Boğaziçi University
Electrical and Electronic Engineering Department
Spring 2004 EE 598 Advanced Digital Design
Term Project

Analog-Digital Interface Circuit for a Line-Following Robot

Submitted to: Selçuk Öğrenci

Submitted by: Çağdaş Kayra Akman

Submission Date: 17.06.2004

Introduction

ROBOCON is one of the international competitions where individuals from all over the world have the opportunity to show how competent they are in technology. Since it is a robot competition, especially electronic control mechanisms combined with mechanical fine art is crucial for success. Although students from universities participate in this competition, they, inevitability, represent their countries as well. That the number of participants may easily increase to such a great number making a well organized event impossible, an elimination stage on national basis is have to passed in order to be able to go to the international event which be held in South Korea this year. National competition is organized by TRT and will take place on June 27th in Ankara.

Participation to this competition from our university has always been minimal. At most one group every couple of years. This year, three students from mechanical engineering department are building three robots for the competition. Two of them should be autonomous, the third will be controlled manually but using radio signals is not permitted. Although I personally know these students, it was after almost two months that I found out that they were in such a great project and that they found out they needed an electronic engineer right from the beginning but didn't bother to ask one. So since May, I am helping them in the electronic circuitry part of the project. The major task is making the autonomous robots find their ways on a grid of white strips on very dark green background. The method they employed initially, one they found using the Internet, was very complicated and they didn't have any room to make modifications. I designed a simple circuit providing a digital input about the status of the robot. A signal has to be generated when the robot leaves the white line. Also the direction had to be known. The circuit uses a couple of buffers, difference amplifiers and hex inverters with build-in Schmitt Trigger action, so that proper digital input for PIC microcontroller can be generated without any chatter. The input for this circuit is obtained using CNY70 optocouplers which are transistors whose emitter currents are controlled by the intensity of reflecting infrared light emitted by emitting diodes also in the same package with the transistors.

The motivation I have in that robot project made me decide to design the digital version of that analog-digital interface circuit.

Specifications

There are two sensors on the dark green surface and four, in this scenario, on the white strip. Because there is a grid of white strips instead of a curved white line to follow, robots should count the white strips that crossed to determine their position, one sensor on dark green surface wouldn't provide the necessary reference value. Two of the sensors on the white line gives signals on how much the robot deviated from the intended course in one of the directions... Proper PWM duty cycles according to these signals will be applied to H-bridges driving the electromotors. The other two keep track of deviations to opposite direction.

The emitter current of CNY70's sink the collector voltage due to a 20k resistor between collector and power supply. On dark green surface V_{CE} is ca. 4.7 V. On white strip it drops below 3.5 volts. ORing the V_{CE}'s from the sensors on the dark surface provides the reference value, the higher of the two. This one and the output signal from the sensor on white line are

passed through buffers and supplied to a difference amplifier which amplifies the difference so that a hex inverter is provided with about 3.5 V V_{in} when the robot is on the course, and awith0.6 V or less when it looses its way. These voltage levels are enough to drive 7414 Hex Inverters with Schmitt Trigger action. 7414's provide the necessary TTL compatible output signals to drive the microcontroller. Buffers and difference amplifiers are implemented using LM324 quad op-amps operating with a single power supply. The circuit is quite independent to visible light intensity due to infrared light employed, but it very sensitive to distance of the coupler to reflecting surface. This especially affects, lowers, the voltage difference on dark surface and white line sensors. To avoid possible problems, a POT is added to the difference amplifier providing asymmetry to the circuit and increasing the gain.

The digital design is based on these measured voltage levels. The only assumption is that all the signals from the sensors are converted to 4 bit words by an appropriate ADC. The possible decreases in the voltage difference due to increasing height from the surface are not resolved in the digital version assuming this difference will never go below 0.6 Volt.

The circuit consists of three sections. The first stage always provides the higher of the input voltages from the sensors on dark surface. To accomplish this, a nibble comparator and a multiplexer whose Select input is controlled by the comparator are designed. 4 subtracting circuits subtract the signals from the sensors on white line from this reference. The four 4–bit outputs and an additional signal per subtractor going high in case of negative result feed an interpreter circuit providing 4-bit words, high bits of which correspond to sensors on dark green surface.

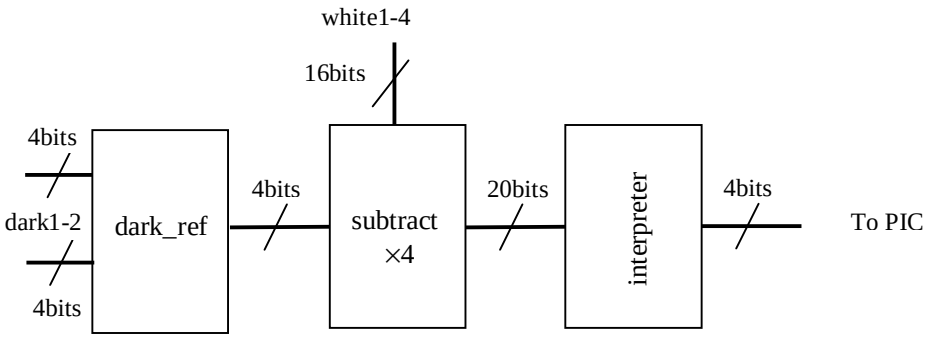


Fig. 1 Block Diagram of the Digital Circuit

Code Listing 1: Control Data *control_data.vhd*

```
package control is
component
dark_ref port(dark1, dark2: in bit_vector(3 downto 0);darkref:out
bit_vector(3 downto 0));
end component;
component
subtract port(input1, input2: in bit_vector(3 downto 0); output: out
bit_vector(3 downto 0); negative: out bit);
end component;
component
interpreter port(input: in bit_vector(19 downto 0); output: out
bit_vector(3 downto 0));
end component;
end control;
```

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE work.control.all;

entity control_data is
port(dark1, dark2, white1, white2, white3, white4: in bit_vector(3 downto
0); data: out bit_vector(3 downto 0));
end control_data;

architecture structural of control_data is
signal darkref: bit_vector(3 downto 0);
signal sub: bit_vector(19 downto 0);
begin
c0: dark_ref port map(dark1, dark2, darkref);
c1: subtract port map(darkref, white1, sub(3 downto 0), sub(4));
c2: subtract port map(darkref, white2, sub(8 downto 5), sub(9));
c3: subtract port map(darkref, white3, sub(13 downto 10), sub(14));
c4: subtract port map(darkref, white4, sub(18 downto 15), sub(19));
c5: interpreter port map(sub,data);
end structural;
```

Stage 1: Reference Voltage

There are two main blocks in this stage: Nibble comparator and 4-bits 2-to-1 multiplexer (under the name select_higher). The nibble comparator is composed of four cascaded 2-bit comparators. The multiplexer is composed of four 1-bit 2-to-1 multiplexers all controlled with the same Select input . The Select signal is generated from the three outputs of the comparator according to the following Boolean function: Select = (a_gt_b OR a_eq_b) OR NOT a_lt_b. So dark 1 is selected when it is greater than or equal to dark 2. Otherwise dark2 selected.

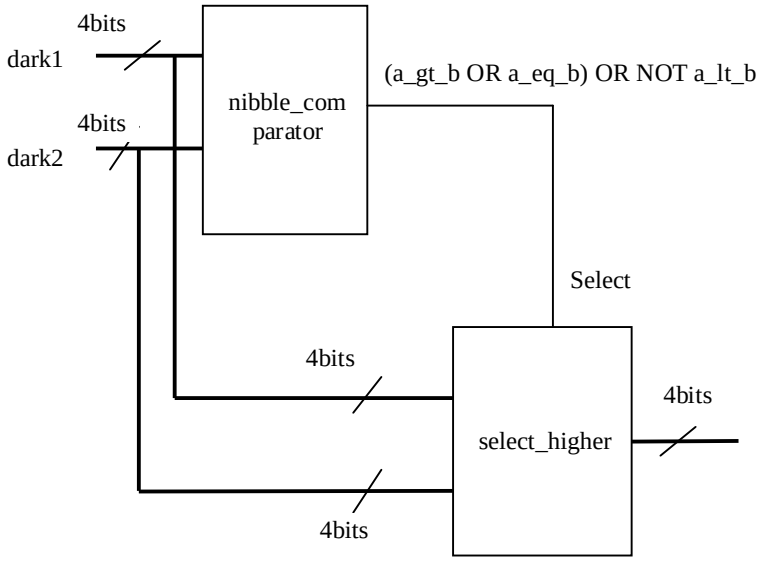


Fig. 2 Block Diagram of Stage 1

Code Listing 2: 2-bit comparator *comparator2.vhd*

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
ENTITY comparator2 IS
PORT(
a, b, AgtB, AeqB, AltB : IN BIT;
a_gt_b, a_eq_b, a_lt_b : OUT BIT);
END comparator2;

ARCHITECTURE dataflow OF comparator2 IS
BEGIN
a_lt_b <=NOT ((((NOT a) NAND b) AND (NOT a  NAND AltB)) AND (b NAND AltB));
a_eq_b <=(NOT ((a AND b)AND AeqB) ) NAND (NOT (NOT A AND NOT b AND AeqB));
a_gt_b <=NOT((a NAND AgtB) AND (NOT b NAND AgtB) AND (a NAND NOT b));
END dataflow;
```

Code Listing 3: Nibble comparator *nibble_comparator.vhd*

```
package comp is
component
comparator2 port(a, b, AgtB, AeqB, AltB : IN BIT;a_gt_b, a_eq_b, a_lt_b :
OUT BIT);
end component;
end comp;

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
use work.comp.all;

entity nibble_comparator is
port(a,b: in bit_vector(3 downto 0); gt, eq, lt: in bit; a_gt_b, a_eq_b,
a_lt_b: out bit);
end nibble_comparator;

architecture structural of nibble_comparator is

signal im: bit_vector(0 to 8);
begin
c0:comparator2 port map(a(0), b(0), gt, eq, lt,im(0), im(1), im(2));
c1:comparator2 port map(a(1), b(1), im(0), im(1), im(2),im(3), im(4),
im(5));
c2:comparator2 port map(a(2), b(2), im(3), im(4), im(5),im(6), im(7),
im(8));
c3:comparator2 port map(a(3), b(3), im(6), im(7), im(8),a_gt_b, a_eq_b,
a_lt_b);
end structural;
```

Code Listing 4: 1-bit 2-to-1 multiplexer *mux2to1.vhd*

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;

entity mux2to1 is
port(a, b, sel: in bit; output: out bit);
end mux2to1;

architecture signalflow of mux2to1 is
```

```
begin
output <= (a AND sel) OR (b AND NOT sel);
end signalflow;
```

Code Listing 5: 4-bit 2-to-1 multiplexer *select_higher.vhd*

```
package multiplex is
component
mux2to1 port(a, b, sel: in bit; output: out bit);
end component;
end multiplex;

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE work.multiplex.all;

entity select_higher is
port(input1, input2: in bit_vector (3 downto 0); sel: in bit; higher: out
bit_vector(3 downto 0));
end select_higher;

architecture structural of select_higher is
begin
c0: mux2to1 port map(input1(0), input2(0),sel, higher(0));
c1: mux2to1 port map(input1(1), input2(1),sel, higher(1));
c2: mux2to1 port map(input1(2), input2(2),sel, higher(2));
c3: mux2to1 port map(input1(3), input2(3),sel, higher(3));
end structural;
```

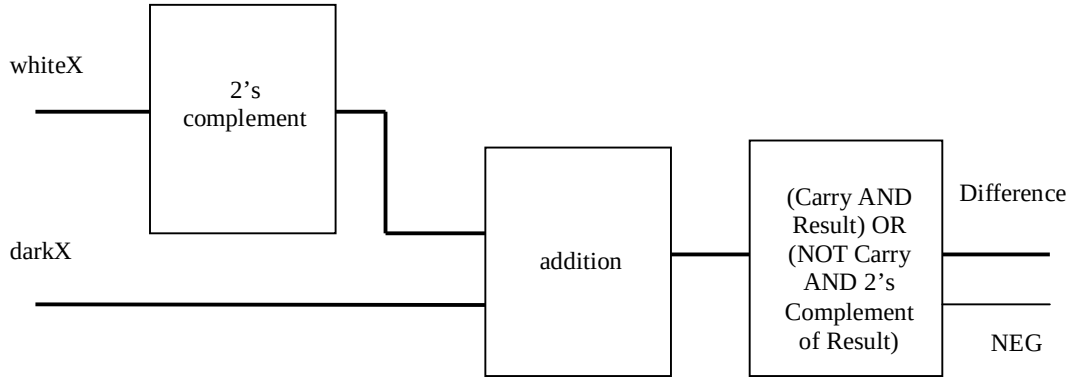
Code Listing 6: Dark Reference *dark_ref.vhd*

```
package nibble is
component
nibble_comparator port(a,b: in bit_vector(3 downto 0); gt, eq, lt: in bit;
a_gt_b, a_eq_b, a_lt_b: out bit);
end component;
end nibble;

package higher is
component select_higher
port(input1, input2: in bit_vector (3 downto 0); sel: in bit; higher: out
bit_vector(3 downto 0));
end component;
end higher;

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE work.nibble.all;
USE work.higher.all;

entity dark_ref is
port(dark1, dark2: in bit_vector(3 downto 0);darkref:out bit_vector(3
downto 0));
end dark_ref;

architecture structural of dark_ref is

signal im: bit_vector(2 downto 0);
begin
```

```
c1: nibble_comparator port map(dark1, dark2,'0','1','0',im(2), im(1),
im(0));
c2: select_higher port map(dark1, dark2, (im(2)OR im(1)) OR NOT im(0),
darkref);
end structural;
```

Stage 2: Subtraction

Te reference signal (4-bir word) is used to find out whether the robot is on the white strip or not. When it moves away from the white strip, the sensors which initially were on the line generate similar voltages as the ones on dark surface. Taking the difference (dark1/2 – white1-4) gives us this information. 2's complement of the subtrahend (whiteX value in this design) is added to the minuend (darkX). The result generates a carry when darkX > whiteX. In this case, the carry is discarded and the result is in normal form. If whiteX is greater than darkX, then the result is in 2's complement form and no carry generated. Taking the 2's complement of this result gives us the absolute value of the result. A minus sign should be used to indicate the correct value. This is done by one pin called NEG per subtractor. This block contains a component that takes the 2's complement. That block uses a 4-bit full adder composed of 4 one bit full adders. According to the carry, either the result of the addition or the 2's complement of it is set to the output. In the latter case the NEG pin goes high to indicate that the result negative. There are 20 output pins going to the interpreter circuit.



Code Listing 7: Full Adder *full_adder.vhd*

```
Library IEEE;
USE IEEE.std_logic_1164.all;

entity full_adder is
port(a, b, cin: in bit; o, cout: out bit);
end full_adder;

architecture dataflow of full_adder is

begin
o <= (a XOR b) XOR cin;
cout <= (a AND b) OR ((a XOR b) AND cin);
end dataflow;
```

Code Listing 8: 4-bits Full Adder *full_adder_4bits.vhs*

```
package adder is
component
full_adder port(a, b, cin: in bit; o, cout: out bit);
end component;
end adder;

Library IEEE;
USE IEEE.std_logic_1164.all;
USE work.adder.all;

entity full_adder_4bits is
port(input1, input2: in bit_vector(3 downto 0); Cin: in bit; output: out
bit_vector(3 downto 0); Cout: out bit);
end full_adder_4bits;

architecture structural of full_adder_4bits is
signal im: bit_vector(2 downto 0);
begin
c0: full_adder port map(input1(0), input2(0), Cin,   output(0), im(0));
c1: full_adder port map(input1(1), input2(1), im(0), output(1), im(1));
c2: full_adder port map(input1(2), input2(2), im(1), output(2), im(2));
c3: full_adder port map(input1(3), input2(3), im(2), output(3), Cout);
end structural;
```

Code Listing 9: 2's Complement *comlpemet_2s.vhd*

