Training Task: Morse Beacon for Navigation

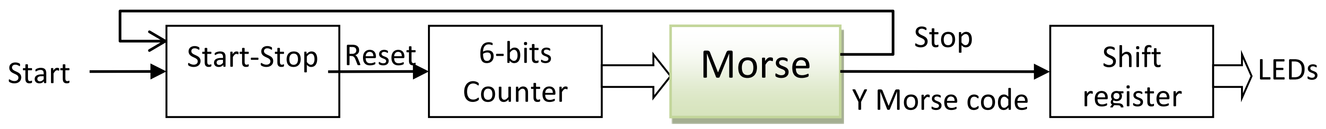
Document version 2025, Sept 05

Table of Contents

List of Figures

Assignment

Design a combinational circuit that encodes the output of a 6-bit binary counter into the Morse sequence assigned to you. The sequence consists of zeros and ones and appears at the Morse circuit Y output. It was generated permutations from your name so that it is unique for all exercises and has approximately the same minimization complexity for all students, as far as possible.



Note: You will design only the ***Morse****, the other circuits are in DCE library.*

Figure 1 — Principal Diagram of the Morse Beacon

The requirements

Find logical equations and describe them in VHDL code. Verify them first using the GHDL simulator and then in demo circuits.

Because the task is training, several constraints are given for the VHDL code of the Morse part.

- In Morse, you can use any number of NOT, AND, NAND, OR, NOR, and XOR operators, symbols for inputs (input), outputs (output), logic 1 (VCC), and 0 (GND).
- However, no more than **one four-input multiplexer** is allowed in Morse. The advantages of multiplexers will be the topic of the lectures.

Notes: We will create the demo circuit during practical exercises together. Then, you only replace MorseMHL2 with your MorseXXX. There is also a *.sof Quartus programmer file that you can load into the Veek-MT2 board for inspiration on the result.

Tutorial: Morse Circuit for Beacon EA

The EA beacon, explained in the second lecture, transmits the sequence 010001011100.

Input **X**	0	1	2	3	4	5	6	7	8	9	10	11
Morse-output **STOP**	0	0	0	0	0	0	0	0	0	0	0	**1**
Y output of MorseEA	0	1	0	0	0	1	0	1	1	1	0	0

The STOP output will be in logic '1' only at the time of the last bit of the sequence and possibly in the subsequent bits as well, but we don't care about those because those outputs don't apply.

Logical equations were proposed in the lecture dedicated to minimizing logical functions:

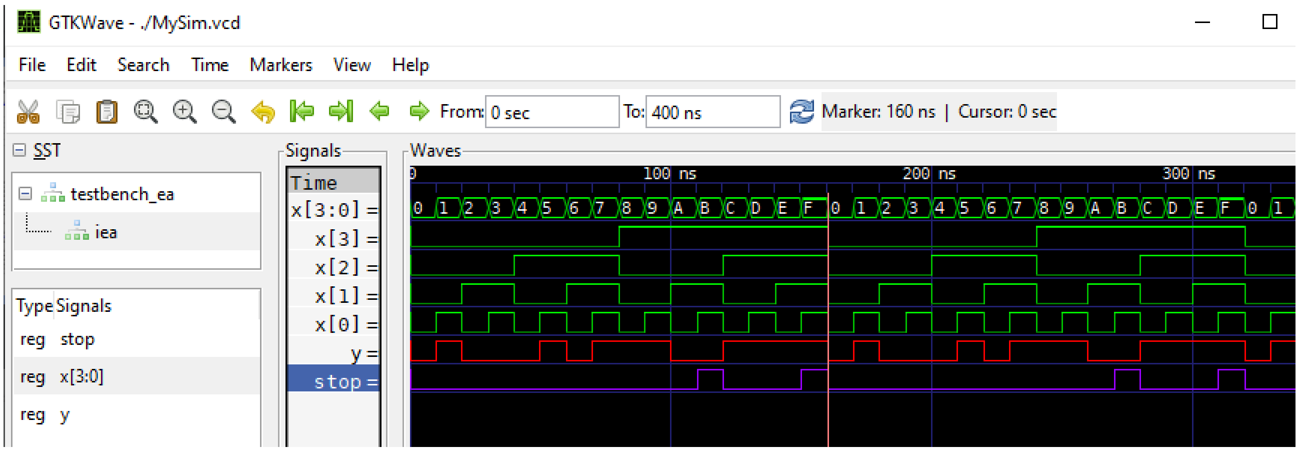
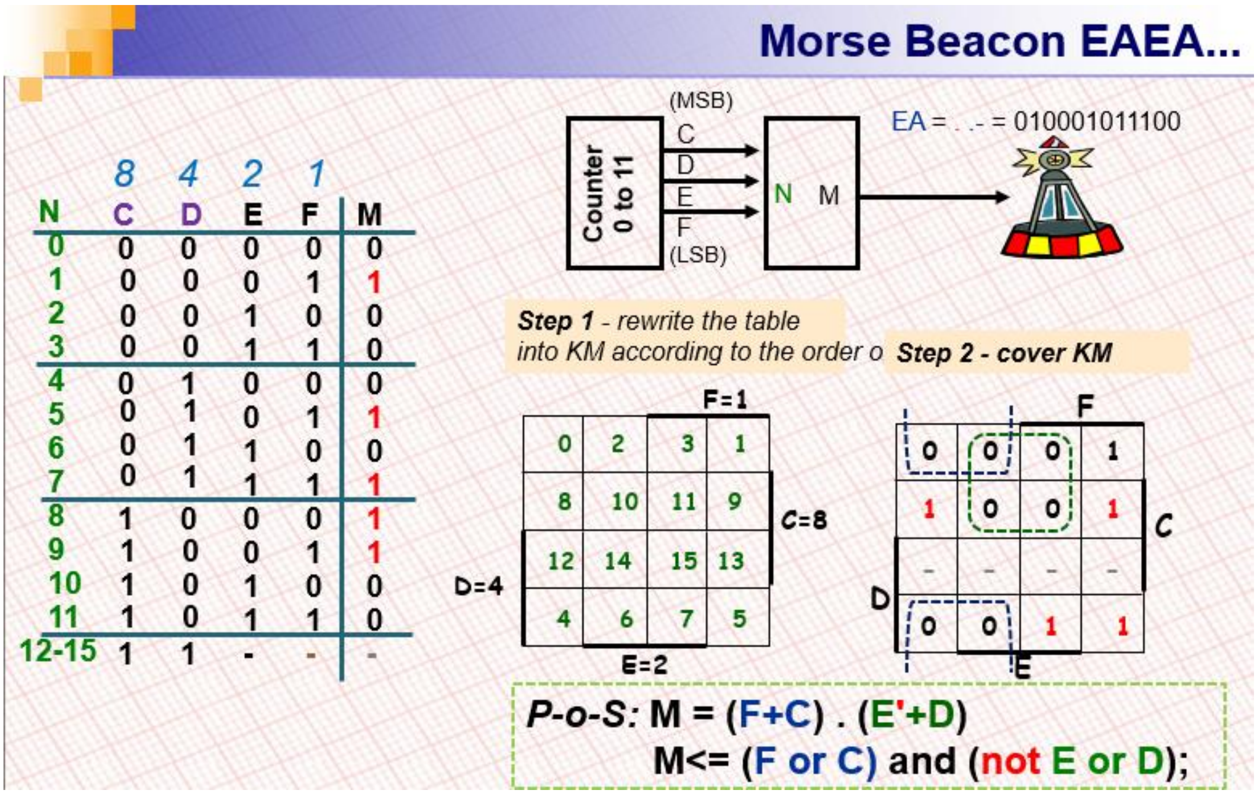


Figure 2 — MorseEA - The input and outputs MorseEA (GHDL simulator)



Morse Beacon EAEA...

