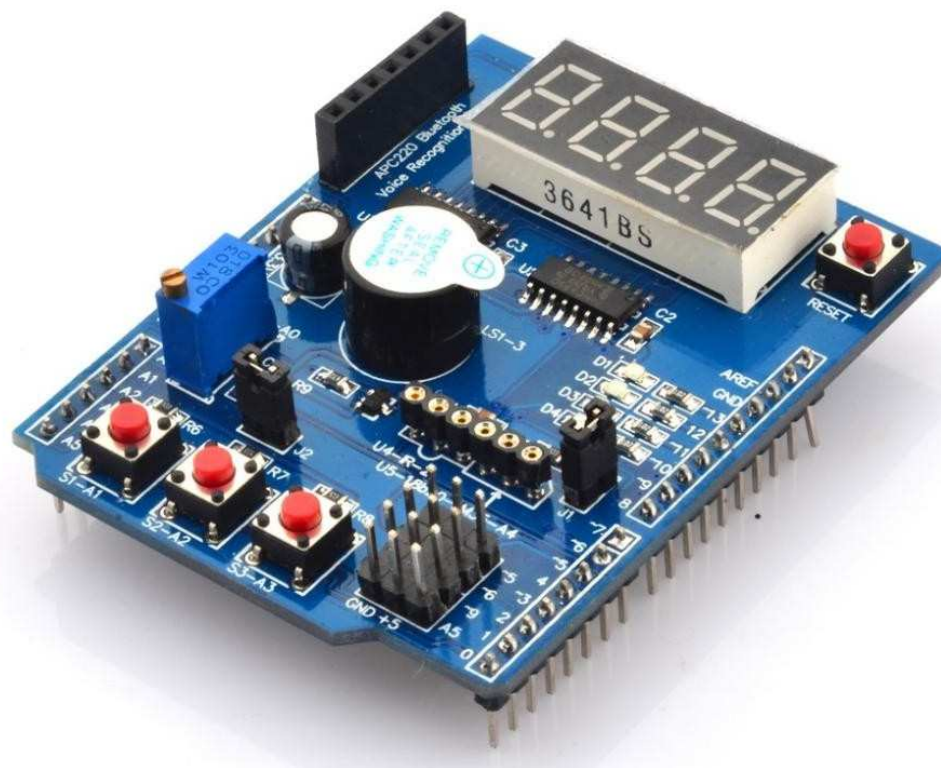


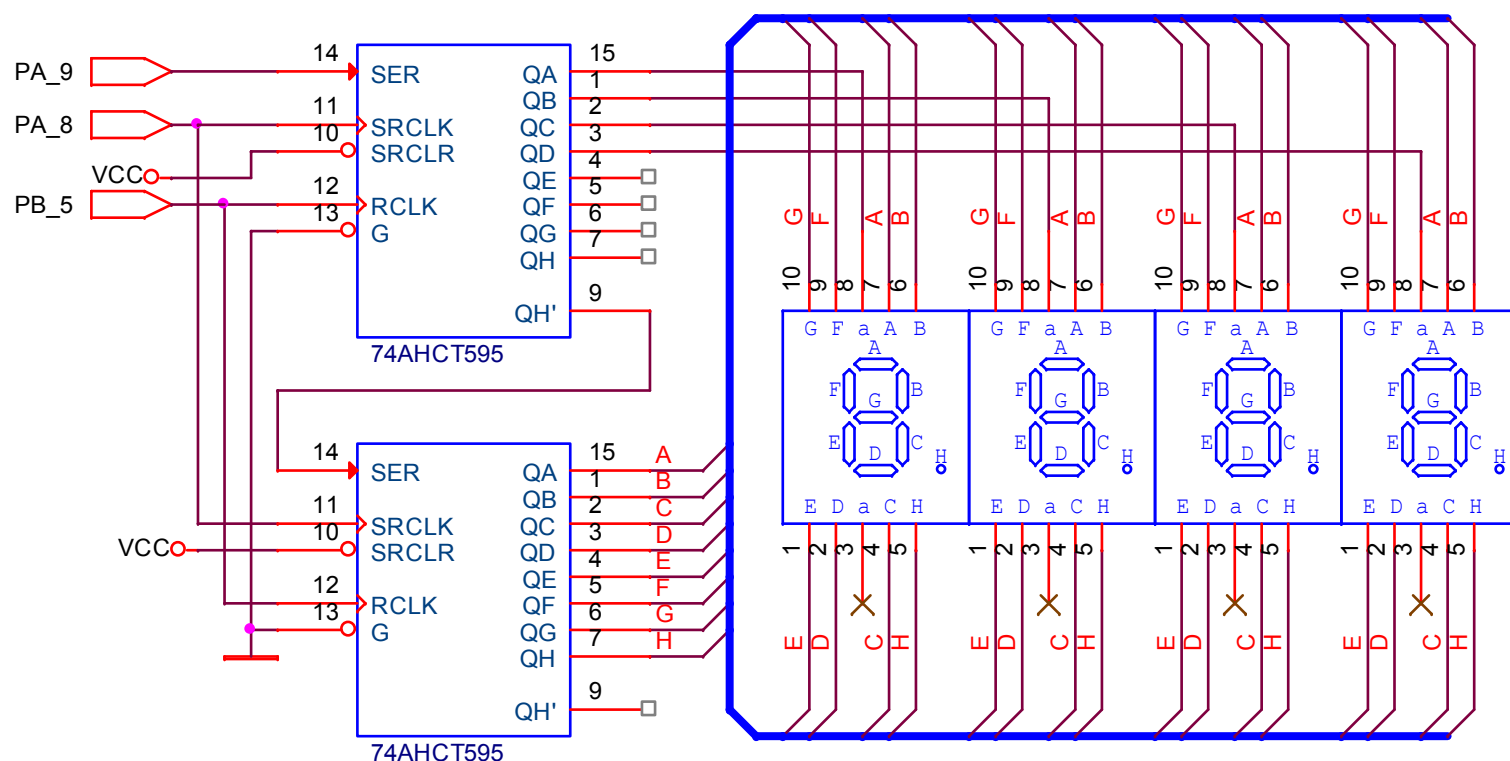
ČTYŘMÍSTNÝ DYNAMICKY OVLÁDANÝ DISPLEJ

Na obrázku je zobrazen nastavný modul, který se zastrčí do konektorů ARDULINO. **Při nasazení na modul NUCLEO je potřeba dát pozor, aby destička trvale nezmáčkla černé tlačítko. Při nasazení na modul DE10 Lite je potřeba dát pozor, aby nedošlo ke zkratu tlačítka reset s kovovým krytem konektoru USB.**

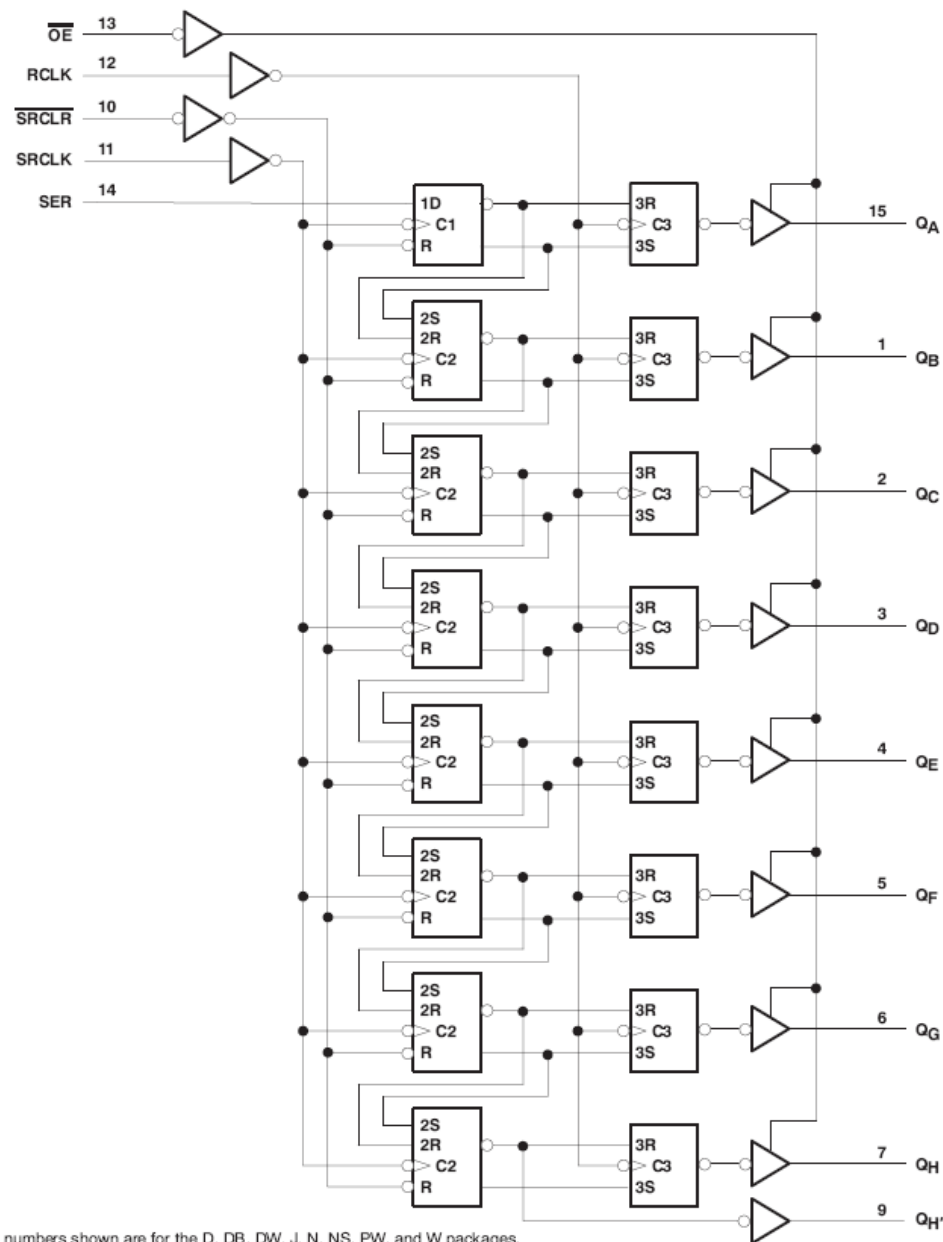
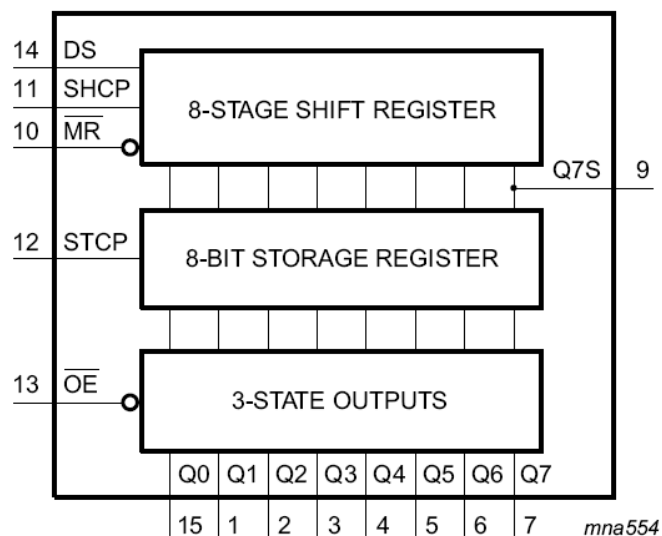


ZAPOJENÍ ČTYŘMÍSTNÉHO DYNAMICKY OVLÁDANÉHO DISPLEJE

Displej na modulu je realizován čtyřmi 7-segmentovými displeji LED se společnou anodou a ovládán dvěma obvody 74595 kaskádně zapojenými. Zobrazovaná informace se ukládá bit po bitu přes sériový vstup (IO vývod 14, AB17 (DE10) a PA_9(NUCLEO) náběžnou hranou hodin SRCLK (IO-11, AA12/PA_8). Po zápisu všech 16 bitů se obsah posuvných registrů přenese náběžnou hranou hodin RCLK (IO-12, AB9/PB_5) do paralelního registru.



