacz odpowiedzi układów na impuls Kroneckera i skok jednostkowy.

Zadanie 2.

Zaimplementuj w języku VHDL AMS obiekty, które wykorzystasz do zbudowania dyskretnych układów integratorów:

1. Układ **sample & hold**
2. Układ opóźniający $\mathbf{Z^{-n}}$
3. Układ kombinacji liniowej

Zadbaj, aby wszystkie układy zachowywały odpowiednie warunki początkowe (zmienna generic *initial*).

Układ 2.1. Sample & Hold

Układ **sample & hold** ma próbkować napięciowy sygnał wejściowy podawany na terminal *input.* Próbkowanie ma następować przy narastającym zboczu zegara *clk*. Skorzystaj z poniższego wzorca:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.electrical_systems.all;
use IEEE.MATH_REAL.all;

entity sample_hold is
      generic (
            initial : real := 0.0;
            clk_edge : std_logic := '1'
      );
      port(
            terminal input: electrical;
            clk: in std_logic;
            output: out real := initial
      );
end entity sample_hold;
```

Dopisz do układu architekturę realizującą założoną funkcję.

Układ 2.2. Układ opóźniający Z^{-n}

Układ opóźniający ma dwa porty wejściowe: sygnał zegarowy (std_logic) oraz sygnał wejściowy (*real*). Na wyjściu sygnał ma pojawić się z n-krotnym opóźnieniem. Opóźnienie zdefiniowane jest w formie parametru generic. Skorzystaj z poniższego wzorca:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.electrical_systems.all;
use IEEE.MATH_REAL.all;

entity ZDelay is
      generic (
            count : integer;
            initial : real := 0.0;
            clk_edge : std_logic := '1'
      );
      port(
            input: in real;
            clk: in std_logic;
            output: out real := initial
      );
end entity ZDelay;
```

Rejestr przesuwny (pamięć), potrzebny do realizacji układu można zaimplementować wykorzystując poniższą strukturę:

```
type holds_array is array ( 0 to count ) of real;
signal holds : holds_array := (others => initial);
```

Dopisz architekturę układu. Do zaimplementowania układu opóźniającego można posłużyć się pomocniczym układem opóźniającym, Z^{-1}, który połączony w n-elementowy ciąg realizuje opóźnienie $\mathbf{Z^{-n}}$. W tym celu można skorzystać z konstrukcji **generate** języka VHDL AMS:

```
generate_label: for N in 1 to count generate
      singleComponent: entity entity_name
            generic map ( … )
            port map ( … );
end generate generate_label;
```

Element pomocniczy może mieć postać:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.electrical_systems.all;
use IEEE.MATH_REAL.all;

entity singleZDelay is
      generic (
            initial : real := 0.0;
            clk_edge : std_logic := '1'
      );
      port(
            input: in real;
```

```
            clk: in std_logic;
            output: out real := initial
      );
end entity singleZDelay;
```

Układ 2.3. Układ kombinacji liniowej

Układ kombinacji liniowej na wejściu ma dwa sygnały typu *real* i sumuje je z odpowiednimi wagami (kombinacja liniowa). Współczynniki określone w postaci parametrów **generic**). Sugerowana struktura entity układu kombinacji liniowej:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.electrical_systems.all;
use IEEE.MATH_REAL.all;

entity linearCombination is
      generic (
            initial : real := 0.0;
            clk_edge : std_logic := '1';
            coeffA : real := 1.0;
            coeffB : real := 1.0
      );
      port(
            inputA: in real;
            inputB: in real;
            output: out real := initial
      );
end entity linearCombination;
```

Układ nie wymaga sygnału synchronizującego zegara. Zaprojektuj architekturę układu w sposób asynchroniczny.

Zadanie 3.

Zaimplementuj układy integratorów według wyznaczonych w zadaniu 1. schematów blokowych. Do ich implementacji należy wykorzystać wyłącznie układy kombinacji liniowej i opóźniające. Porty wejściowe integratora to sygnał typu real oraz sygnał zegarowy typu std_logic. Układ **sample & hold** będzie znajdować się w końcowym układzie testującym. Struktura entity integratora:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.electrical_systems.all;
use IEEE.MATH_REAL.all;

entity myInt is
      generic (
            initial : real := 0.0;
            clk_edge : std_logic := '1'
      );
      port(
```

```
            input: in real;
            clk: in std_logic;
            output: out real := initial
      );
end entity myInt;
```

Zadanie 4.

Korzystając z układów integratorów z zadania 3. przetestuj i porównaj działanie każdego z układów całkujących. Porównaj odpowiedzi układów na wymuszenia impulsem Kroneckera i skokiem jednostkowym z obliczeniami z zadania 1.

Wykorzystaj poniższy układ test-bench do przetestowania układu:

```
library ieee, EDULIB;
use ieee.std_logic_1164.all;
use ieee.electrical_systems.all;
use EDULIB.all;
use IEEE.MATH_REAL.all;
use work.all;

entity integrators_tb is
end entity integrators_tb;

architecture bench of integrators_tb is

      terminal sigCont_sin : electrical;
      terminal sigCont_1 : electrical;
      terminal sigCont_pulse : electrical;

      signal sigDiscr_sin : real := 0.0;
      signal sigDiscr_1 : real := 0.0;
      signal sigDiscr_pulse : real := 0.0;

      constant clk_hp : time := 10ns;
      signal out_backInt_sin : real := 0.0;
      signal out_backInt_1 : real := 0.0;
      signal out_backInt_pulse : real := 0.0;

      signal out_forwardInt_sin : real := 0.0;
      signal out_forwardInt_1 : real := 0.0;
      signal out_forwardInt_pulse : real := 0.0;
      signal out_bilinInt_sin : real := 0.0;
      signal out_bilinInt_1 : real := 0.0;
      signal out_bilinInt_pulse : real := 0.0;

      signal clk : std_logic := '0';

begin

      source_sin: entity v_sine
            generic map(
                freq => 1000000.0,
            amplitude => 12.0,
            phase => 0.0,
            offset => 0.0,
            ac_mag => 1.0
      )
```

```
	port map (
		pos => sigCont_sin,
		neg => ground
	);

	source_1: entity v_pulse
		generic map(
		initial => 0.0,          -- initial value [Volts]
		pulse => 1.0,                 -- pulsed value [Volts]
		ti2p => 0ns,          -- initial to pulse [Sec]
		tp2i => 0ns,          -- pulse to initial [Sec]
		delay => 0ns,          -- delay time [Sec]
		width => 1000ms,         -- duration of pulse [Sec]
		period => 1000ms)        -- period [Sec]
	port map (
		pos => sigCont_1,
		neg => ground
	);

	source_pulse: entity v_pulse
		generic map(
		initial => 0.0,          -- initial value [Volts]
		pulse => 1.0,                 -- pulsed value [Volts]
		ti2p => 0ns,          -- initial to pulse [Sec]
		tp2i => 0ns,          -- pulse to initial [Sec]
		delay => 0ns,          -- delay time [Sec]
		width => clk_hp+1ns,       -- duration of pulse [Sec]
		period => 1000ms)                    -- period [Sec]
	port map (
		pos => sigCont_pulse,
		neg => ground
	);

	sh_sin : entity sample_hold
		port map (
			input => sigCont_sin,
			output => sigDiscr_sin,
			clk => clk
		);

	sh_1 : entity sample_hold
		port map (
			input => sigCont_1,
			output => sigDiscr_1,
			clk => clk
		);

	sh_pulse : entity sample_hold
		port map (
			input => sigCont_pulse,
			output => sigDiscr_pulse,
			clk => clk
		);

	clk_proc: process
	begin
		clk <= '0';
		wait for clk_hp;
		clk <= '1';
```

```
		wait for clk_hp;
	end process clk_proc;

	backInt_sin : entity backInt
		port map (
			clk => clk,
			input => sigDiscr_sin,
			output => out_backInt_sin
		);

	backInt_1 : entity backInt
		port map (
			clk => clk,
			input => sigDiscr_1,
			output => out_backInt_1
		);

	backInt_pulse : entity backInt
		port map (
			clk => clk,
			input => sigDiscr_pulse,
			output => out_backInt_pulse
		);

	forwardInt_sin : entity forwardInt
		port map (
			clk => clk,
			input => sigDiscr_sin,
			output => out_forwardInt_sin
		);

	forwardInt_1 : entity forwardInt
		port map (
			clk => clk,
			input => sigDiscr_1,
			output => out_forwardInt_1
		);

	forwardInt_pulse : entity forwardInt
		port map (
			clk => clk,
			input => sigDiscr_pulse,
			output => out_forwardInt_pulse
		);

	bilinInt_sin : entity bilinInt
		port map (
			clk => clk,
			input => sigDiscr_sin,
			output => out_bilinInt_sin
		);

	bilinInt_1 : entity bilinInt
		port map (
			clk => clk,
			input => sigDiscr_1,
			output => out_bilinInt_1
		);
```

```
	bilinInt_pulse : entity bilinInt
		port map (
			clk => clk,
			input => sigDiscr_pulse,
			output => out_bilinInt_pulse
		);

end architecture bench;
```

Zadanie 5.

Zbadaj przebiegi sygnałów wyjściowych i porównaj z wykresami poniżej: