ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ
КАФЕДРА ЕОМ

СИСТЕМА КРУЇЗ-КОНТРОЛЮ НА ОСНОВІ ПЕРЕПРАГРАМОВАНОЇ FPGA

КВАЛІФІКАЦІЙНА РОБОТА
ДРУГИЙ (МАГІСТЕРСЬКИЙ) РІВЕНЬ

Автор:
Куліш Д.В.
ст. гр. КСМм-21-1

Керівник:
Горбачов В. О.
проф. каф. ЕОМ

МЕТА РОБОТИ

Метою кваліфікаційної роботи є розробка системи круїз-контролю на основі перепрограмованої FPGA.

Огляд сучасних систем круїз-контролю.

Програмна реалізація системи круїз-контролю, яка буде підтримувати задану швидкість автомобіля, незалежно від зовнішніх умов. Тестування розробленої системи в різних ситуаціях.

Аналіз отриманих результатів дослідження

АКТУАЛЬНІСТЬ ТЕМИ

В минулому сторіччі активно розроблялися системи допомоги водієві. Наприклад такі системі як: круїз-контроль, система контролю стійкості автомобіля, анти блокувальна система гальм, система контролю тяги та інші.

Система круїз-контролю встановлена майже на всіх сучасних автомобілях, автоматично регулює поздовжню швидкість транспорту за допомогою регулювання дросельної заслінки. Систему круїз-контролю автомобіля активує водій, який хоче підтримувати постійну швидкість під час тривалої їзди.

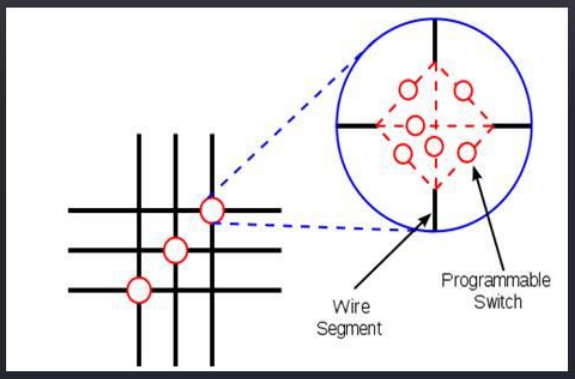
3

VIRTEX 7

- До двох мільйонів логічних комірок;
- Пропускна здатність 1866 Мбіт/с
- Об'єм пам'яті до 96Мбіт;
- Блок обробки аналогових сигналів;
- Динамічна часткова реконфігурація.

4

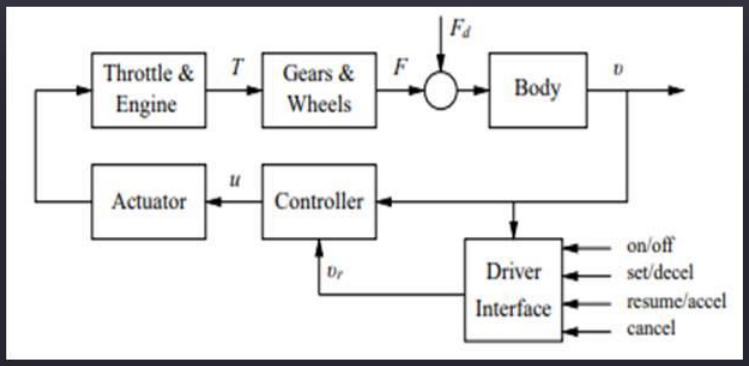
РЕКОНФІГУРАЦІЯ



FPGA складається з логічних блоків, що конфігуруються (CLB). CLB знаходяться в комутаційній матриці, яка визначає з'єднання входів і виходів для них. CLB складаються з блоку, що задає булеву функцію від кількох аргументів, він має назву таблиця відповідності (LUT) та тригера.

5

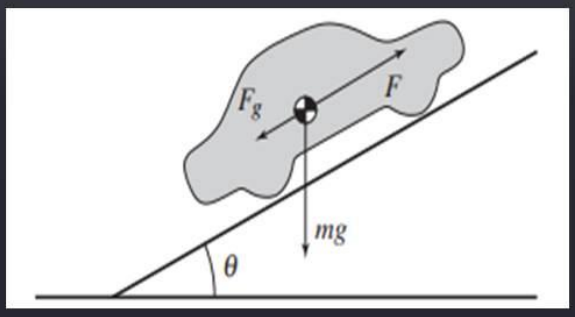
СИСТЕМА КРУЇЗ-КОНТРОЛЮ



Контролер приймає сигнали v і vr і генерує керуючий сигнал u, який надсилається на привід, що контролює положення дросельної заслінки.

6

МАТЕМАТИЧНА МОДЕЛЬ

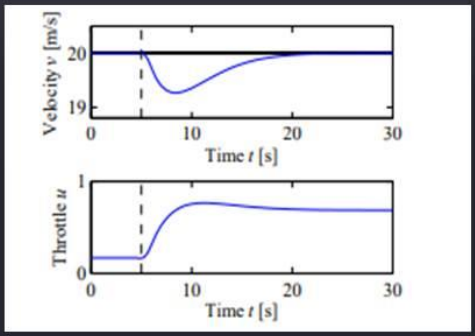


$$m\frac{dv}{dt} = F - F_d,$$

F – сила породжена контактом коліс із дорогою, а F_d – сила збурення через силу тяжіння, тертя та аеродинамічного опору.

7

РЕАКЦІЯ СИСТЕМИ НА ЗМІНУ КУТА НАХИЛУ ДОРОГИ

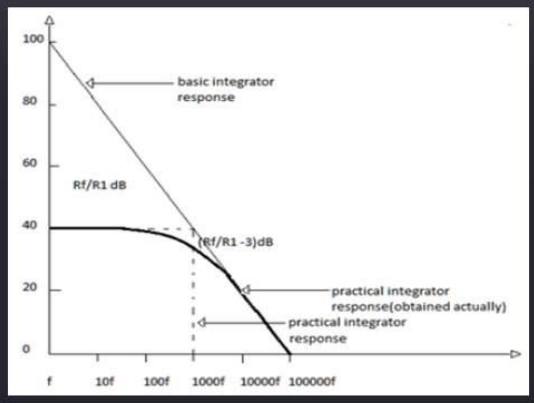


$$\frac{ds}{dt} = v_r - v, \quad u = k_p(v_r - v) + k_i s,$$

v – поточна швидкість, vr – бажана швидкість, κ_p – коефіцієнт пропорційності, κ_i – коефіцієнт інтеграції, s – стан інтегратора

8

ІНТЕГРАТОР



Інтегратор у застосуваннях вимірювання та керування — це елемент, вихідний сигнал якого є інтегралом часу від його вхідного сигналу. Він накопичує вхідну кількість протягом визначеного часу.

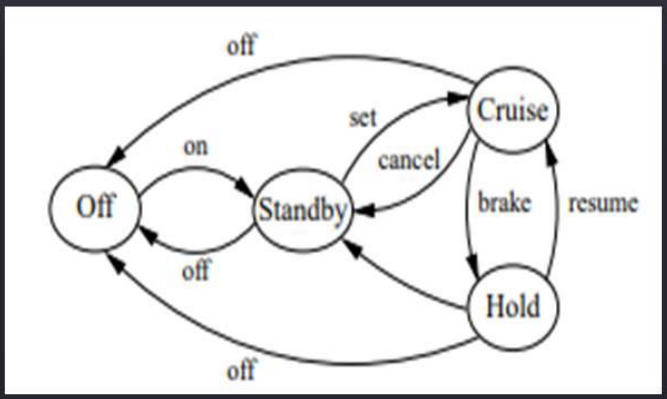
Інтеграція є важливою частиною багатьох інженерних і наукових застосувань

9

СКІНЧЕНИЙ АВТОМАТ ДЛЯ СИСТЕМИ КРУЇЗ-КОНТРОЛЮ

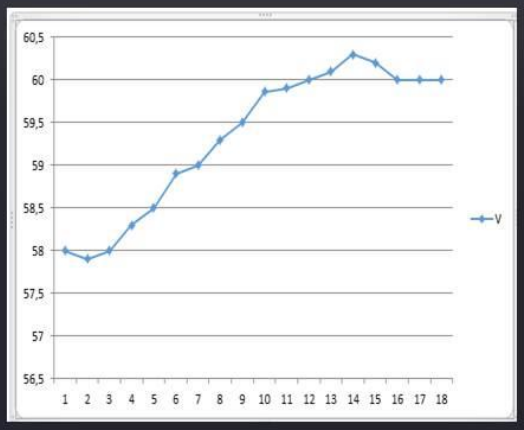
Стани

- вимкнено;
- стан очікування;
- стан круїз-контролю;
- стан підтримки бажаної швидкості.



10

РОБОТА СИСТЕМИ КРУЇЗ-КОНТРОЛЮ



Швидкість інтеграційно збільшується від 58 до 60км/год, коли досягнуто бажаної швидкості система підтримує її

11

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проаналізовано приклади сучасних реалізацій системи круїз-контролюі.

Було розроблено математичну модель руху транспортного засобу. В процесі аналізу математичної моделі було виведено функцію для роботи системи круїз-контролю

Було описано програмну реалізацію системи круїз-контролю, з використанням технології реконфігурації на базі SoC сімейства Virtex7.

Було проведено тестування розробленої системи в різних умовах.

Шляхи розвитку

- прогностичний круїз-контроль;
- адаптивний круїз-контроль.