	8	4	2	1	
N	C	D	E	F	M
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12-15	1	1	-	-	-

Step 1 - rewrite the table into KM according to the order o

0	2	3	1
8	10	11	9
12	14	15	13
4	6	7	5

Step 2 - cover KM

0	0	0	1
1	0	0	1
-	-	-	-
0	0	1	1

P-o-S: M = (F+C) . (E'+D)

M<= (F or C) and (not E or D);

Figure 3 — EA Beacon from the 2nd lecture

The vector **X** is our input, so the signals F, E, D, and C (labeled identically to the later solution of MHL2 in this paper) associate the lowest bit of X(0) with f, X(1) with e, X(2) with d, and X(3) with the highest bit c. We define X as an unsigned type to simplify GHDL simulations.

To signal the end of Morse, we add the **STOP** output in the state $11_{10} = 2^3+2^1+2^0 \rightarrow 1011_2$. We join these bits in '1' by the AND operation, because such a combination will be the first in an ascending series of binary numbers starting at 0000_2.

We call the circuit **MorseEA** and write its equations directly in VHDL.

```
library ieee;use ieee.std_logic_1164.all;use ieee.numeric_std.all;
entity MorseEA is
    port (X    : in unsigned(3 downto 0):=(others=> '0'); --CDEF
          Y, STOP : out std_logic:='0');
end entity;
architecture behavioral of MorseEA is
signal c,d,e,f : std_logic:='0';
begin
    f<=X(0); e<=X(1); d<=X(2); c<=X(3);
    Y <= (f or c) and (not e or d);
    STOP <= c and e and f;
end architecture;
```

Remember that **parentheses are necessary in VHDL.** Only the unary NOT operation has higher precedence. In C, the AND (&&) operator has taken precedence over the OR (||) operator, but not in VHDL and Boolean algebra. Logical operators have no precedence over each other.

We recommend using the free VSC (Visual Studio Code) environment as an editor and the GHDL simulator. Its installation is described in detail at the end of the GHDL installation guide on the page: https://dcenet.fel.cvut.cz/edu/fpga/install_en.aspx . There is also the commented simulation of Morse.

On the VEEK-MT2 development board, MorseEA can be tested simultaneously in the following circuit:

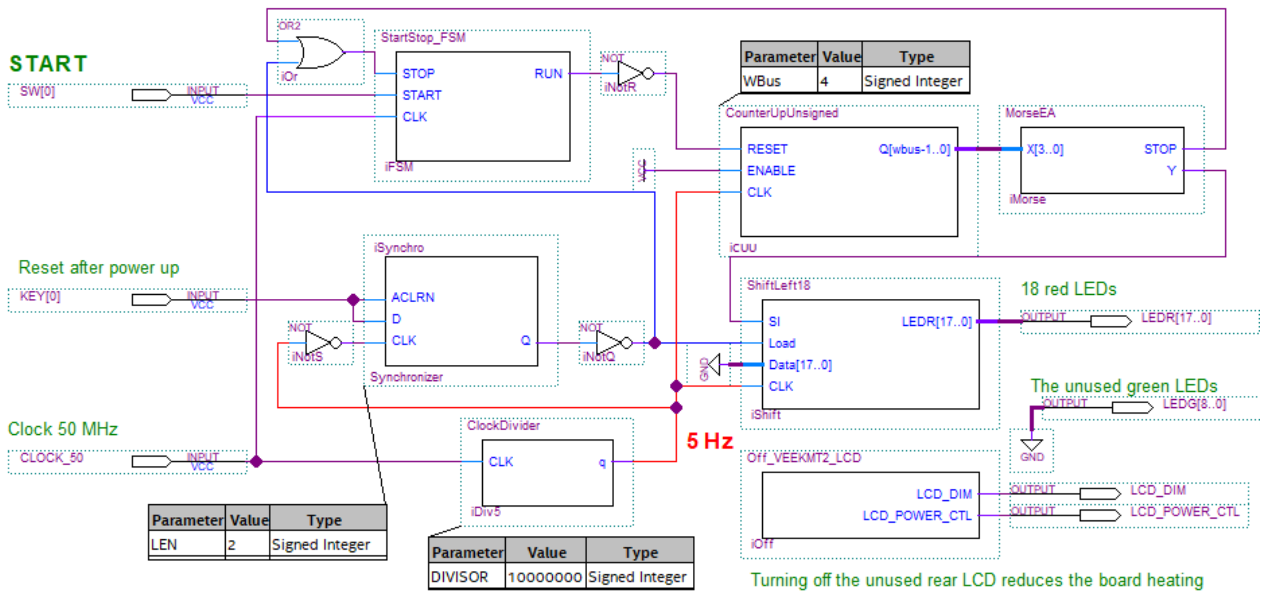


Figure 4 — Testing MorseEA

The START-STOP circuit is a finite automaton and needs a faster clock, so it is connected directly to the 50 MHz clock generator of the Veek-MT2 board.

Schematic Symbol of MorseEA

We create it in Quartus in the Project Navigator window, on its Files tab, where we right-click the file name in the list and select Create Symbol Files for Current File.

Note: This context menu works only for VHDL files. Schematics (.bdf - block diagram files) require a longer way, which also applies to VHDL.*

The file must first be opened. Then, we select from the main Qurtus menu -> File. In its submenu, we choose Create Symbol Files for Current File.