VNITŘNÍ STRUKTURA OBVODU 74AHCT595



Pin numbers shown are for the D, DB, DW, J, N, NS, PW, and W packages.

PROPOJENÍ VÝVODŮ ARDUINO NA VÝVODY PLD DE10-LITE

Table 3-8 Pin Assignments for Arduino Uno Expansion Header connector

Schematic Signal Name	FPGA Pin No.	Description	Specific features For Arduino	I/O Standard
Arduino_IO0	PIN_AB5	Arduino IO0	RXD	3.3-V LVTTL
Arduino_IO1	PIN_AB6	Arduino IO1	TXD	3.3-V LVTTL
Arduino_IO2	PIN_AB7	Arduino IO2		3.3-V LVTTL
Arduino_IO3	PIN_AB8	Arduino IO3		3.3-V LVTTL
Arduino_IO4	PIN_AB9	Arduino IO4		3.3-V LVTTL
Arduino_IO5	PIN_Y10	Arduino IO5		3.3-V LVTTL
Arduino_IO6	PIN_AA11	Arduino IO6		3.3-V LVTTL
Arduino_IO7	PIN_AA12	Arduino IO7		3.3-V LVTTL
Arduino_IO8	PIN_AB17	Arduino IO8		3.3-V LVTTL
Arduino_IO9	PIN_AA17	Arduino IO9		3.3-V LVTTL
Arduino_IO10	PIN_AB19	Arduino IO10	SS	3.3-V LVTTL
Arduino_IO11	PIN_AA19	Arduino IO11	MOSI	3.3-V LVTTL
Arduino_IO12	PIN_Y19	Arduino IO12	MISO	3.3-V LVTTL
Arduino_IO13	PIN_AB20	Arduino IO13	SCK	3.3-V LVTTL
Arduino_IO14	PIN_AB21	Arduino IO14	SDA	3.3-V LVTTL
Arduino_IO15	PIN_AA20	Arduino IO15	SCL	3.3-V LVTTL
ARDUINO_RESET_N	PIN_F16	Reset signal, low active.		3.3 V SCHMITT TRIGGER"

ARDUINO KONEKTOR NA DE10-LITE

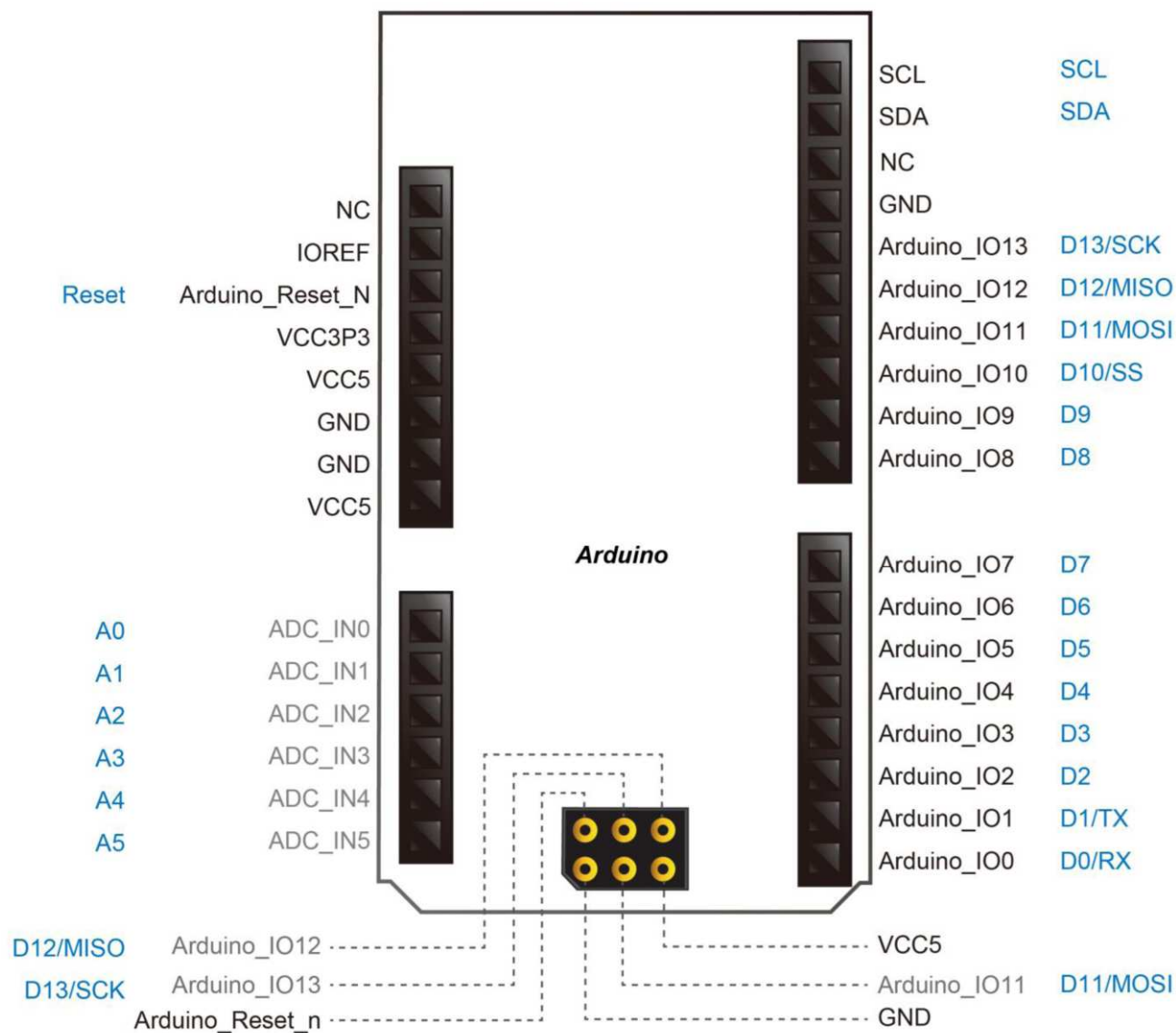
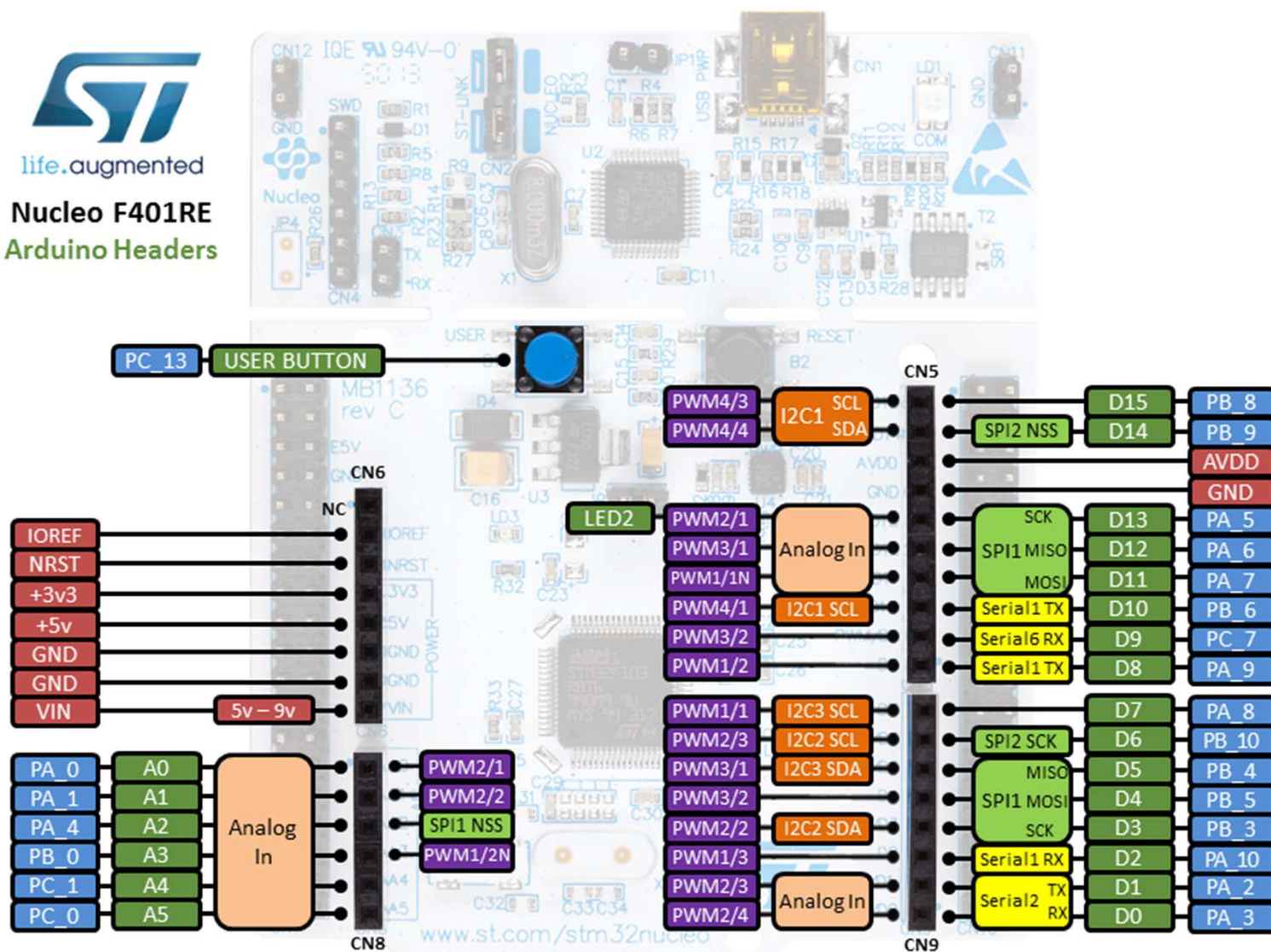


