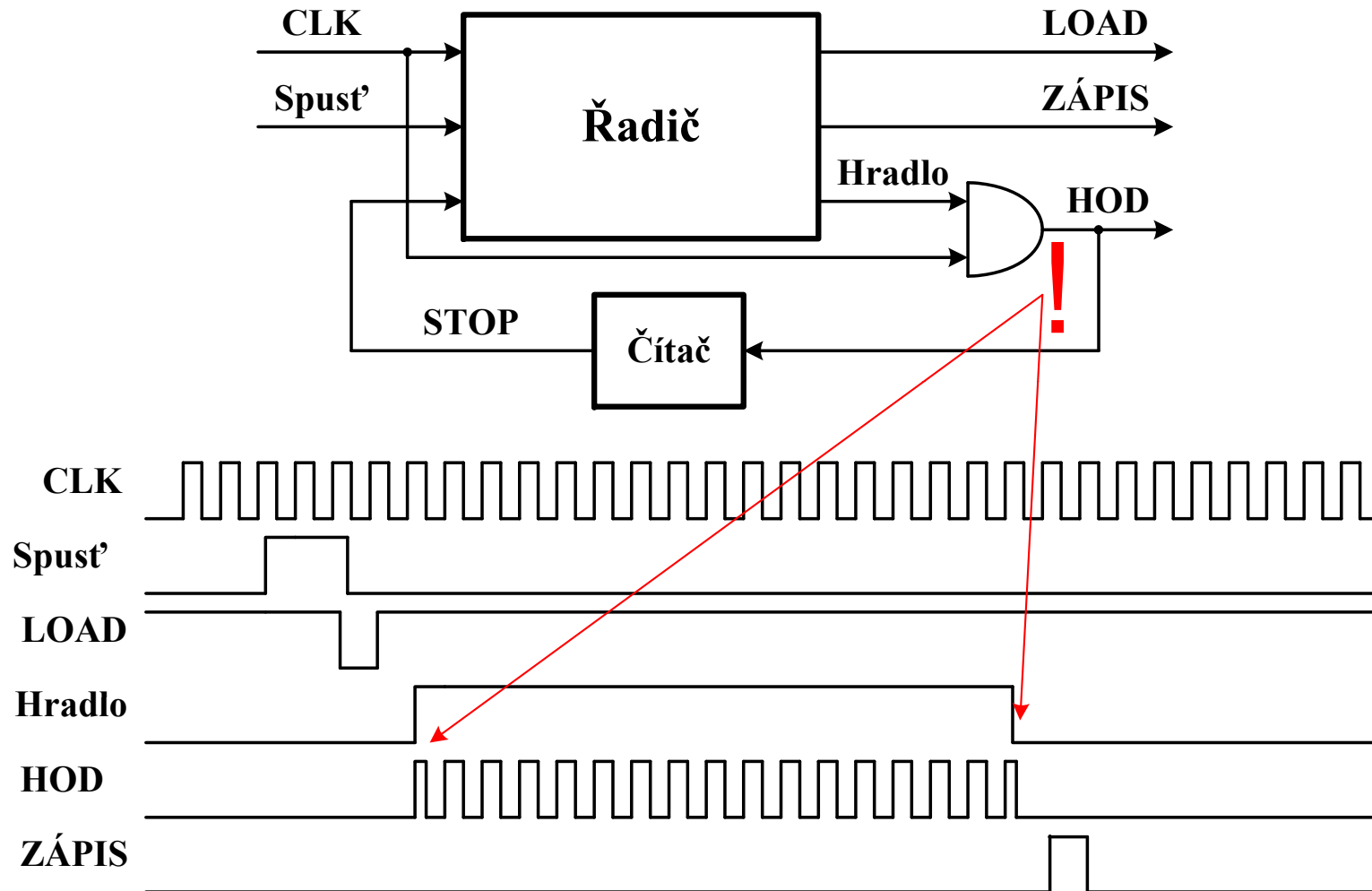


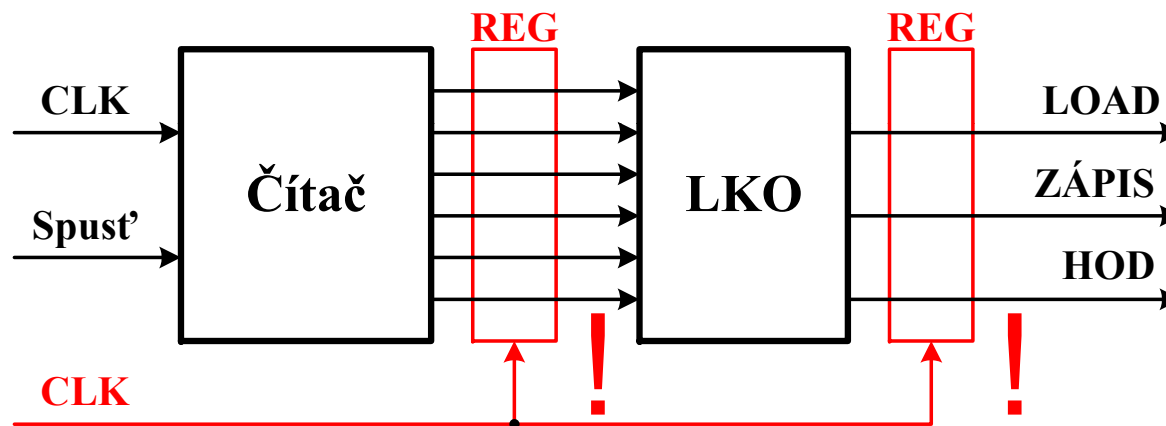
REALIZACE ŘADIČE PRO DYNAMICKY OVLÁDANÝ DISPLEJ - VAR.1

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



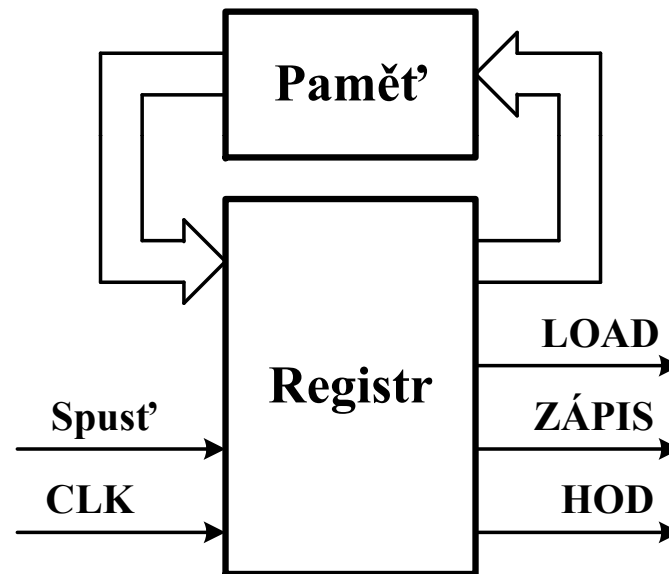
REALIZACE ŘADIČE PRO DYNAMICKY OVLÁDANÝ DISPLEJ - VAR.2

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.3

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 SPOUŠTĚCÍHO IMPULZU SPUSŤ

U všech návrhů se přepokládá spuštění řadiče od změny hodinového signálu zajišťujícího přepínání jednotlivých segmentů. Doba trvání tohoto impulsu v log.1 je velká a je nebezpečí opětovného spuštění řadiče. Z tohoto důvodu bude náběžnou případně sestupnou hranou vytvořen impuls (log.1), který bude ukončen spuštěním řadiče a vytvořením 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 segmentů (Znak_CLK) a hodinový signál pro řadič. V tomto případě předpokládám, že si student z katalogu Texas Instruments si vybere vhodný typ, který se nachází i v knihovně PALASM2.