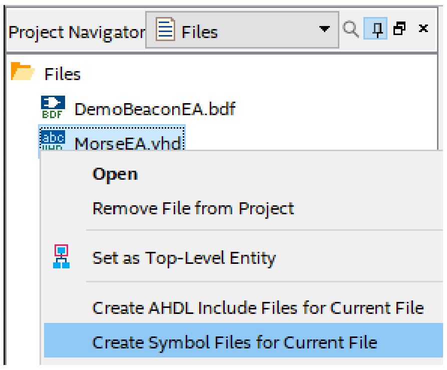
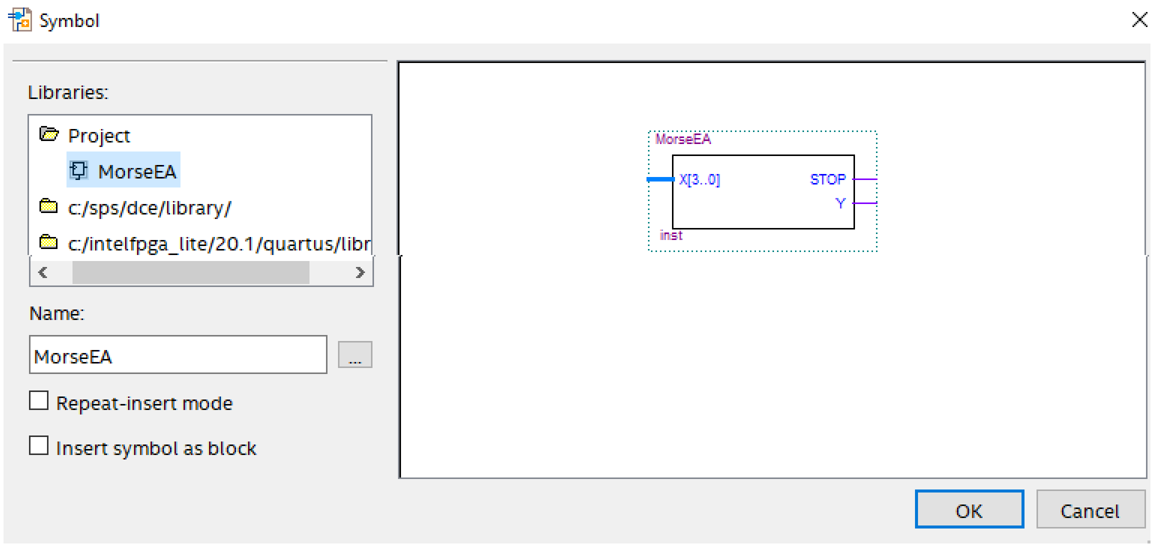


Figure 5 - Creating Schematic Symbol

The created symbol will appear in the symbolic editor in the project files:



Name: MorseEA

☐ Repeat-insert mode

☐ Insert symbol as block

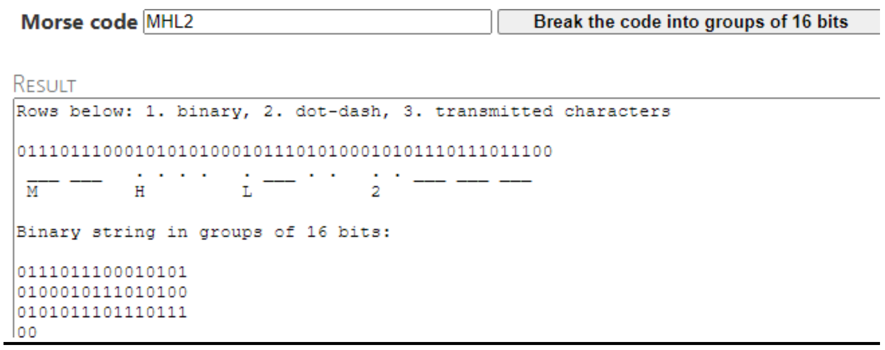
Figure 6 – Our Symbol in the Schematic Editor

Note: The difference between the indexes in VHDL and the schema

- The schematic editor relies on the syntax of the internal HDL language AHDL (Altera Hardware Description Language), see AHDL or the AHDL wiki for more details, which is designed to be as short as possible to squeeze the descriptions into schemas. Today, it is only used by Quartus for schemas and prototypes of mega-functions, such as ROMs and pin assignments.
- In a VHDL entity, the output **LEDR: out std_logic_vector(17 downto 0);** would be in the diagram abbreviated to **LEDR[17..0]**. For example, VHDL writes the index **LEDR(17)**, but the schema has it as **LEDR[17]** in AHDL.
- By analogy, input KEY: in std_logic_vector(3 downto 0); appears in the schema as KEY[3..0] in AHDL, and KEY[0] in the schema will be KEY(0) in VHDL.
- **ATTENTION!** The vector outputs SW, LEDR, LEDG, and KEY used in the schema must form one **continuous slice of their sequence**, i.e., similarly to their PIN assignment definitions.
- It is possible to connect only KEY[2] and KEY[1], but you cannot use only KEY[0] and KEY[2], i.e., omit KEY[1]. Alternatively, KEY[1] must be inserted, perhaps unplugged, but it just exists in the diagram to keep a continuous vector slice.

The editor of block diagram schematics is a bit of an archaism. It was left in Quartus only at the request of the world's universities. Its schemas are not compiled directly. They are converted to some HDL language that has yet to be translated. And in VHDL, we always have to use a continuous part of the vector.

Tutorial: MHL2 Morse Sequence



Morse code MHL2 — Break the code into groups of 16 bits

RESULT

```
Rows below: 1. binary, 2. dot-dash, 3. transmitted characters

01110111000101010100010111010100010101110111011100
 ___ ___   . . . .   . ___ . .   . . ___ ___ ___
 M         H         L           2

Binary string in groups of 16 bits:

0111011100010101
0100010111010100
0101011101110111
00
```

Figure 7 – Online Converter MHL2

Morse timing complies with the International Maritime Standard:

- *short mark, dot or "dit": "dot duration" is one-time unit long;*
- *longer mark, dash or "dah": three time units long;*
- *inter-element gap between the dots and dashes within a character: one dot duration or one unit long;*
- *short gap (between letters): three-time units long.*

Why does the sequence begin with '0' and end with a pair of "00"?

In the beginning, the counter equals 0, and Morse Y is 0; thus, there is no signal. However, when the beacon runs continuously (START is permanently on), a 3-time pause is inserted between repetitions because the binary sequence starts at 0 and ends at 00. So the last character is separated from the first by the correct gap between characters three times units long, i.e., 000.

How to convert another TEXT to Morse?

LSP site contains the converter of a text to Morse: https://dcenet.fel.cvut.cz/edu/fpga/Morse_en.aspx

Logic function design

To shorten the equations, we introduce temporary variable labels to avoid typos, which are frequent sources of errors. We will write

Input of MorseMHL2:	X(0)	X(1)	X(2)	X(3)	X(4)	X(5)
Our associations:	f	e	d	c	b	a

Note: Of course, you can find many online minimizers on the web; some of them are even in the form of an Excel spreadsheet, into which you can paste the values, and behold, you already have the result.

However, the final exam will include Karnaugh maps (KM), which are easier to check than expressions. ☺ Because of this, the KMs are still used by professional designers in this day and age of computing. After all, they will provide a better overview of the function. So make Karnaugh maps of your task by own hand to learn how to use them.

