# Lösningsförslag övningsuppgifter 2025-03-28

**1.** På mikrodatorn ATmega328P används en multiplexer för att analoga kanaler PORTC0 – PORTC7 ska kunna dela på en enda AD-omvandlare. Enbart en av de analoga kanalernas insignaler släpps igenom till AD-omvandlaren vid ett givet tillfälle, vilket kontrolleras via selektorbitar MUX[2:0] i registret ADMUX *(ADC Multiplexer Select Register)* enligt nedanstående tabell:

| MUX[2:0] | Kanal |
|---|---|
| 000 | PORTC0 |
| 001 | PORTC1 |
| 010 | PORTC2 |
| 011 | PORTC3 |
| 100 | PORTC4 |
| 101 | PORTC5 |
| 110 | PORTC6 |
| 111 | PORTC7 |

*Tabell 1: Sanningstabell för multiplexer för ATMega328:s analoga kanaler.*

Vi ska i denna uppgift konstruera en sådan 8-to-1 multiplexer (8-to-1 indikerar åtta inportar samt en utport). För läsbarhetens skull sätter vi att A - H = PORTC[7:0] samt att S[2:0] = MUX[2:0] i resten av uppgiften. Vi kan också beteckna multiplexerns utsignal till X. Sanningstabellen ovan kan därmed skrivas om såsom visas nedan:

| S[2:0] | X |
|---|---|
| 000 | A |
| 001 | B |
| 010 | C |
| 011 | D |
| 100 | E |
| 101 | F |
| 110 | G |
| 111 | H |

*Tabell 2: Sanningstabell för uppgift 1.*

a) Härled en logisk ekvation för multiplexerns utsignal X via insignaler A - F samt selektorbitar S[2:0].

b) Realisera grindnätet i CircuitVerse. Kontrollera att det fungerar korrekt.

a) Implementera konstruktionen i VHDL via en modul döpt *mux_8_to_1.* Välj FPGA-kort Terasic DE0 (enhet 5CEBA4F23C7). Toppmodulen ska ha samma namn som projektet.

b) Verifiera konstruktionen på ett FPGA-kort. Anslut insignaler A - H till var sin slide-switch, selektorbitar S[2:0] till var sin tryckknapp samt utsignal X till en lysdiod, se databladet för PIN-nummer (finns här).

**OBS!** Tryckknapparna har aktivt låg insignal, så insignalen är låg vid nedtyckning. Vi kan enkelt lösa detta genom att använda inverterande signaler i toppmodulen.

**OBS! Vänd blad!**

## Lösning

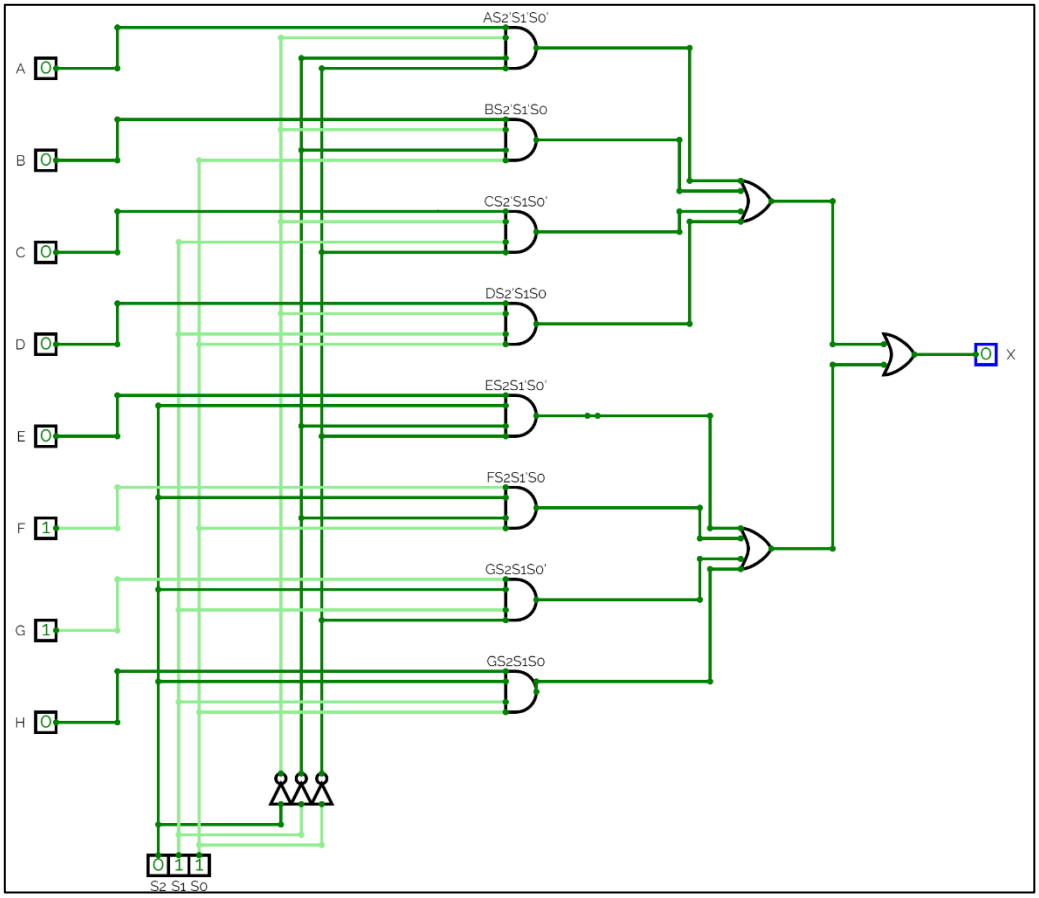
Utsignal X ska bli lika med:

- A när S[2:0] = 000
- B när S[2:0] = 001
- C när S[2:0] = 010
- D när S[2:0] = 011
- E när S[2:0] = 100
- F när S[2:0] = 101
- G när S[2:0] = 110
- H när S[2:0] = 111

Vi kan härleda detta till en ekvation, där S[2:0] = 000 skrivs $S_2'S_1'S_0'$, S[2:0] = 001 skrivs $S_2'S_1'S_0$ och så vidare:

$$X = AS_2'S_1'S_0' + BS_2'S_1'S_0 + CS_2'S_1S_0' + DS_2'S_1S_0 + ES_2S_1'S_0' + FS_2S_1'S_0 + GS_2S_1S_0' + HS_2S_1S_0$$

Vi kan realisera motsvarande grindnät såsom visas nedan:




*Figur 1: Grindnät för realisering av 8-to-1 multiplexern.*

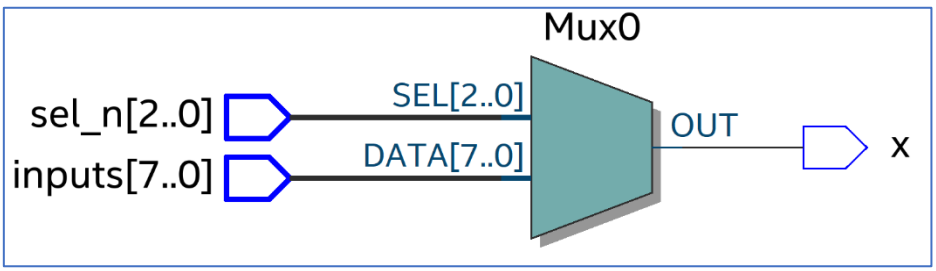
**OBS! Vänd blad!**

Multiplexern kan realiseras via en case-sats i VHDL, såsom visas i modulen *mux_8_to_1* nedan. I detta fall inverteras de ingående selektorsignalerna, då de har anslutits till knappar med aktivt låg signal, dvs. insignalen är låg vid nedtryckning. Eftersom dessa selektorsignaler är inverterande döper vi motsvarande bitvektor till *sel_n.*

För att simulera att knapparna är höga vid nedtryckning används inversen av insignalerna från knapparna i arkitekturen. För enkelhets skull har också insignalerna A - H implementerats via en 8-bits vektor döpt *inputs:*

```
--------------------------------------------------------------------------------
-- @brief Design of a 8-to-1 multiplexer.
--
-- @param inputs[in] Multiplexer inputs connected to slide switches.
-- @param sel_n[in]  Inverted selector bits connected to active low buttons.
-- @param x[out]     Multiplexer output connected to a LED.
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity mux_8_to_1 is
    port(inputs: in std_logic_vector(7 downto 0);
         sel_n : in std_logic_vector(2 downto 0);
         x     : out std_logic);
end entity;

--------------------------------------------------------------------------------
-- @note The selector bits are connected to active low buttons, i.e. the input
--       is low when pressed, hence the inverse values are used internally.
--------------------------------------------------------------------------------
architecture behaviour of mux_8_to_1 is
signal sel: std_logic_vector(2 downto 0) := (others => '1');
begin

    ------------------------------------------------------------------------------
    -- @brief Process that updates the output upon change of the selector signals.
    ------------------------------------------------------------------------------
    output_process: process (sel) is
    begin
        case (sel) is
            when "000"  => x <= inputs(0);
            when "001"  => x <= inputs(1);
            when "010"  => x <= inputs(2);
            when "011"  => x <= inputs(3);
            when "100"  => x <= inputs(4);
            when "101"  => x <= inputs(5);
            when "110"  => x <= inputs(6);
            when "111"  => x <= inputs(7);
            when others => x <= '0';
        end case;
    end process;

    sel <= not sel_n;

end architecture;
```

Det syntetiserade grindnätet visas nedan. Som synes har en multiplexer konstruerats:




*Figur 2: Syntetiserat grindnät för multiplexern.*