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

S4

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	

Messages

Type ID Message

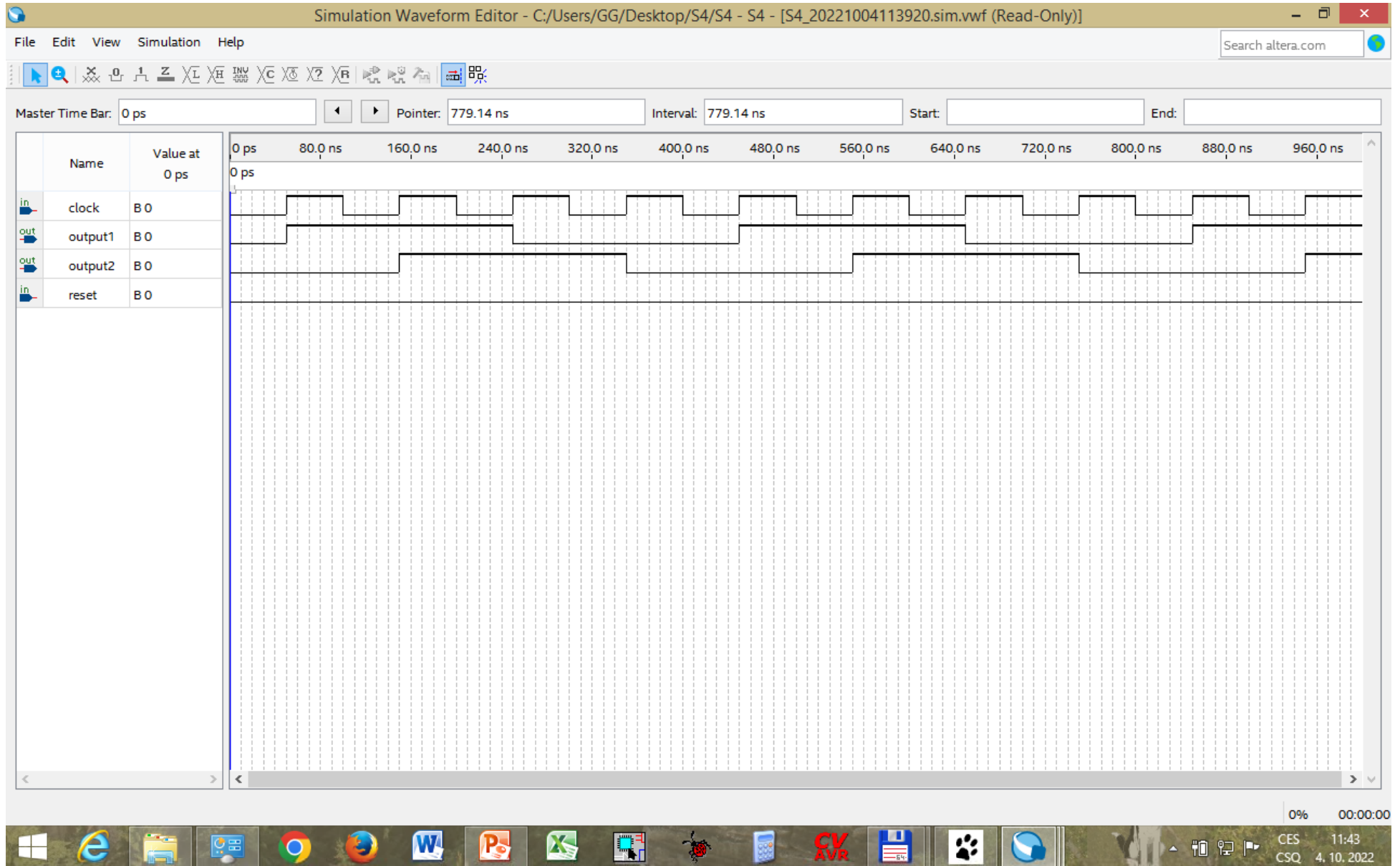
- 293000 Quartus Prime Full Compilation was successful. 0 errors, 15 warnings

System (7) Processing (120)

100% 00:01:22

CES 11:43
CSQ 4. 10. 2022

NÁVRH LSO ZE STAVOVÉHO DIAGRAMU



REALIZACE ŘADIČE PRO DYNAMICKY OVLÁDANÝ DISPLEJ - VAR.1

Quartus II - [Radic_1]

File Edit View Project Assignments Processing Tools Window Help

Project Navigator

Entity

- Compilation Hierarchy

Hierarchy Files Design Units

Status

Module	Progress %	Time

Radic_1

Input Ports

- reset
- clock
- SPUST
- STOP

Diagram description: A state machine with 7 states (S0 to S6). S0 is the initial state. Transitions: S0 to S1 on SPUST; S1 to S2; S2 to S3; S3 to S4; S4 to S5 on STOP; S5 to S6; S6 to S0. Self-loops on S0 (~SPUST) and S4 (~STOP).