For practice reasons, we first rewrite the given binary sequence into a truth table, see the next page.

Carefully double-check all Morse code to KM conversions.
Past semesters have shown that errors are the most common here!

Status	a	b	c	d	e	f	Y (output Beacon)	STOP output
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	0
2	0	0	0	0	1	0	1	0
3	0	0	0	0	1	1	1	0
4	0	0	0	1	0	0	0	0
5	0	0	0	1	0	1	1	0
6	0	0	0	1	1	0	1	0
7	0	0	0	1	1	1	1	0
8	0	0	1	0	0	0	0	0
9	0	0	1	0	0	1	0	0
10	0	0	1	0	1	0	0	0
11	0	0	1	0	1	1	1	0
12	0	0	1	1	0	0	0	0
13	0	0	1	1	0	1	1	0
14	0	0	1	1	1	0	0	0
15	0	0	1	1	1	1	1	0
16	0	1	0	0	0	0	0	0
17	0	1	0	0	0	1	1	0
18	0	1	0	0	1	0	0	0
19	0	1	0	0	1	1	0	0
20	0	1	0	1	0	0	0	0
21	0	1	0	1	0	1	1	0
22	0	1	0	1	1	0	0	0
23	0	1	0	1	1	1	1	0
24	0	1	1	0	0	0	1	0
25	0	1	1	0	0	1	1	0
26	0	1	1	0	1	0	0	0
27	0	1	1	0	1	1	1	0
28	0	1	1	1	0	0	0	0
29	0	1	1	1	0	1	1	0
30	0	1	1	1	1	0	0	0
31	0	1	1	1	1	1	0	0
32	1	0	0	0	0	0	0	0
33	1	0	0	0	0	1	1	0
34	1	0	0	0	1	0	0	0
35	1	0	0	0	1	1	1	0
36	1	0	0	1	0	0	0	0
37	1	0	0	1	0	1	1	0
38	1	0	0	1	1	0	1	0
39	1	0	0	1	1	1	1	0
40	1	0	1	0	0	0	0	0
41	1	0	1	0	0	1	1	0
42	1	0	1	0	1	0	1	0
43	1	0	1	0	1	1	1	0
44	1	0	1	1	0	0	0	0
45	1	0	1	1	0	1	1	0
46	1	0	1	1	1	0	1	0
47	1	0	1	1	1	1	1	0
48	1	1	0	0	0	0	0	0
49	1	1	0	0	0	1	0	1
50	1	1	0	0	1	0	X	X
51	1	1	0	0	1	1	X	X
52	1	1	0	1	0	0	X	X
53	1	1	0	1	0	1	X	X
54	1	1	0	1	1	0	X	X
55	1	1	0	1	1	1	X	X
56	1	1	1	0	0	0	X	X
57	1	1	1	0	0	1	X	X
58	1	1	1	0	1	0	X	X
59	1	1	1	0	1	1	X	X
60	1	1	1	1	0	0	X	X
61	1	1	1	1	0	1	X	X
62	1	1	1	1	1	0	X	X
63	1	1	1	1	1	1	X	X

Minimizing a logic function with a Karnaugh map would be difficult for such a large truth table. We prefer Shannon expansion for the Y' output (see lectures or our textbook Logic Circuits on FPGAs, page 49. Again, we create the STOP output by applying knowledge of circuit behavior.

<u>We'll start with Shannon's expansion of the Beacon output</u>:

We split the truth table into four smaller truth tables ("Shannon cofactors") of 4 variables each, with the other two variables having a constant value. They will be "splitting variables", see lecture 2:

a = 0, b = 0

Status	c	d	e	f	Y (Morse)
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

a = 0, b = 1

Status	c	d	e	f	Y (Morse)
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

a = 1, b = 0

Status	c	d	e	f	Y (Morse)
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

a = 1, b = 1

Status	c	d	e	f	Y (Morse)
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	X
3	0	0	1	1	X
4	0	1	0	0	X
5	0	1	0	1	X
6	0	1	1	0	X
7	0	1	1	1	X
8	1	0	0	0	X
9	1	0	0	1	X
10	1	0	1	0	X
11	1	0	1	1	X
12	1	1	0	0	X
13	1	1	0	1	X
14	1	1	1	0	X
15	1	1	1	1	X

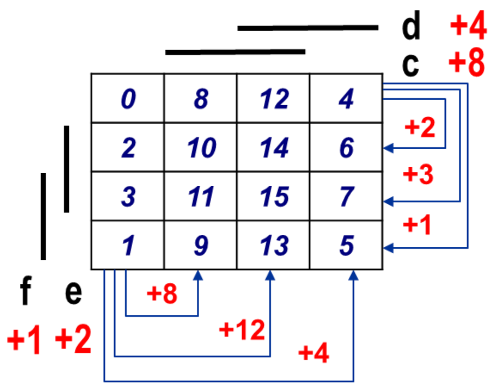
We now solve the minimization of logic functions using simple Karnaugh maps with four variables for the output Y (Morse output of Morse code), which we denote by Y(a,b) **= f(c,d,e,f)**, where symbols **a**, **b** stand for logical values of the inputs a, b of the given map.

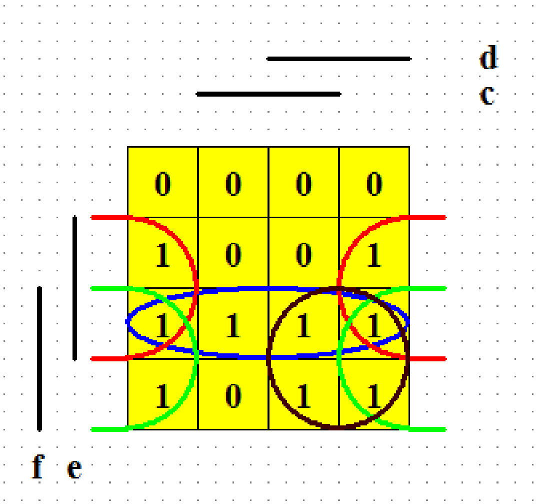
In Karnaugh maps, the values do not go in sequence, but in the order of the bit weights, see

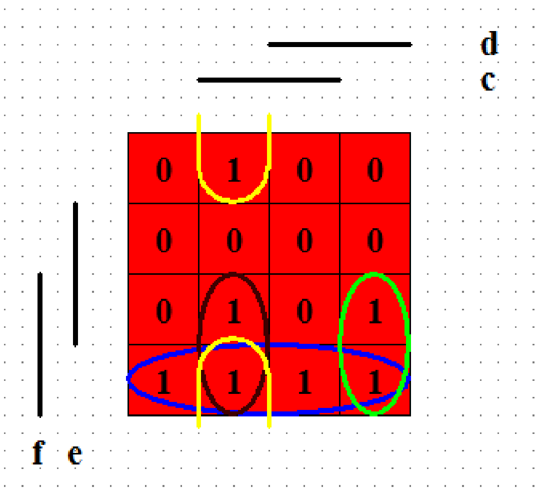
Logic Circuits on FPGAs, page 34 onwards.