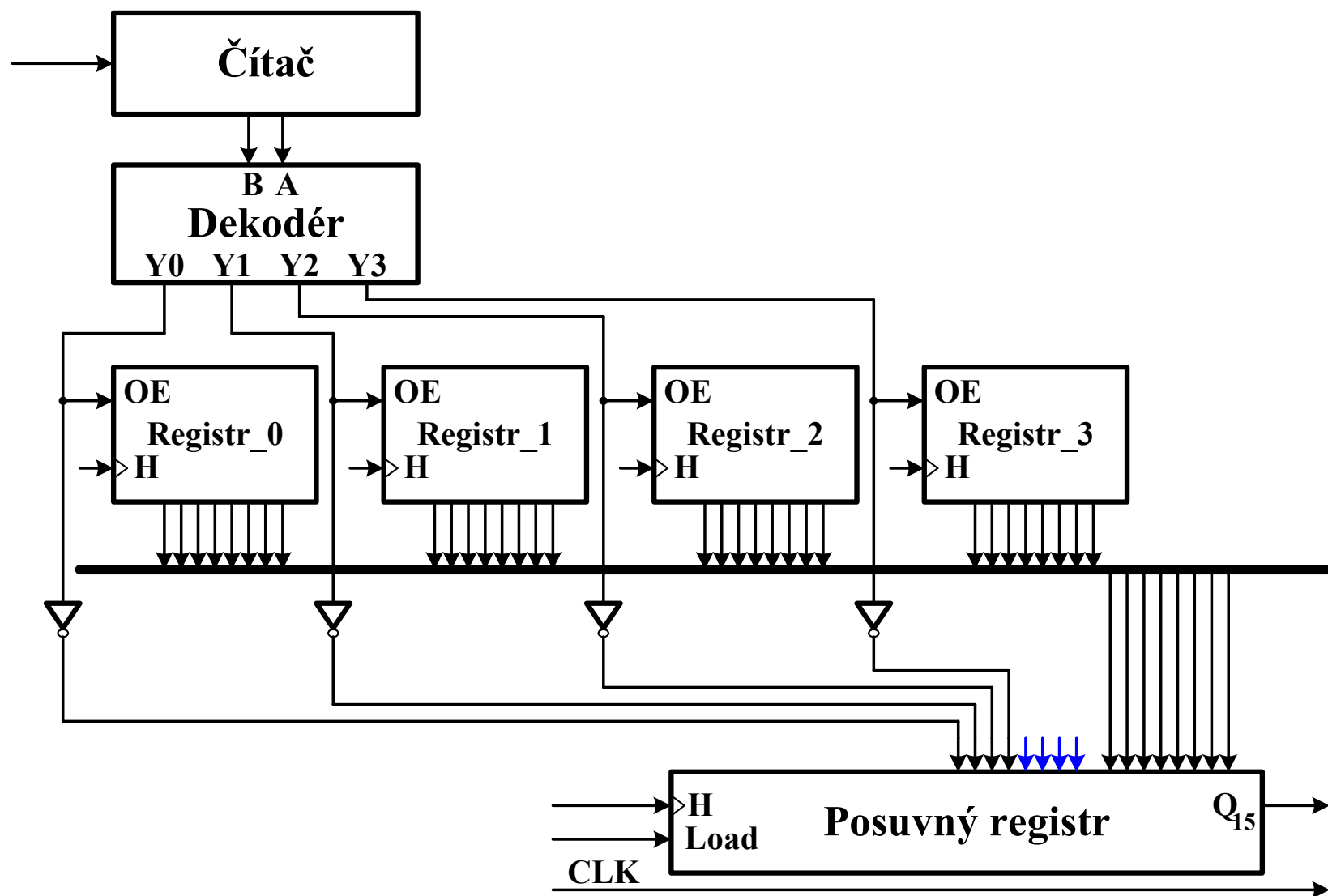
Figure 3-19 lists the all the pin-out signal name of the Arduino Uno connector. The blue font represents the Arduino pin-out definition.