```
package adder4bits is
component
full_adder_4bits port(input1, input2: in bit_vector(3 downto 0); Cin: in
bit; output: out bit_vector(3 downto 0); Cout: out bit);
end component;
end adder4bits;

Library IEEE;
USE IEEE.std_logic_1164.all;
use work.adder4bits.all;

entity complement_2s is
port(input: in bit_vector(3 downto 0); output: out bit_vector(3 downto 0));
end complement_2s;

architecture dataflow of complement_2s is
signal im: bit_vector(3 downto 0);
begin
im(0) <= NOT input(0);
im(1) <= NOT input(1);
im(2) <= NOT input(2);
im(3) <= NOT input(3);
c0: full_adder_4bits port map(im,"0001",'0',output,open);
end dataflow;
```

Code Listing 10: Subtraction *subtract.vhd*

```
package complement is
component
complement_2s port(input: in bit_vector(3 downto 0); output: out
bit_vector(3 downto 0));
end component;
end complement;

package adder4 is
component
full_adder_4bits port(input1, input2: in bit_vector(3 downto 0); Cin: in
bit; output: out bit_vector(3 downto 0); Cout: out bit);
end component;
end adder4;

Library IEEE;
USE IEEE.std_logic_1164.all;
USE work.complement.all;
USE work.adder4.all;

entity subtract is
port(input1, input2: in bit_vector(3 downto 0); output: out bit_vector(3
downto 0); negative: out bit);
end subtract;

architecture dataflow of subtract is
signal im1: bit_vector(3 downto 0);
signal im2: bit_vector(3 downto 0);
signal im3: bit_vector(3 downto 0);
signal carry: bit;
begin
c0: complement_2s port map(input2,im1);
c1: full_adder_4bits port map(input1, im1, '0', im2, carry);
c2: complement_2s port map(im2, im3);
output(0) <= (carry AND im2(0)) OR (NOT carry AND im3(0));
output(1) <= (carry AND im2(1)) OR (NOT carry AND im3(1));
output(2) <= (carry AND im2(2)) OR (NOT carry AND im3(2));
output(3) <= (carry AND im2(3)) OR (NOT carry AND im3(3));
negative <= NOT carry;
end dataflow;
```

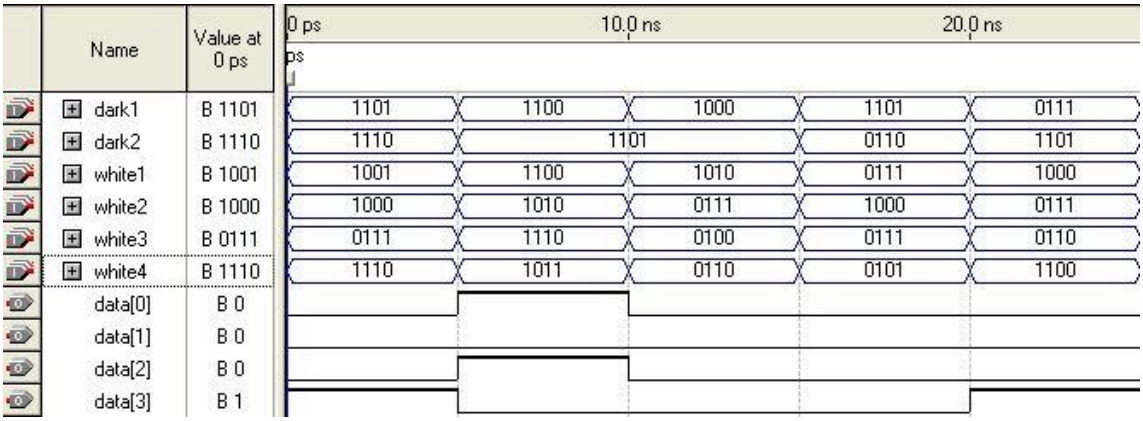
Stage 3: Interpreter

Each subtractor circuit has 5 output pins. But only 4 of them are used to generate the necessary control word. Since 4 bits resolution correspond to 16 levels, each combination of 4 bits correspond to a voltage range of (5 / 16) 0.3125 V. The interpreter circuit makes an output pin HIGH when the corresponding white strip sensor's voltage level becomes as close as 0.62 volt, i.e. the difference is "0001" or "0000" or if it is negative.

Code Listing 11: Interpreter *interpreter.vhd*