We recommend first clarifying the order by writing down serial numbers in cells. The numbering depends on your arrangement of inputs f, e, d, and c. If you change their arrangement, the numbering will also change. In the exams, it can also be different; so learn how to build it for any arrangement of inputs in the Karnaugh map..

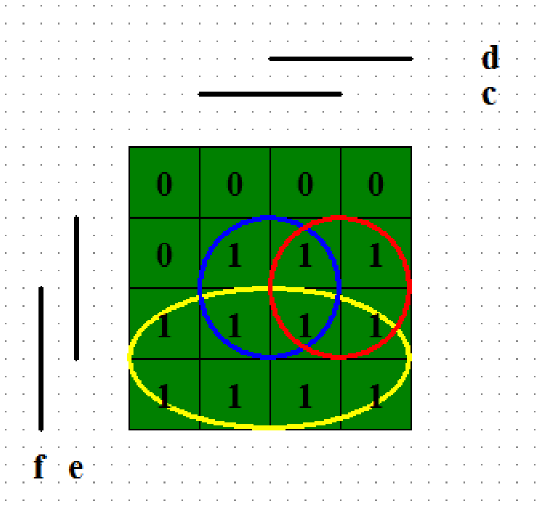
The following maps have the order corresponding to cdef weights 8421. Their sum refers to the KM line number.

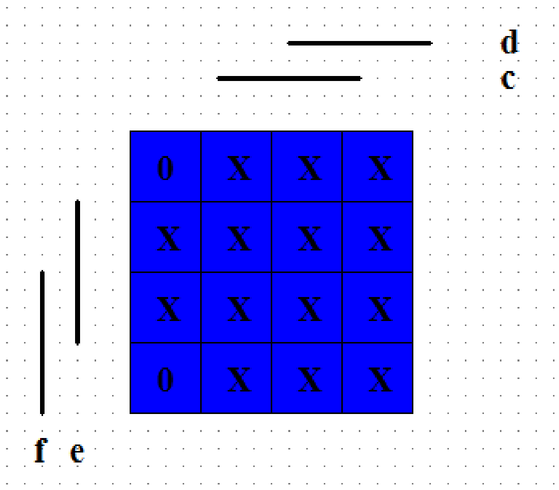






$$Y00 = ef + \bar{c}e + \bar{c}f + df \quad Y01 = \bar{e}f + c\bar{d}f + \bar{c}df + cd\bar{e}$$





$Y10 = ce + de + f$ We don't have to solve the Y11 = 0 contradiction

Note: The letter codes assigned to you can always be decomposed into four Karnaugh 4x4 maps regardless of their Morse code length. No one got a sequence longer than 64 bits.

For output STOP, we utilize knowledge of the circuit behavior. We know that our counter counts from zero to STOP state, and then it resets. So its sequence always ends in the number 49, which has the binary code:

$$49_{10} = 110001_2 = 2^5 + 2^4 + 1$$

The number 49 will be the first number in the number series 0 to 49 to have 1's in the 5th, 4th and 0th bits. No other number can appear before it because it would be greater than 49, contrary to the nature of an arithmetic number series. Thus, the equation is sufficient for the KOM output of STOP:

$$STOP = abf$$

Validating MHL2 in the Quartus development environment

First, we check the equations in VHDL to ensure we have correctly designed the logic functions.
Notes: 1/ The comments /* */ are only known in VHDL2008, where we are writing.
2/ Use only ASCII characters in VHDL, Quartus doesn't know special characters!

```
library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all;
entity MorseMHL2 is
port ( X : in unsigned(5 downto 0):=(others=> '0'); -- I/O initializations are only relevant for simulations.
        Y, STOP : out std_logic:='0' );
end entity;
architecture behavioral of MorseMHL2 is
signal a, b, c, d, e, f : std_logic:='0';
signal Y00, Y01, Y10, Y11 : std_logic:='0';
begin
-- renaming of signals
a<= X(5); b<= X(4); c<= X(3); d<= X(2); e<= X(1); f<= X(0);
Y00 <= (e and f) or (not c and e) or (not c and f) or (d and f);
Y01 <= (not e and f) or (c and not d and f) or (not c and d and f) or (c and not d and not e);
Y10 <= (c and e) or (d and e) or f;
Y11 <= '0';
-- Multiplexer ----------------------------
 with X(5 downto 4) select
 Y <= Y00 when "00", Y01 when "01", Y10 when "10",
       Y11 when others; -- 2 std_logic bits can have 9*9 values!!!
 STOP<=  a and b and f; -- 2**5 + 2**4 + 1 = 49
end architecture;
```

We write a testbench for the simulation, which will be explained at the lecture. It represents the part that acts as a generator of simulation inputs, in this case, the values of the vector x from 0 to 63

```
library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all;
library work;
entity testbench_MHL2 is
end entity;

architecture rtl of testbench_MHL2 is
signal x: unsigned(5 downto 0):=(others=> '0');
signal y, stop : std_logic:='0';
begin
  imhl2 : entity work.MorseMHL2 port map (x, y, stop);
          x<= x+1 after 10 ns;
end architecture;
```

We will test the design in the GHDL simulator.