POPIS KONEKTORŮ NA DESCE NUCLEO ST32F401RE

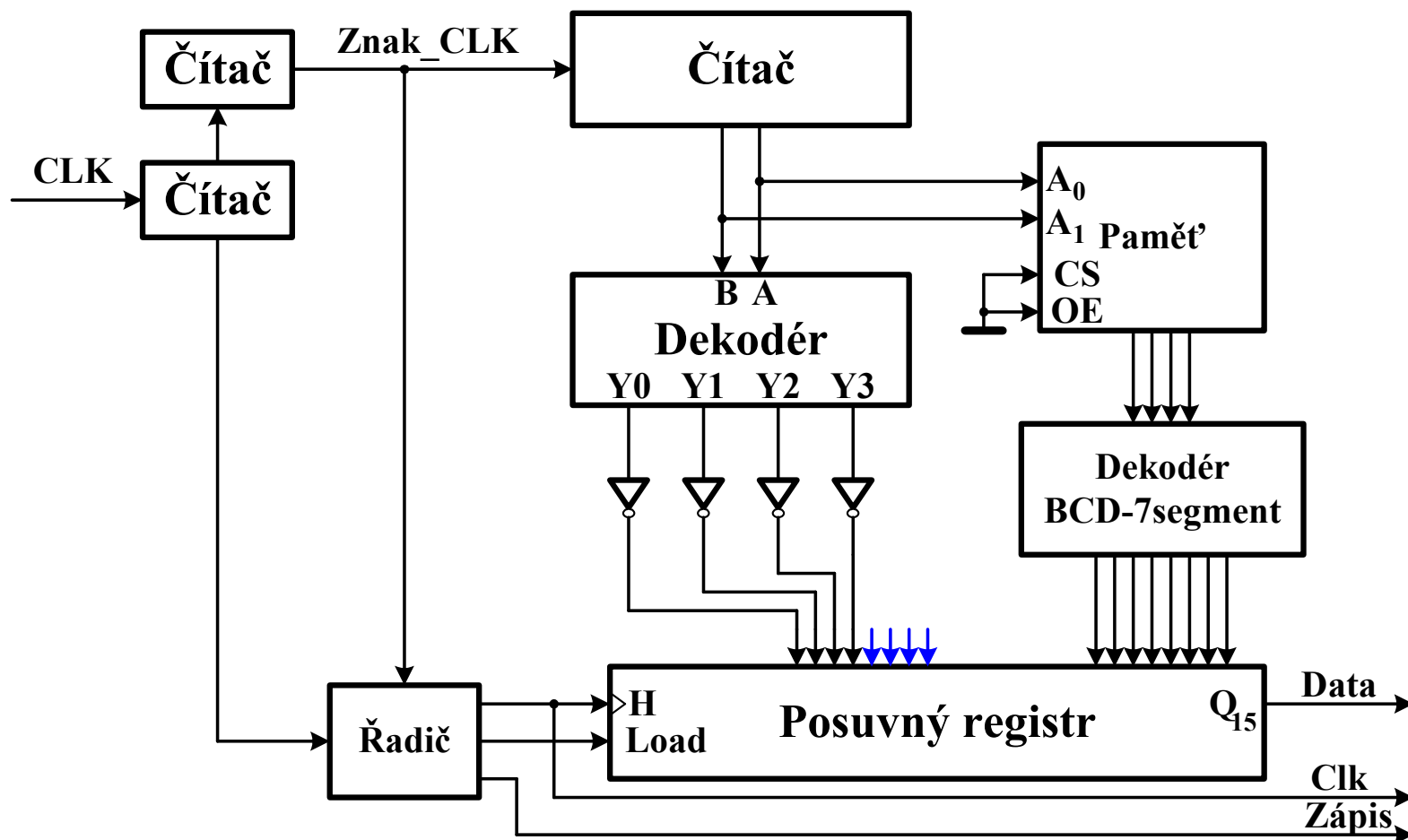

 life.augmented
Nucleo F401RE
 Arduino Headers



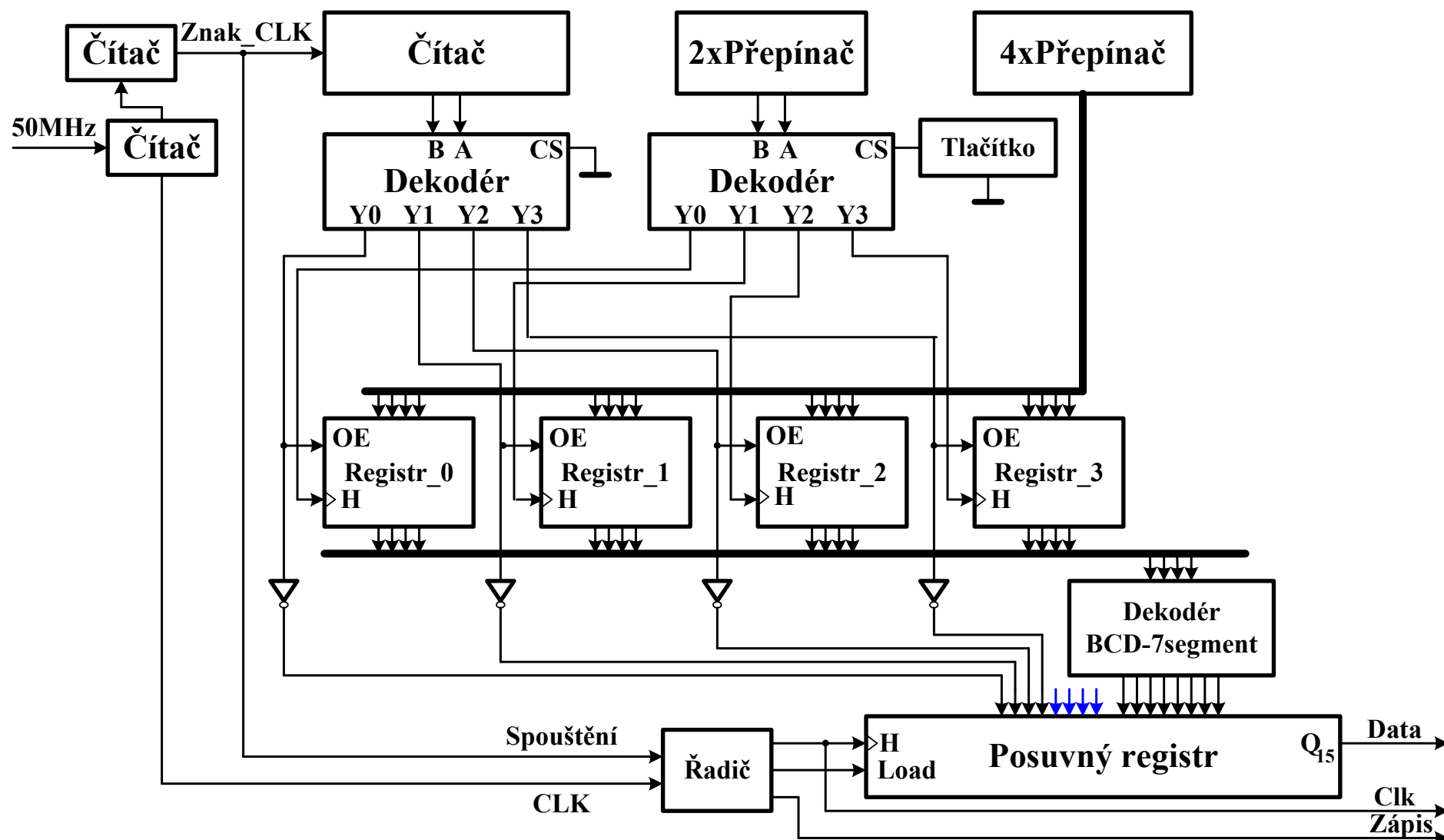
OBVODOVÉ OVLÁDÁNÍ DISPLEJE – PRINCIPIÁLNÍ ŘEŠENÍ



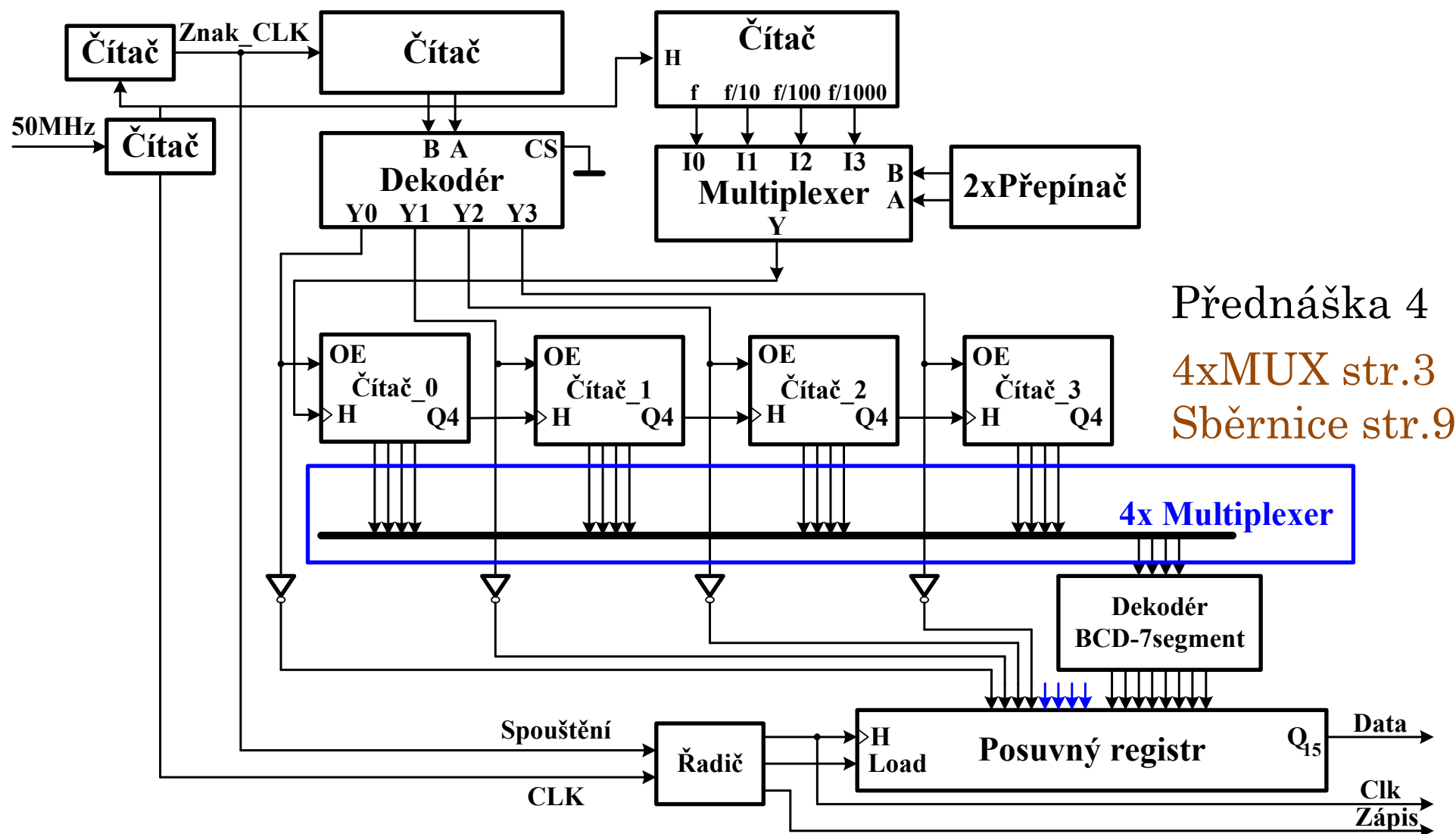
OBVODOVÉ OVLÁDÁNÍ DISPLEJE – ŘEŠENÍ S PAMĚTÍ



OBVODOVÉ OVLÁDÁNÍ DISPLEJE – ŘEŠENÍ S REGISTRY

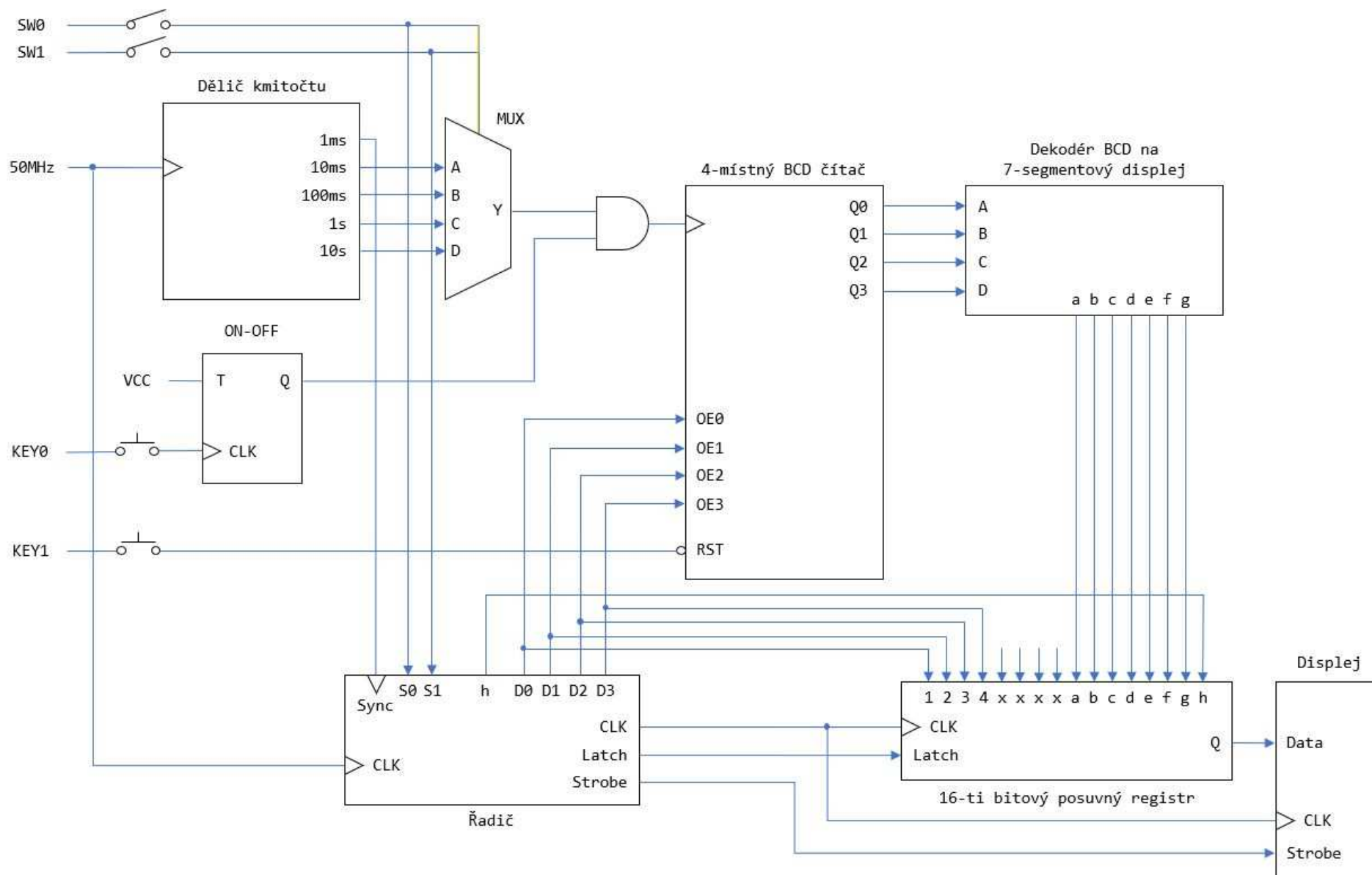


OBVODOVÉ OVLÁDÁNÍ DISPLEJE – ŘEŠENÍ S ČÍTAČI



Přednáška 4
4xMUX str.3
Sběrnice str.9

OBVODOVÉ OVLÁDÁNÍ DISPLEJE – ŘEŠENÍ S ČÍTAČI



U všech návrhů se předpokládá spuštění řadiče od změny hodinového signálu zajišťujícího přepínání jednotlivých segmentů. Doba trvání impulsu v log.1 bude velká (cca 2,5ms) a bude-li řadič spouštěn úrovní, je nebezpečí jeho opětovného spuštění. V takovém případě musíme náběžnou nebo sestupnou hranou vytvořit impuls (log.1), který bude ukončen spuštěním řadiče nebo generováním signálu LOAD.

Generování potřebných hodinových signálů

Na přípravku jsou k dispozici dva hodinové signály (10MHz a 50MHz). Z jednoho z nich bude potřeba vytvořit hodinový signál pro přepínání segmentovek (Znak_CLK) a hodinový signál pro řadič. V případě použití TTL obvodů se předpokládá výběr obvodu z katalogu Texas Instruments. Následuje ověření zda se zvolený typ nachází i v knihovně PALASM2 (QuartusII – schématický návrh – Block Editor – Libraries – others – maxplus2).