12

ДОДАТОК Б

Фрагменти вихідного коду

```
entity fullAdder is
  port (c1, a1, a2 : in  bit;
        c2, s2     : out bit);
end fullAdder;

architecture fullAdder_arch of fullAdder is
begin
  s2 <= ((not c1) and (not a1) and a2) or
        ((not c1) and a1 and (not a2)) or
        (c1 and (not a1)and (not a2)) or
        (a1 and a2 and c1);
  c2 <= (a1 and c1) or (a2 and c1) or (a1 and a2);
end fullAdder_arch;

entity adder is
  generic (N : natural := 32);
  port (adder_a, adder_b : in  bit_vector (0 to N-1);
        adder_s    : out bit_vector (0 to N-1);
        adder_c    : out bit);
end adder;

architecture adder_arch of adder is
  component fullAdder
    port(c1, a1, a2 : in  bit;
         c2, s2     : out bit);
  end component;
  signal c_in : bit_vector (0 to N);

begin
  c_in(0) <= '0';
  adder : for i in 0 to N-1
  generate
    all_bit  :  fullAdder  port  map  (c1  =>  c_in(i),  a1  =>
adder_a(N-1-i), a2 => adder_b(N-1-i),
                            c2  => c_in(i+1), s2 => adder_s(N-1-
i));
  end generate adder;
  adder_c <= c_in(N);
end adder_arch;

LIBRARY ieee;
use IEEE.std_logic_1164.all;
USE ieee.numeric_std.ALL;

entity multiplier is
  generic (N : natural := 16);
```

```
  port (a, b : in  bit_vector (0 to N-1);
          s    : out bit_vector (0 to N-1)
          );
end multiplier;

architecture multiplier_arch of multiplier is
  component adder
    port(a1, a2 : bit_vector (0 to N-1);
         s2     : bit_vector (0 to N-1));
  end component;
  signal c_in : bit_vector (0 to N);
begin
process is
variable a_shift, sum : bit_vector (0 to N-1);
variable needAdd : bit;
begin

     a_shift := a;
     sum := "0000111010001110";
     for i in 0 to N-1 loop
     a_shift := a sll i;
          needAdd := b(i);
          add: if needAdd = '1' generate
          all_bit : adder port map (a, b, s);
          end generate add;
          end if;
     end loop;
     s <= sum;
     wait on a;
end process;

end multiplier_arch;

entity integrator is
  generic (N : natural := 16);
  port (a, b : in  bit_vector (0 to N-1);
          s    : out bit_vector (0 to N-1)
          );
end integrator;

architecture integrator_arch of integrator is
  component adder
    port(a1, a2 : bit_vector (0 to N-1);
         s2     : bit_vector (0 to N-1));
  end component;
  signal iteration : bit := '1'
begin
process is
variable sum : bit_vector (0 to N-1);
begin
     sum := a;
     adder : for i in 0 to N-1
     generate
```

```
    all_bit : fullAdder port map (sum => a, a2 => iteration, s2
=> s);
     end generate adder;
     s <= sum;
     wait on a;
end process;

end integrator_arch;

entity cruise_control is
  generic (N : natural := 16);
  port (v, vr : in  bit_vector (0 to N-1);
          u    : out bit_vector (0 to N-1)
          );
end cruise_control;

architecture cruise_control_arch of cruise_control is
  component multiplier
    port(v, vr : bit_vector (0 to N-1);
         u     : bit_vector (0 to N-1));
  end component;
begin
process is
variable temp_u : bit_vector (0 to N-1);
begin
     temp_u := "0000111010001110";

     IF input = '1' or '2' THEN
          u <= u;
     ELSIF input = '3' THEN
          add: generate
     all_bit : adder port map (v, vr, temp_u);
          end generate add;

          add: generate
          all_bit : multiplier port map (temp_u, k);
          end generate add;

          u <= temp_u;
          wait on v, vr;
     Else
          u <= u;
     end if;
end process;

end cruise_control_arch


library std;
use std.standard.all;

entity dip_e is
     port(
```

```
          actual_speed, need_speed : in REAL;
          U1 : out REAL
          );
end dip_e;

architecture dip_a of dip_e is
     signal U1_var : REAL;
begin
     process is
     begin
          --U1 <= 0.3;
          U1_var <= 0.5 * (need_speed - actual_speed) + 0.1 *
(need_speed - actual_speed);
          if U1_var >= 1.0 then
               U1 <= 0.9;
          elsif U1_var < 0.0 then
               U1 <= 0.00001;
          elsif need_speed - actual_speed < 0.5 then
               U1 <= 0.3;
          else
               U1 <= U1_var;
          end if;
          wait on need_speed, actual_speed;
     end process;
end dip_a;


entity hello is
end hello;

architecture empty of hello is
begin
     assert 1/=1 report "Hello,world";
end empty;
```