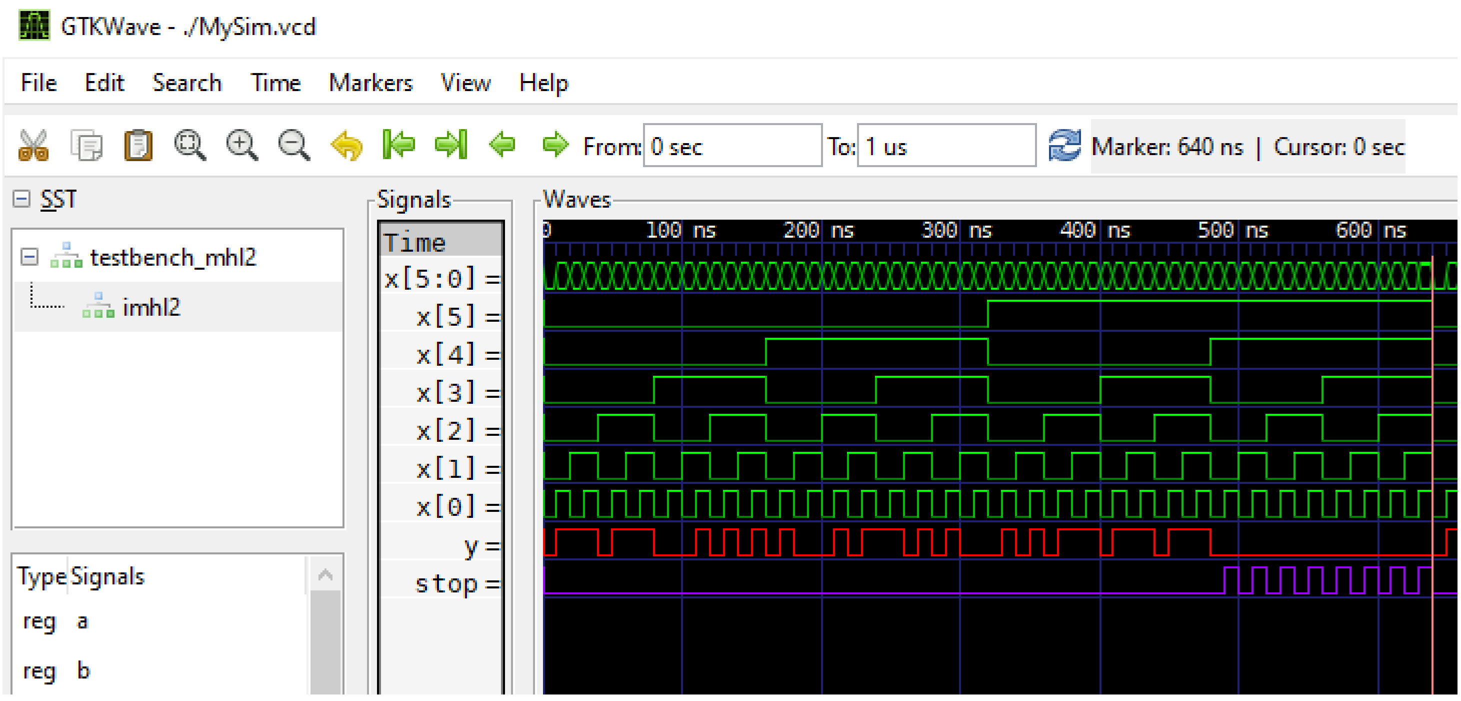


Figure1 - MorseMHL2 simulator output

We can make the scheme even more user-friendly. We add a buzzer to make the Morse code beep, and we can still display the counter's instantaneous reading. It's also helpful to display static text on the LCD with the transmitted code; see the diagram on the next page. We also turn off the unused green LEDs, LEDG[8..0], so they do not glow dimly.

The frequent mistake:

If your circuit starts to encrypt, it may be due to incorrectly connected bits of the counter output to the Morse circuit input, with their order mixed up. Remember that the output of counter Q0 is the lowest bit, so it must lead to X0 (originally f), while Q4 connects to the lower bit of the multiplexer and Q5 to the higher bit. Recheck!

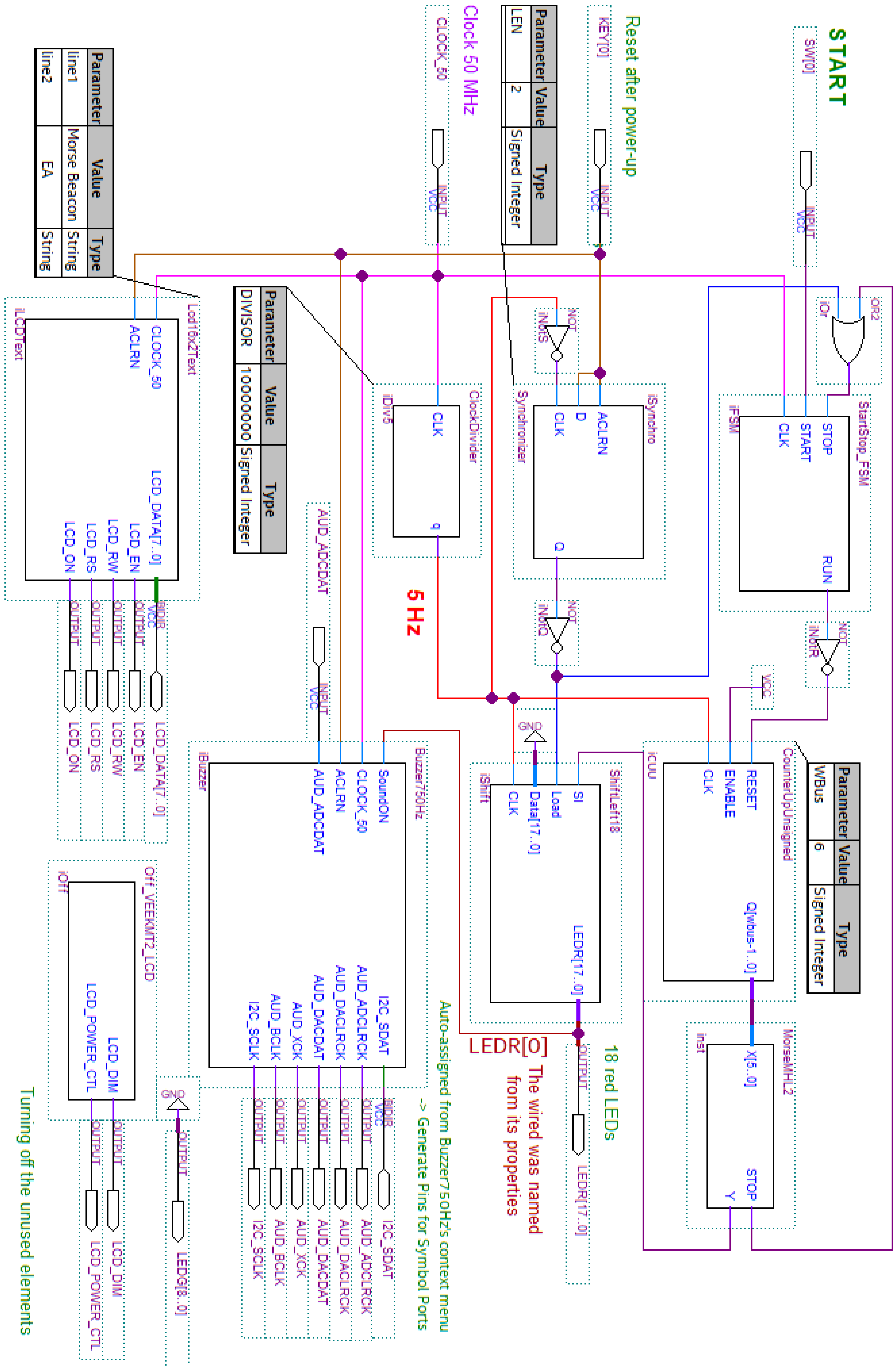


Figure 8 — Extended MHL2 Test Scheme

Premium task- Expansion + 3 points

Rewrite the Morse equations into a block diagram, see example MHL2 below, to practice the relation between schema and equations. It can be drawn very quickly. The schema below took less than half an hour. Use a Multiplexer4x1 from the DCE/library library for the output.