- **Dekodér** – BIN→7SEG, BCD →7SEG
 - ✓ Navržený jako LKO se zvoleným typem obvodů
 - ✓ Realizovaný pamětí ROM (tabulka) nebo RAM
 - ✓ Integrovaný obvod z TTL řady (7447)
- **Čítač** – mod 4
 - ✓ Klasický návrh s PČ - JK, T, D
 - ✓ Realizovaný pamětí ROM a sčítačkou
 - ✓ Realizovaný mikroprogramovatelnou strukturou
 - ✓ Integrovaný obvod z TTL řady (např. 7474)
- **Posuvný registr** – paralelně sériový
 - ✓ Realizovaný obvody TTL (např. 74164)
 - ✓ Realizovaný ve VHDL

- **Hodinové signály** (děliče kmitočtu) – z 10/50 MHz
 - ✓ Integrované čítače dekadické/binární (7490, 7493)
 - ✓ Řešení ve VHDL
- **Realizace 4 nepřekrývajících signálů**
 - ✓ Dekodérem a čítačem např. 74139, 74163, 7474
 - ✓ Speciálním čítačem se stavy 1, 2, 4 a 8
- **Řadič** – obvod zajišťující přenos 16 bitů do posuvných registrů a jejich vysunutí do displeje.
 - ✓ Obvodový řadič
 - ✓ Čítačem a dekódujícím obvodem
 - ✓ Mikroprogramovatelným řadičem

NÁVRH LSO ZE STAVOVÉHO DIAGRAMU

Pro řízení dekodéru i vybavení jednotlivých zobrazovaných hodnot, bude potřeba čítač modulo 4. Ten můžeme navrhnout klasiky s PČ D, JK, T nebo pomocí návrhového systému QuartusII.

- ❖ Založíme nový projekt
- ❖ V položce New vybereme State Machine File
- ❖ Umístíme na plochu 4 stavy
- ❖ Vytvoříme mezi nimi přechody, které v tomto případě budou nepodmíněné.
- ❖ Nadefinujeme dvě výstupní proměnné
- ❖ Klikneme na jednotlivé stavy a definujeme v položce Action odpovídající stav výstupních proměnných.
- ❖ Ikonou HDL necháme vytvořit soubor VHDL
- ❖ Necháme zkompileovat vytvořený projekt
- ❖ Vytvoříme simulační soubor a ověříme návrh

Více v - Návod na použití programu Quartus.pdf

NÁVRH LSO ZE STAVOVÉHO DIAGRAMU

Quartus Prime Lite Edition - C:/Users/GG/Desktop/S4/S4 - S4

File Edit View Project Assignments Processing Tools Window Help

Search altera.com

Project Navigator Hierarchy

Entity: Instance

MAX 10: 10M08DAF484C8G

S4

Tasks

Compilation

Task

Compile Design

Analysis & Synthesis

Fitter (Place & Route)

Assembler (Generate programmin

TimeQuest Timing Analysis

EDA Netlist Writer

Edit Settings

Program Device (Open Programmer)

Input Table

Input Port

1 reset

2 clock

Output Table

Output Port

1 output1

2 output2

State Table

	Source State	Destination State	Transition (In Verilog or VHDL 'OTHERS')
1	state1	state2	
2	state4	state1	
3	state2	state3	

General Inputs Outputs States Transitions Actions

Messages

Type ID Message

293000 Quartus Prime Full Compilation was successful. 0 errors, 15 warnings

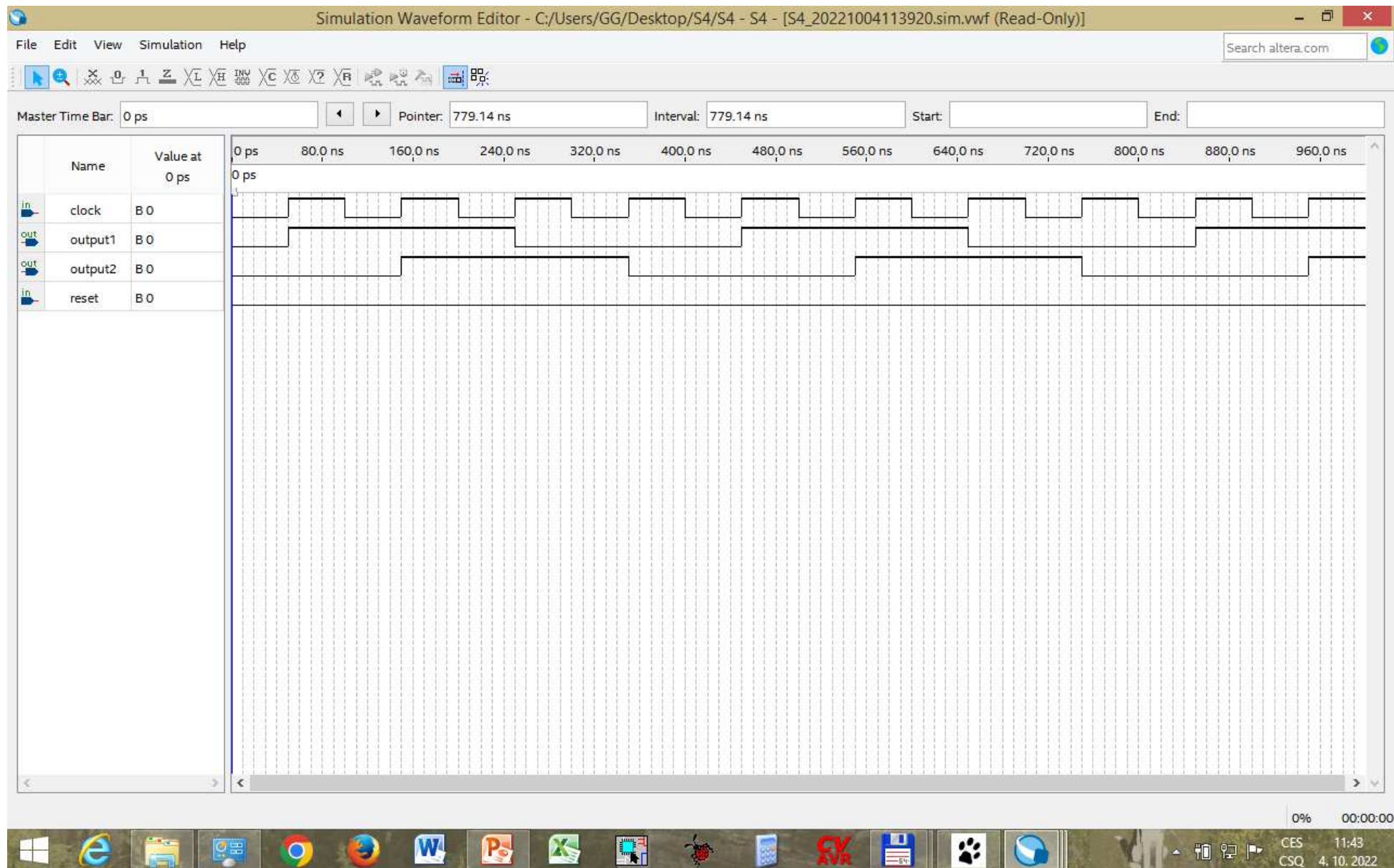
System (7) Processing (120)

100% 00:01:22

CES 11:43
CSQ 4. 10. 2022

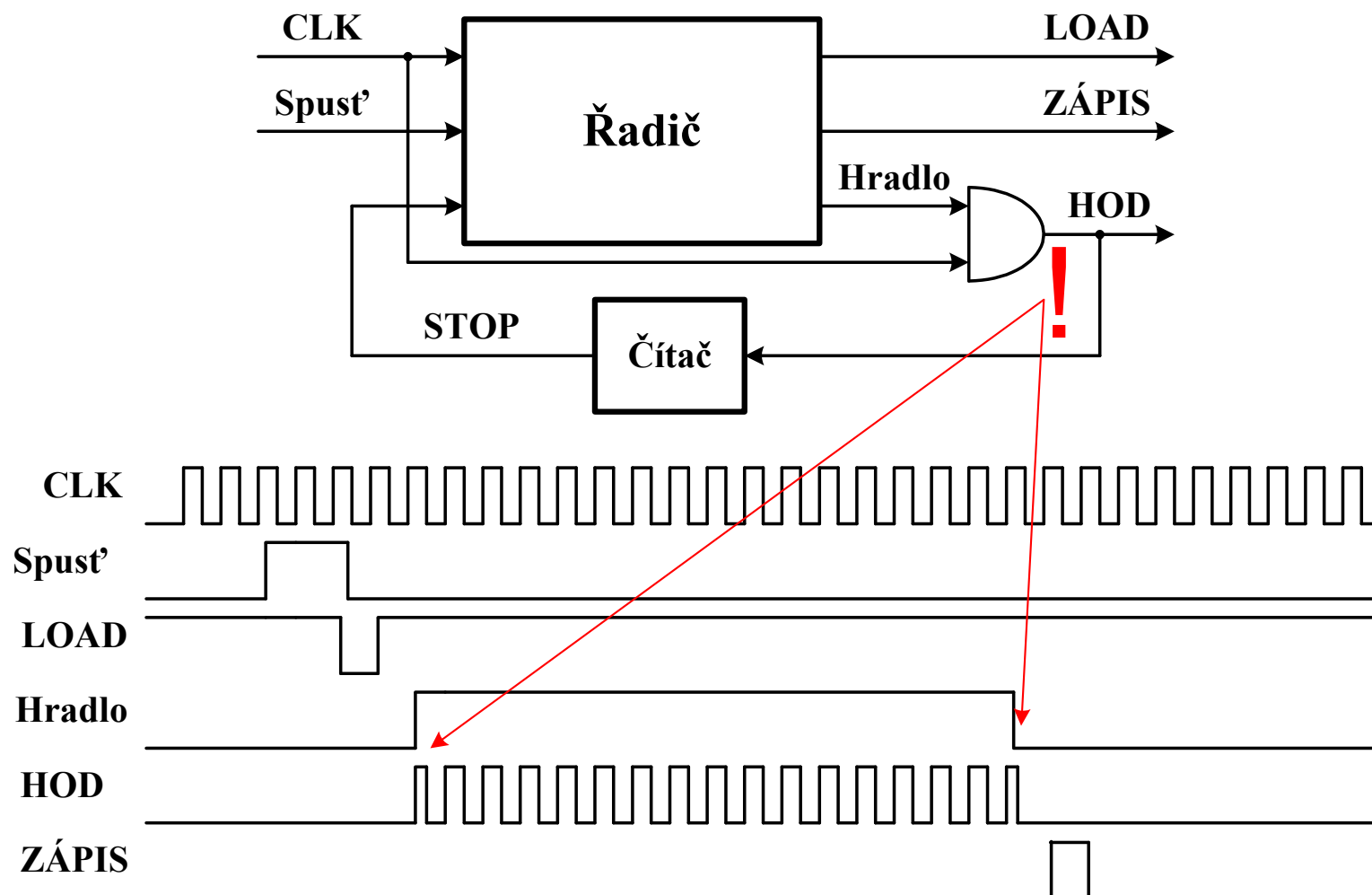
```
graph LR; state1((state1)) --> state2((state2)); state2 --> state3((state3)); state3 --> state4((state4)); state4 --> state1;
```