```
package compare is
component
nibble_comparator port(a,b: in bit_vector(3 downto 0); gt, eq, lt: in bit;
a_gt_b, a_eq_b, a_lt_b: out bit);
end component;
end compare;

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE work.compare.all;

entity interpreter is
port(input: in bit_vector(19 downto 0); output: out bit_vector(3 downto
0));
end interpreter;

architecture signalflow of interpreter is

begin

output(0) <= input(4) OR NOT (input(1) OR input(2) OR input(3));
output(1) <= input(9) OR NOT (input(6) OR input(7) OR input(8));
output(2) <= input(14) OR NOT (input(11) OR input(12) OR input(13));
output(3) <= input(19) OR NOT (input(16) OR input(17) OR input(18));

end signalflow;
```

Simulation Results



First column:
dark2 > dark1 => dark_ref = dark2;
“1110”-“1110” = “0000” < “0001” => data(3) should be high.
All other are smaller than white4, so others should be low too.

Second column: dark2 > dark1 => dark_ref = dark2;
“1101”-“1100” = “0001” => data(0) should be high;
“1101”-“1010“ = “0011” > “0001” => data(1) should be low;
“1101”-“1110” < 0 => data(2) should be high;
“1101”-“1011” = “0010” > “0001” => data(3) should be high;

Third column: dark2 > dark1 => dark_ref = dark2;
“1101”-“1010” = “0011” > “0001” => data(0) should be high;
All other are smaller than white1, so others should be low too.

Fourth column: dark 2 quite low, but the higher one is used as reference. Since the highest of four white sensors (“1000”) is much lower than “1101”, but it is greater than the other dark sensor, all output pins should remain low.

Fifth column:
dark2 > dark1 => dark_ref = dark2;
“1101”-“1100” = “0001” => data(3) should be high. The highest of the other three (“1000”) is “0101” less than dark2, so others should be low.

Simulation results show that the circuit functions as intended.

Analysis, Synthesis and Fitting Results for Different Devices

1.