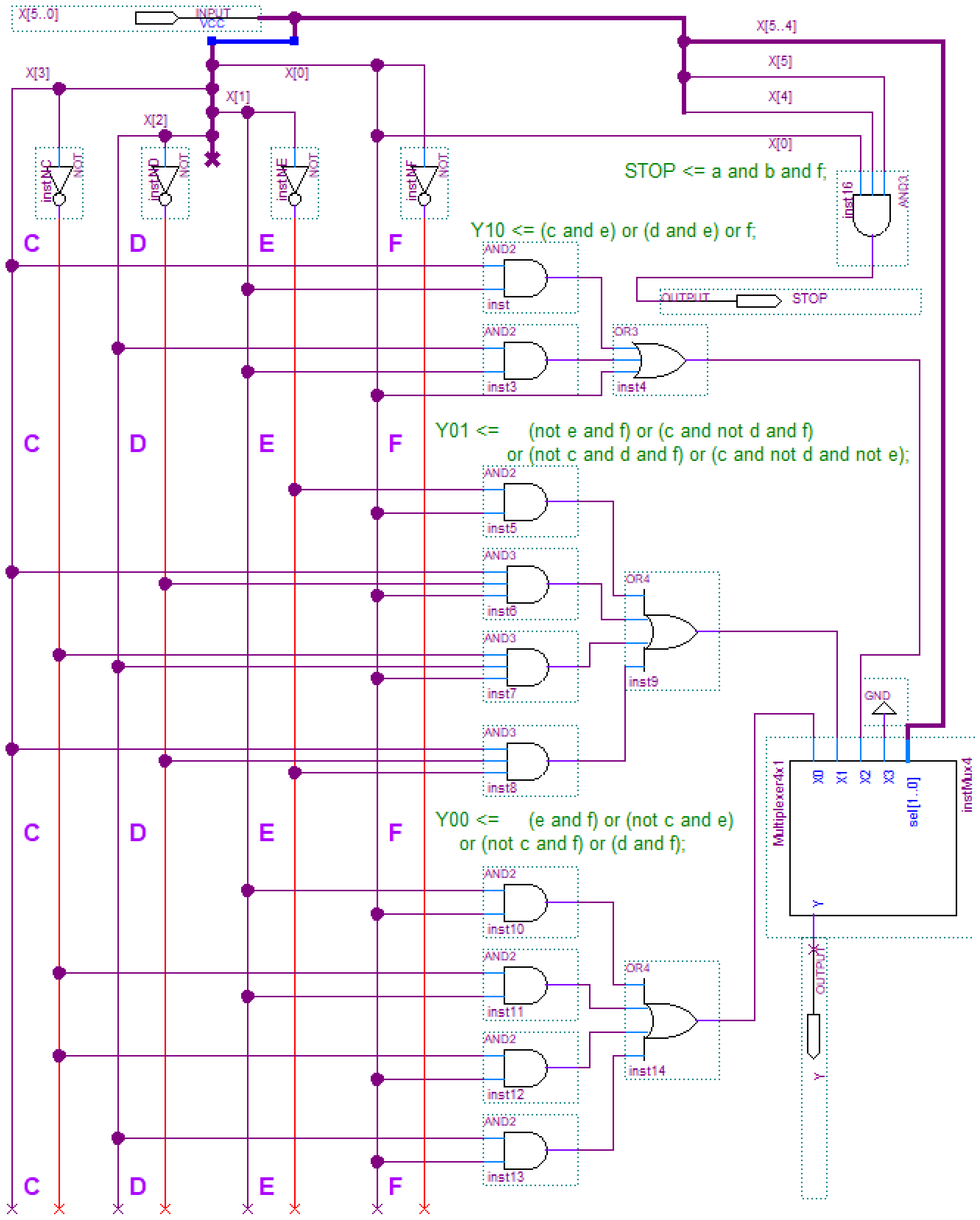


Figure 9 — Morse BDF Diagram of MHL2

To make your work easier, you can use a zipped prototype on the task website, in which the bus signals are already named. In the graphic editor, a bus branch is done by perpendicularly connecting the wire to the bus and correctly naming it via the Properties context menu.

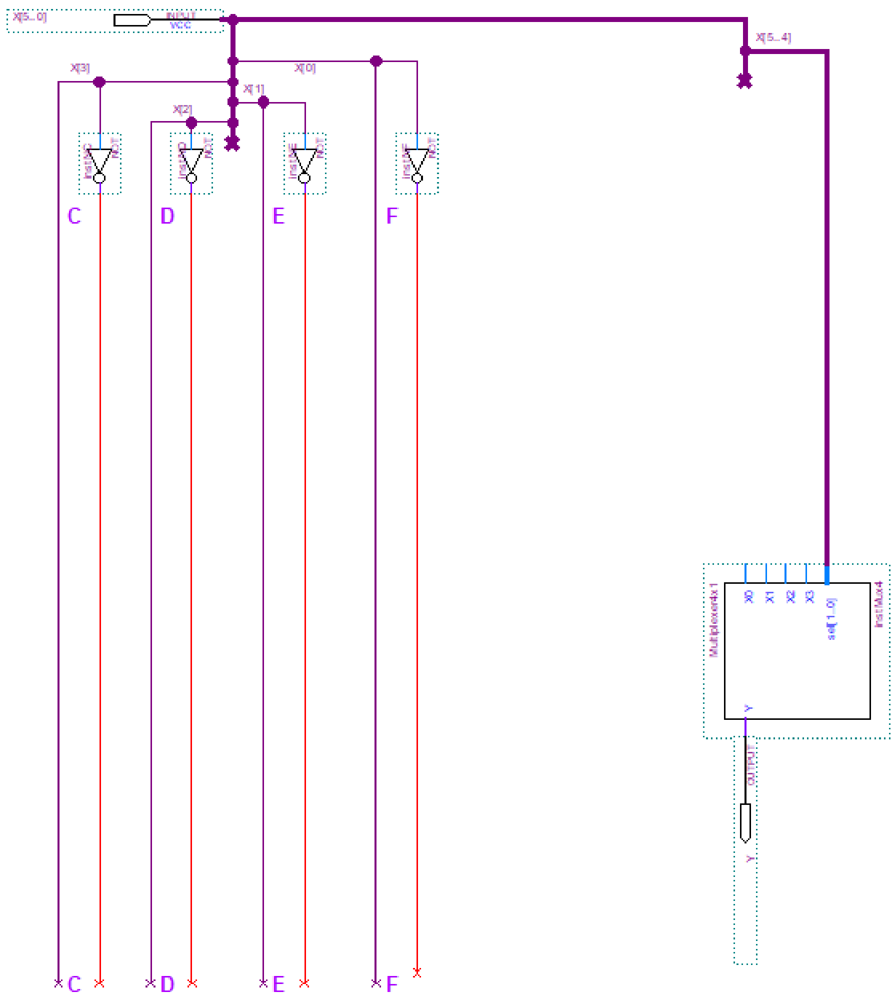
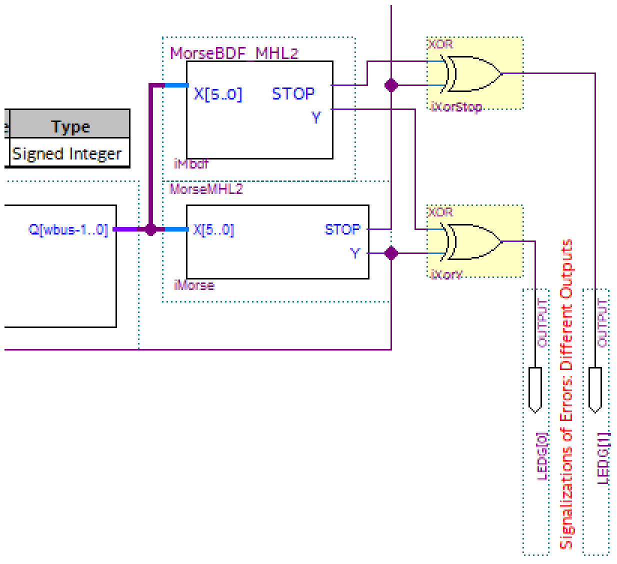