NÁVRH LSO ZE STAVOVÉHO DIAGRAMU

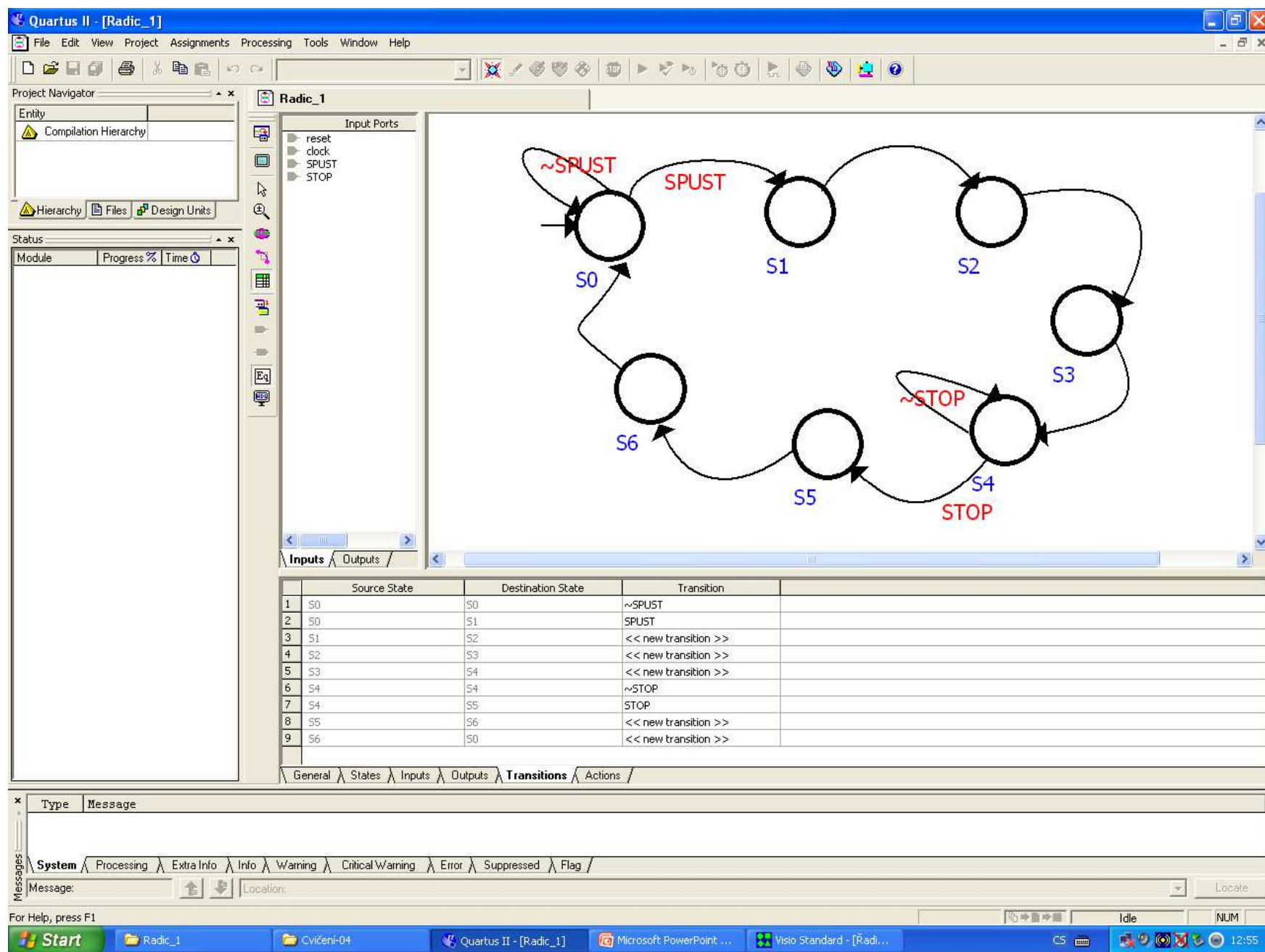


REALIZACE ŘADIČE PRO DYNAMICKY OVLÁDANÝ DISPLEJ - VAR.1

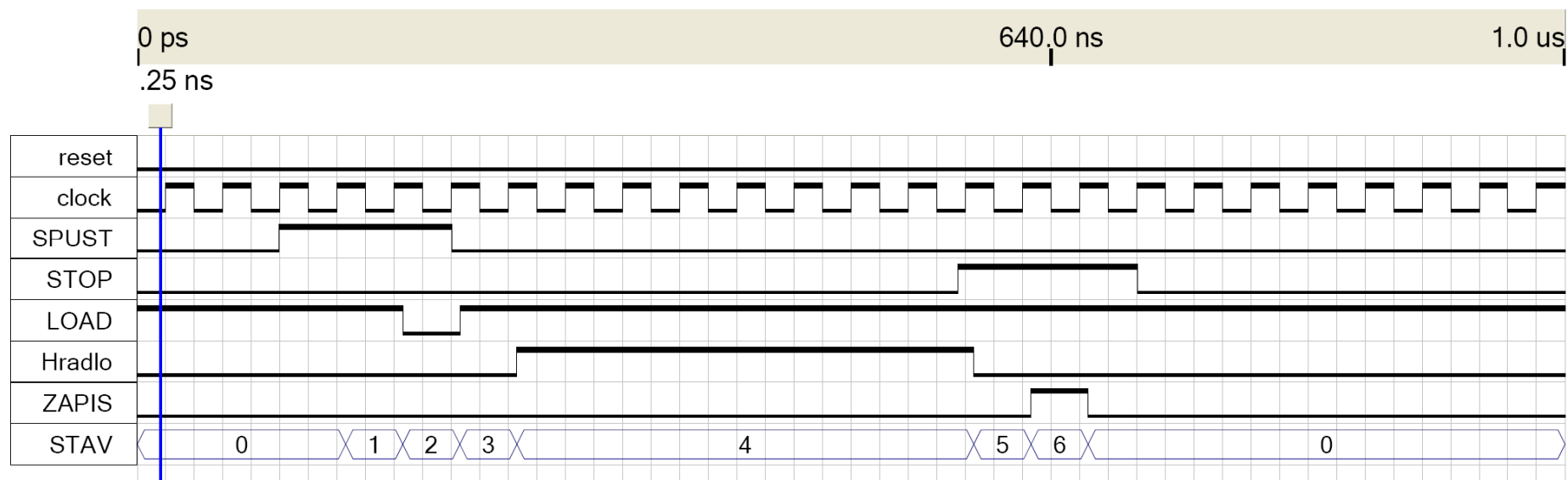
Řadič s malým počtem stavů využívající hradlování hodinového signálu. **Výhody:** signál HOD má 2x vyšší kmitočet proti ostatním řešením. **Nevýhody:** Nebezpečí krátkého impulsu HOD.



REALIZACE ŘADIČE PRO DYNAMICKY OVLÁDANÝ DISPLEJ - VAR.1



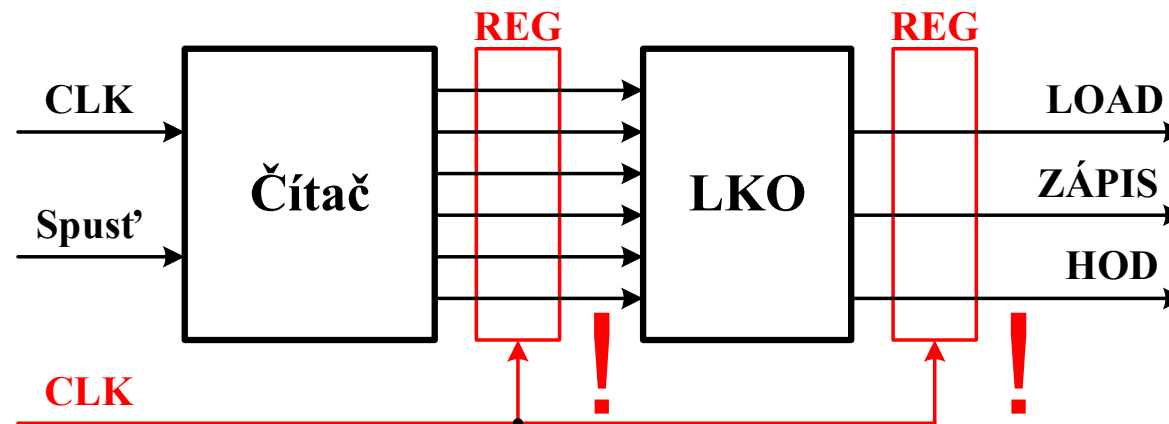
REALIZACE ŘADIČE PRO DYNAMICKY OVLÁDANÝ DISPLEJ - VAR.1



REALIZACE ŘADIČE PRO DYNAMICKY OVLÁDANÝ DISPLEJ - VAR.2

Řadič je tvořen čítačem, jehož stavy jsou kombinačním obvodem dekodovány na potřebné řídicí signály. **Výhody:** Snadný, byť složitější návrh řídicích signálů. **Nevýhody:** Nebezpečí se skrývá ve vlastnostech čítače a postupných změn jeho výstupů (asynchronní/synchronní). Řídicí signály mohou vykazovat statické hazardní stavy.

Odstranění případných problémů pomocí vyrovnávacích registrů, řízených stejným hodinovým signálem.



REALIZACE ŘADIČE PRO DYNAMICKY OVLÁDANÝ DISPLEJ - VAR.2

K realizaci bude potřeba:

Prodleva – 1 stav, LOAD – 2 stavy, HOD – 32 stavů, ZÁPIS – 2 stavy

Výstupní funkce vedou na mapy 7 proměnných – trochu problém.

LKO budeme realizovat pamětí ROM.