```
+-----------------------------------------------------------------------+
; Analysis & Synthesis Summary                                          ;
+-----------------------------+-----------------------------------------+
; Analysis & Synthesis Status ; Successful - Thu Jun 17 22:28:44 2004 ;
; Revision Name               ; control_data                            ;
; Top-level Entity Name       ; control_data                            ;
; Family                      ; ACEX1K                                  ;
; Total logic elements        ; 35                                      ;
; Total pins                  ; 28                                      ;
; Total memory bits           ; 0                                       ;
; Total PLLs                  ; 0                                       ;
+-----------------------------+-----------------------------------------+
+-------------------------------------------------------------+
; Fitter Summary                                              ;
+----------------------+--------------------------------------+
; Fitter Status        ; Successful - Thu Jun 17 22:28:48 2004 ;
; Revision Name        ; control_data                          ;
; Top-level Entity Name ; control_data                         ;
; Family               ; ACEX1K                               ;
; Device               ; EP1K10TC100-1                        ;
```

; Total logic elements ; 35 / 576 (6 %) ;
; Total pins ; 28 / 66 (42 %) ;
; Total memory bits ; 0 / 12,288 (0 %) ;
; Total PLLs ; 0 / 1 (0 %) ;
+---------------------+-------------------------------------+

2.

+--+
; Analysis & Synthesis Summary ;
+---------------------------+--------------------------------------+
; Analysis & Synthesis Status ; Successful - Thu Jun 17 22:33:28 2004 ;
; Revision Name ; control_data ;
; Top-level Entity Name ; control_data ;
; Family ; APEX II ;
; Total logic elements ; 44 ;
; Total pins ; 28 ;
; Total memory bits ; 0 ;
; Total PLLs ; 0 ;
+---------------------------+--------------------------------------+

+--+
; Fitter Summary ;
+---------------------+------------------------------------+
; Fitter Status ; Successful - Thu Jun 17 22:33:41 2004 ;
; Revision Name ; control_data ;
; Top-level Entity Name ; control_data ;
; Family ; APEX II ;
; Device ; EP2A15B724C7 ;
; Total logic elements ; 44 / 16,640 (< 1 %) ;
; Total pins ; 28 / 492 (5 %) ;
; Total memory bits ; 0 / 425,984 (0 %) ;
; Total PLLs ; 0 / 4 (0 %) ;
+---------------------+------------------------------------+

3.

+--+
; Analysis & Synthesis Summary ;
+---------------------------+--------------------------------------+
; Analysis & Synthesis Status ; Successful - Thu Jun 17 22:34:38 2004 ;
; Revision Name ; control_data ;
; Top-level Entity Name ; control_data ;
; Family ; MAX II ;
; Total logic elements ; 42 ;
; Total pins ; 28 ;
; UFM blocks ; 0 ;
+---------------------------+--------------------------------------+

+--+
; Fitter Summary ;
+---------------------+------------------------------------+
; Fitter Status ; Successful - Thu Jun 17 22:34:42 2004 ;
; Revision Name ; control_data ;

; Top-level Entity Name ; control_data ;
; Family ; MAX II ;
; Device ; EPM240T100C3 ;
; Total logic elements ; 42 / 240 (17 %) ;
; Total pins ; 28 / 80 (35 %) ;
; UFM blocks ; 0 / 1 (0 %) ;
+----------+----------+

4.

+----------+----------+
; Analysis & Synthesis Summary ;
+----------+----------+
; Analysis & Synthesis Status ; Successful - Thu Jun 17 22:36:44 2004 ;
; Revision Name ; control_data ;
; Top-level Entity Name ; control_data ;
; Family ; MAX3000A ;
; Total macrocells ; 80 ;
; Total pins ; 28 ;
+----------+----------+

+----------+----------+
; Fitter Summary ;
+----------+----------+
; Fitter Status ; Successful - Thu Jun 17 22:36:49 2004 ;
; Revision Name ; control_data ;
; Top-level Entity Name ; control_data ;
; Family ; MAX3000A ;
; Device ; EPM3128ATC100-5 ;
; Total macrocells ; 80 / 128 (62 %) ;
; Total pins ; 32 / 80 (40 %) ;
+----------+----------+

5.

+----------+----------+
; Analysis & Synthesis Summary ;
+----------+----------+
; Analysis & Synthesis Status ; Successful - Thu Jun 17 22:53:22 2004 ;
; Revision Name ; control_data ;
; Top-level Entity Name ; control_data ;
; Family ; MAX7000S ;
; Total macrocells ; 80 ;
; Total pins ; 28 ;
+----------+----------+
+----------+----------+
; Fitter Summary ;
+----------+----------+
; Fitter Status ; Successful - Thu Jun 17 22:53:26 2004 ;
; Revision Name ; control_data ;
; Top-level Entity Name ; control_data ;

; Family ; MAX7000S ;
; Device ; EPM7128SLC84-6 ;
; Total macrocells ; 80 / 128 (62 %) ;
; Total pins ; 32 / 68 (47 %) ;
+----------+----------+

As seen from the results, the circuit uses about 30 pins and about 40 logic elements in case of first three devices., and 80 macrocells and about 30 pins in case of the MAX families. Very moderate FPGA IC's can be used to implement this circuit.

Development Environment: Quartus II

References:

Z. Navabi, *VHDL - Analysis and Modeling of Digital Systems*, (New York: McGraw-Hill, Inc, 1993)

C. H. Roth, *Digital System Design Using VHDL*, (Boston: PWS Publishing Company, 1997)