Figure 10 — Prototype



The resulting circuit, MorseBDF_MHL2.bdf, must be tested in the same scheme as MorseMHL2.vhd.

Insert both in parallel and compare their outputs with XOR gates. Errors, i.e., different outputs, show '1' as an error.

Figure 11 — Test of Both MHL2

Warning: Square gates are not proper gates!

Don't use (N)AND 5, 7, and 9 inputs, OR with 5 and other gates with schematic markers drawn as **square blocks**, but compose them from adder gates. **Square gates are not proper gates!** They belong to the MAX2+ library, which was dropped from Quartus version 20 and above and not replaced by anything, since the symbolic editor was left over from the old days just for the sake of universities and is no longer maintained.

The note above refers exclusively to square gates. Multiplexers and other circuits usually have square schematic markers.

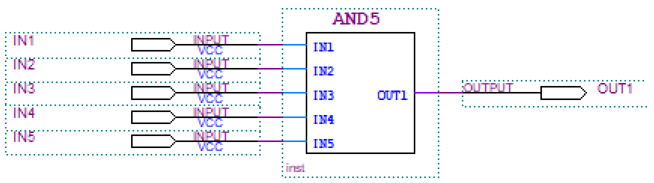
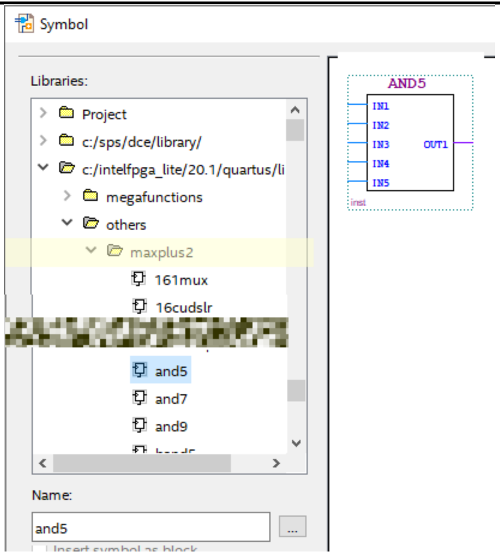
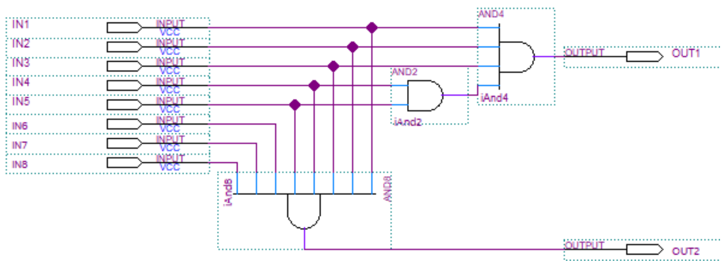
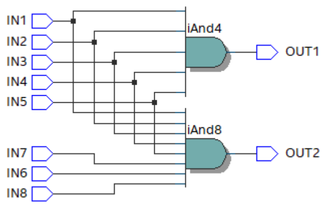
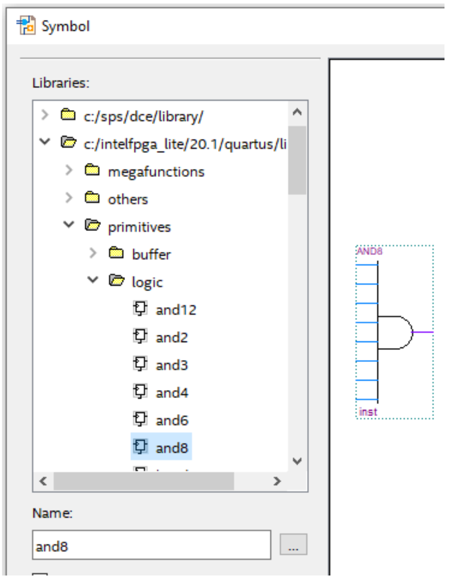
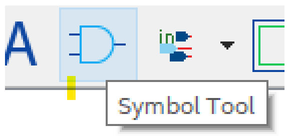
Square gate belonging to the ancient MAX2+	others→Max2plus
 AND5 IN1 IN2 IN3 IN4 IN5 INPUT VCC OUT1 OUTPUT inst *Error (12004): Port "OUT1" does not exist in primitive "AND5" of instance "inst"*	 Symbol Libraries: Project c:/sps/dce/library/ c:/intelfpga_lite/20.1/quartus/li megafunctions others maxplus2 161mux 16cudslr and5 and7 and9 Name: and5 AND5 IN1 IN2 IN3 IN4 IN5 OUT1 inst
Only the rounded gate works as we need it to!	**primitives→logic**
 IN1 IN2 IN3 IN4 IN5 IN6 IN7 IN8 INPUT VCC AND4 iAnd4 AND2 iAnd2 AND8 iAnd8 OUTPUT OUT1 OUT2 Info (293000): ...compilation was successful. 0 errors. <u>Quartus got involved (its RTLViewer)</u>  IN1 IN2 IN3 IN4 IN5 IN7 IN6 IN8 iAnd4 iAnd8 OUT1 OUT2	 Symbol Libraries: c:/sps/dce/library/ c:/intelfpga_lite/20.1/quartus/li megafunctions others primitives buffer logic and12 and2 and3 and4 and6 and8 Name: and8 AND8 inst

Figure 12 — Wrong square gate and correct rounded gate



The Symbol tool in the schematic editor can offer you something from MAX2+, but you don't have to accept everything someone slips you. ☺

~ T h e   e n d ~