Čítač	LOAD HOD ZÁPIS	Čítač	LOAD HOD ZÁPIS	Čítač	LOAD HOD ZÁPIS	Čítač	LOAD HOD ZÁPIS	Čítač	LOAD HOD ZÁPIS
0	1 0 0	8	1 1 0	16	1 1 0	24	1 1 0	32	1 1 0
1	1 0 0	9	1 0 0	17	1 0 0	25	1 0 0	33	1 0 0
2	0 0 0	10	1 1 0	18	1 1 0	26	1 1 0	34	1 1 0
3	1 0 0	11	1 0 0	19	1 0 0	27	1 0 0	35	1 0 0
4	1 1 0	12	1 1 0	20	1 1 0	28	1 1 0	36	1 0 1
5	1 0 0	13	1 0 0	21	1 0 0	29	1 0 0	37	1 0 0
6	1 1 0	14	1 1 0	22	1 1 0	30	1 1 0	38	1 0 0
7	1 0 0	15	1 0 0	23	1 0 0	31	1 0 0	39	1 0 0

REALIZACE ŘADIČE PRO DYNAMICKY OVLÁDANÝ DISPLEJ - VAR.2

```
library IEEE;
```

```
use IEEE.std_logic_1164.all;
```

```
entity ROM64x3 is
```

```
port (address : in INTEGER range 0 to 63;
```

```
      data : out std_logic_vector (2 downto 0));
```

```
end entity ROM64x3;
```

```
architecture LKO of ROM64x3 is
```

```
type rom_array is array (0 to 63) of std_logic_vector (2 downto 0);
```

```
constant rom : rom_array :=
```

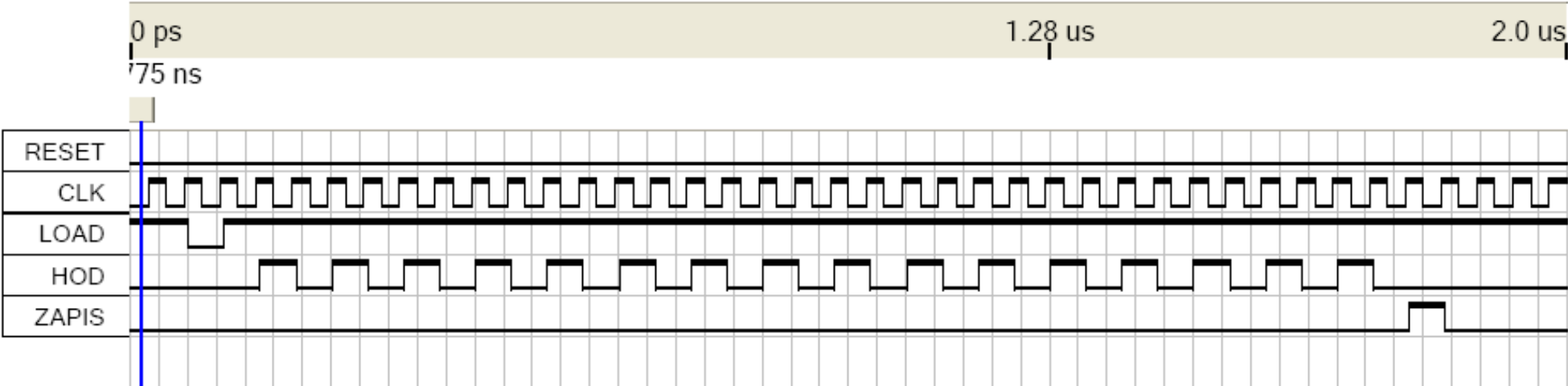
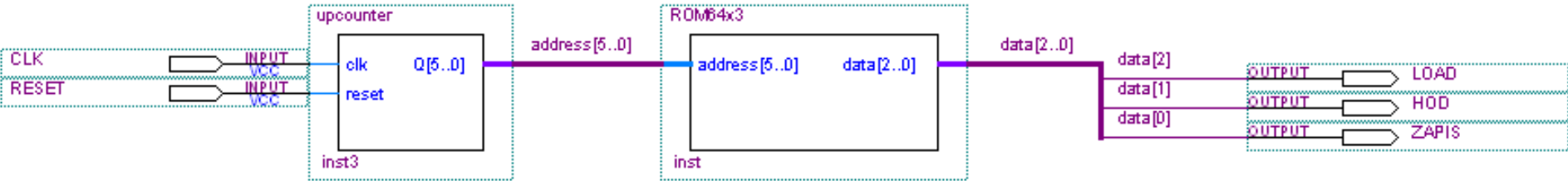
```
  ("100","100","000","100","110","100","110","100","110","100","110","100",  
   "110","100","110","100","110","100","110","100","110","100","110","100",  
   "110","100","110","100","110","100","110","100","110","100","110","100",  
   "101","100","100","100","100","100","100","100","100","100","100","100",  
   "100","100","100","100","100","100","100","100",  
   "100","100","100","100","100","100","100","100");
```

```
begin
```

```
  data <= rom(address);
```

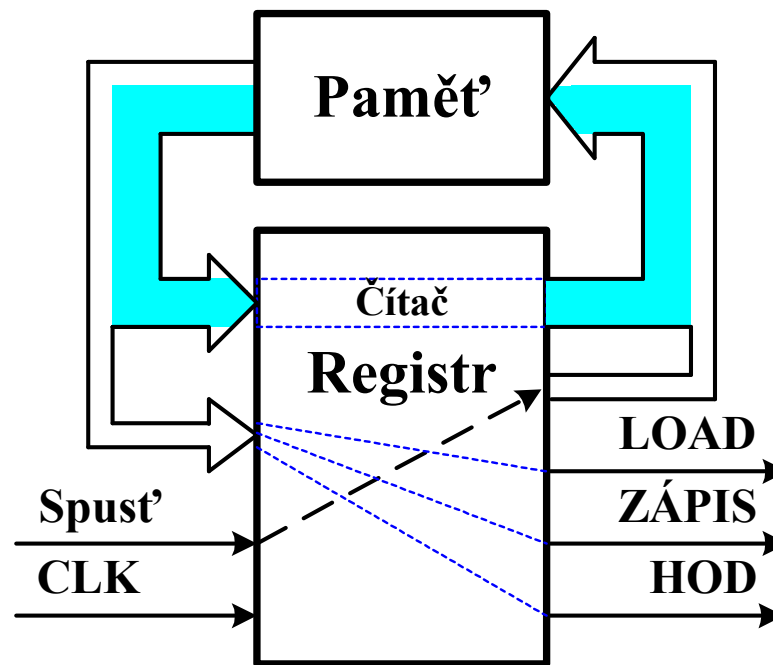
```
end architecture LKO;
```

REALIZACE ŘADIČE PRO DYNAMICKY OVLÁDANÝ DISPLEJ - VAR.2

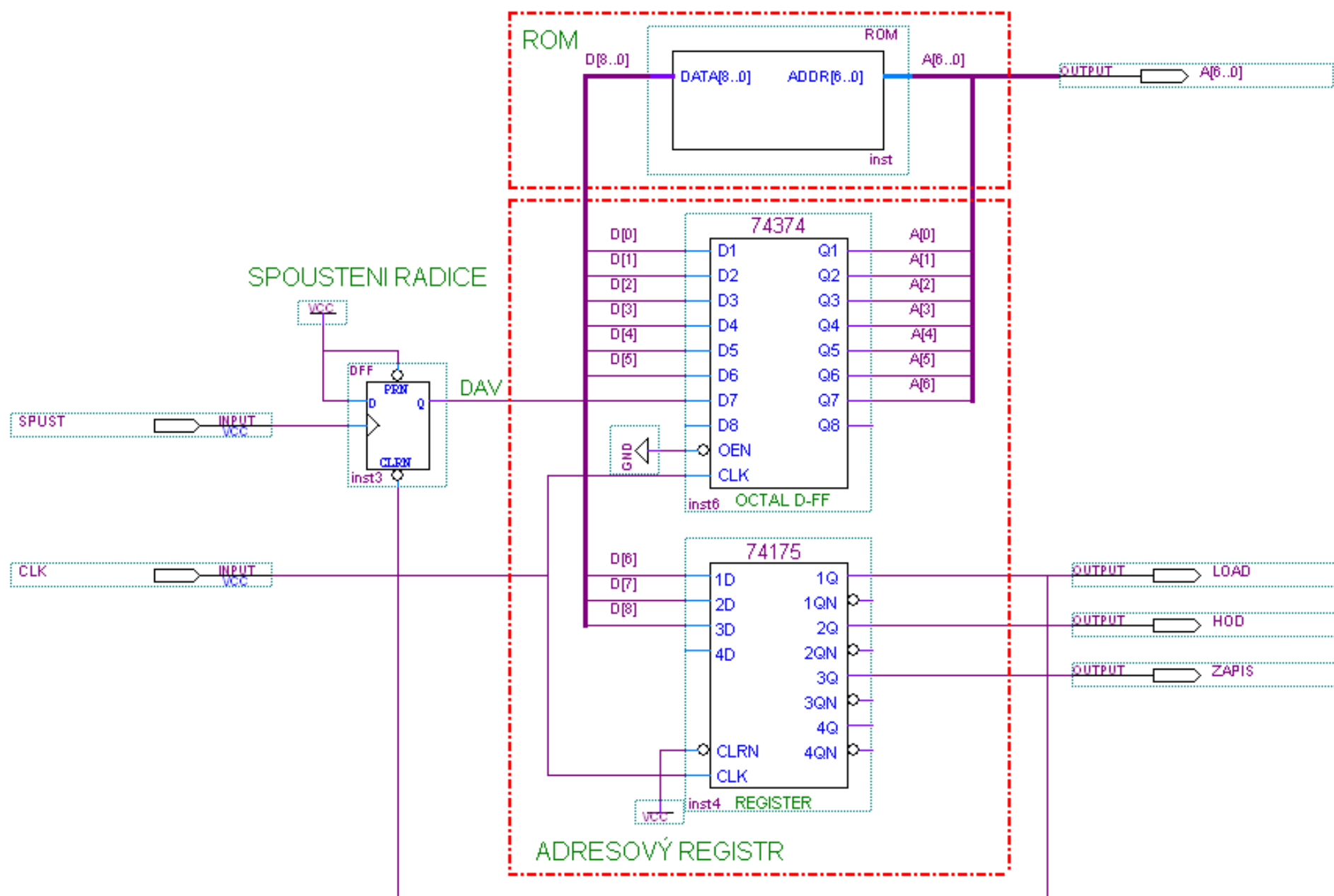


Mikroprogramovatelný řadič, realizovaný registrem a pamětí ROM (LKO, RAM). Na každém paměťovém místě paměti je uložen stav řídicích signálů (Load, Zápis, Hod) a následující adresy, z které se bude číst nový stav řídicích signálů a následující adresy.

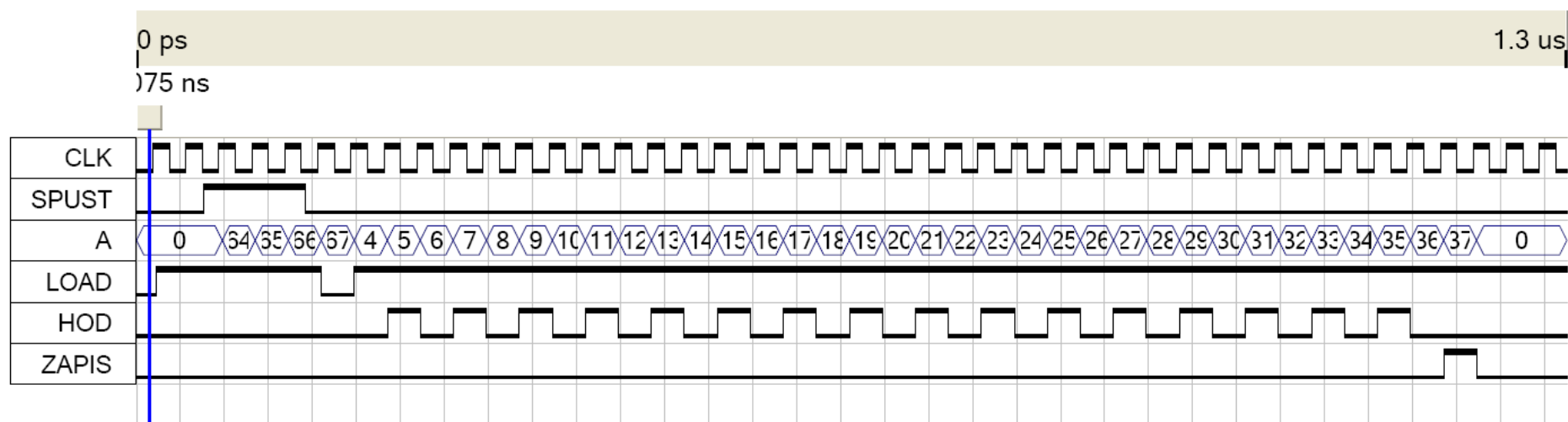
Výhody: Jednoduchý bezproblémový návrh. **Nevýhody:** Vzhledem k velkému počtu hodinových impulzů definovaných dvěma stavy obvodu, je vytvoření obsahu paměti rozsáhlé.



REALIZACE ŘADIČE PRO DYNAMICKY OVLÁDANÝ DISPLEJ - VAR.3



REALIZACE ŘADIČE PRO DYNAMICKY OVLÁDANÝ DISPLEJ - VAR.3



Mikroprogramovatelný řadič, realizovaný registrem a pamětí.

- Vstupem obvodu bude signál DAV a Hodiny
- Výstupem budou signály Nul, Load, Hod a Ready.
- Operačním kódem je signál DAV.
- Výstupem paměti ROM budou řídicí signály Nul, Load, Hod, Ready a následující adresa do paměti ROM ($A_4 A_3 A_2 A_1 A_0$) ukládaná do adresového registru spolu se signálem DAV ($DAV A_4 A_3 A_2 A_1 A_0$ = adresa do paměti ROM).

Paměť ROM má konfiguraci 64x9 bitů. Adresa, která disponuje 6 bity, se skládá z bitů v následujícím pořadí

DAV (DATA Valid) a bity současné adresy A_4, A_3, A_2, A_1, A_0 .

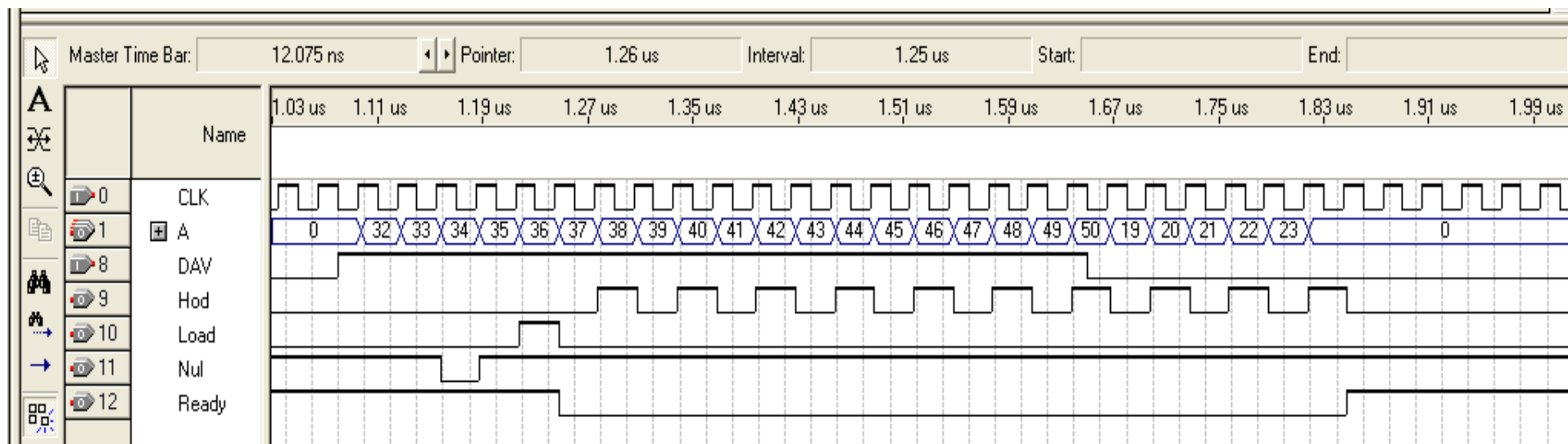
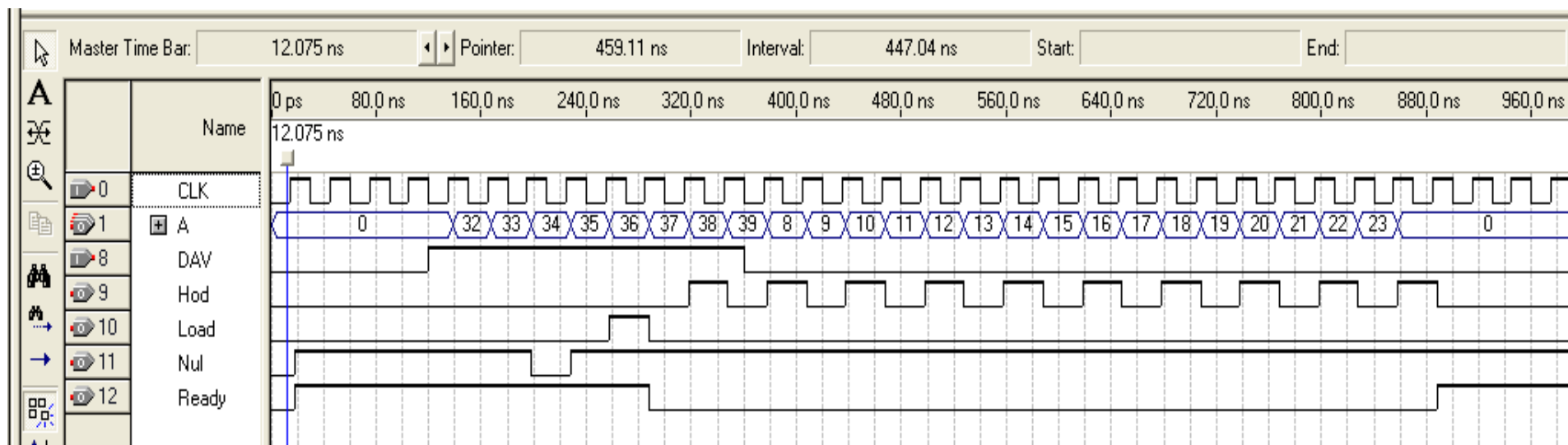
Výstupem paměti ROM jsou řídicí bity v pořadí

Nul, Load, Hod, Ready a $(A_4^{i+1}, A_3^{i+1}, A_2^{i+1}, A_1^{i+1}, A_0^{i+1})$,

z které se bude číst nový výstupní stav řídicích signálů a další následující adresy.

ŘEŠENÍ PŘÍKLADU OBDOBNÉHO ŘADIČE

Ze simulace pak získáme následující průběhy signálů.



ŘEŠENÍ PŘÍKLADU OBDOBNÉHO ŘADIČE

type ROM_TABLE is array (0 to 63) of ROM_WORD;

-- ADRESA bude 6 bitů složených v pořadí DAV A4 A3 A2 A1 A0

-- DATA budou 9 bitů složených z Nul, Load, HOD, Ready, A4 A3 A2 A1 A0

constant ROM: ROM_TABLE := ROM_TABLE'(

ROM_WORD("1001"&"00000"),	-- ADR DAV=0 ADR=00000	0
ROM_WORD("0001"&"00000"),	-- ADR DAV=0 ADR=00001	1
ROM_WORD("1001"&"00000"),	-- ADR DAV=0 ADR=00010	2
ROM_WORD("1101"&"00000"),	-- ADR DAV=0 ADR=00011	3

-- Hodnoty pro převod zapsány při signálu DAV=1 a řadič definitivně spuštěn

ROM_WORD("1000"&"00101"),	-- ADR DAV=0 ADR=00100	4
ROM_WORD("1010"&"00110"),	-- ADR DAV=0 ADR=00101	5
ROM_WORD("1000"&"00111"),	-- ADR DAV=0 ADR=00110	6
ROM_WORD("1010"&"01000"),	-- ADR DAV=0 ADR=00111	7
ROM_WORD("1000"&"01001"),	-- ADR DAV=0 ADR=01000	8
ROM_WORD("1010"&"01010"),	-- ADR DAV=0 ADR=01001	9
ROM_WORD("1000"&"01011"),	-- ADR DAV=0 ADR=01010	10
ROM_WORD("1010"&"01100"),	-- ADR DAV=0 ADR=01011	11
ROM_WORD("1000"&"01101"),	-- ADR DAV=0 ADR=01100	12
ROM_WORD("1010"&"01110"),	-- ADR DAV=0 ADR=01101	13

ŘEŠENÍ PŘÍKLADU OBDOBNÉHO ŘADIČE

-- DATA budou 9 bitů složených z Nul, Load, HOD, Ready, A4 A3 A2 A1 A0

ROM_WORD("1001"&"00001"),	-- ADR DAV=1 ADR=00000	32
ROM_WORD("0001"&"00010"),	-- ADR DAV=1 ADR=00001	33
ROM_WORD("1001"&"00011"),	-- ADR DAV=1 ADR=00010	34
ROM_WORD("1101"&"00100"),	-- ADR DAV=1 ADR=00011	35

-- Hodnoty pro převod zapsány signálem LOAD – řadič definitivně spuštěn

ROM_WORD("1000"&"00101"),	-- ADR DAV=1 ADR=00100	36
ROM_WORD("1010"&"00110"),	-- ADR DAV=1 ADR=00101	37
ROM_WORD("1000"&"00111"),	-- ADR DAV=1 ADR=00110	38
ROM_WORD("1010"&"01000"),	-- ADR DAV=1 ADR=00111	39
ROM_WORD("1000"&"01001"),	-- ADR DAV=1 ADR=01000	40
ROM_WORD("1010"&"01010"),	-- ADR DAV=1 ADR=01001	41
ROM_WORD("1000"&"01011"),	-- ADR DAV=1 ADR=01010	42
ROM_WORD("1010"&"01100"),	-- ADR DAV=1 ADR=01011	43
ROM_WORD("1000"&"01101"),	-- ADR DAV=1 ADR=01100	44
ROM_WORD("1010"&"01110"),	-- ADR DAV=1 ADR=01101	45
ROM_WORD("1000"&"01111"),	-- ADR DAV=1 ADR=01110	46
ROM_WORD("1010"&"10000"),	-- ADR DAV=1 ADR=01111	47