	Source State	Destination State	Transition
1	S0	S0	~SPUST
2	S0	S1	SPUST
3	S1	S2	<< new transition >>
4	S2	S3	<< new transition >>
5	S3	S4	<< new transition >>
6	S4	S4	~STOP
7	S4	S5	STOP
8	S5	S6	<< new transition >>
9	S6	S0	<< new transition >>

Inputs Outputs

	Source State	Destination State	Transition
1	S0	S0	~SPUST
2	S0	S1	SPUST
3	S1	S2	<< new transition >>
4	S2	S3	<< new transition >>
5	S3	S4	<< new transition >>
6	S4	S4	~STOP
7	S4	S5	STOP
8	S5	S6	<< new transition >>
9	S6	S0	<< new transition >>

General States Inputs Outputs Transitions Actions

Messages

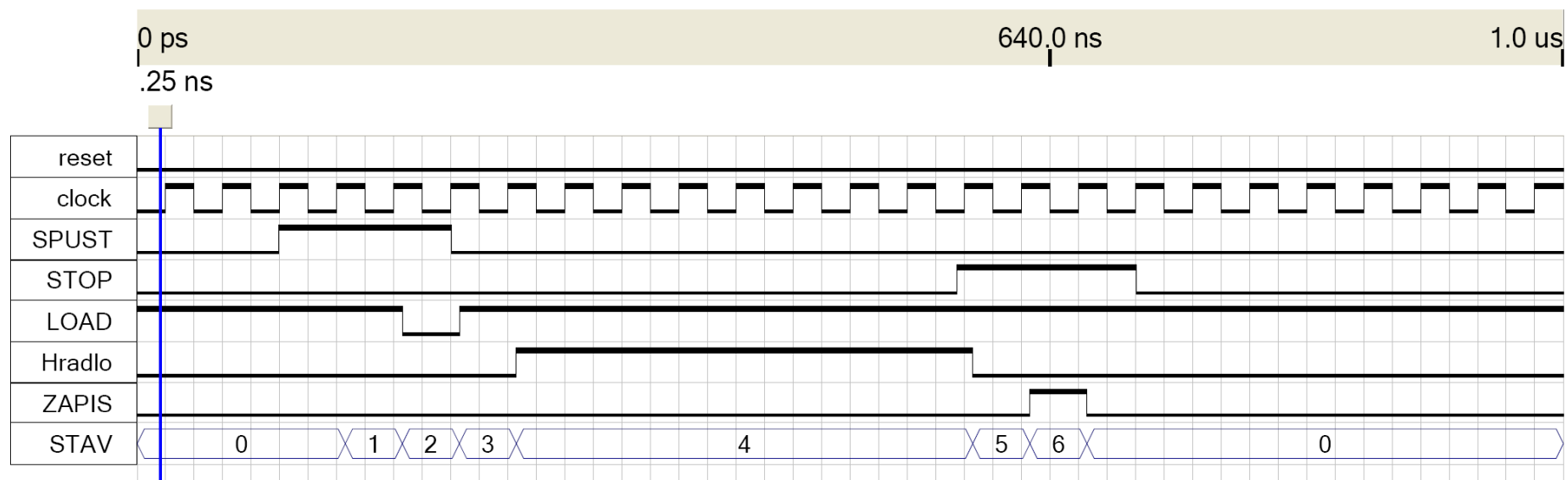
Type	Message
System	Processing

Message: Location: Locate

For Help, press F1

Start Radic_1 Cvičení-04 Quartus II - [Radic_1] Microsoft PowerPoint ... Visio Standard - [Řadi... CS NUM 12:55

REALIZACE ŘADIČE PRO DYNAMICKY OVLÁDANÝ DISPLEJ - VAR.1



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 :=
```

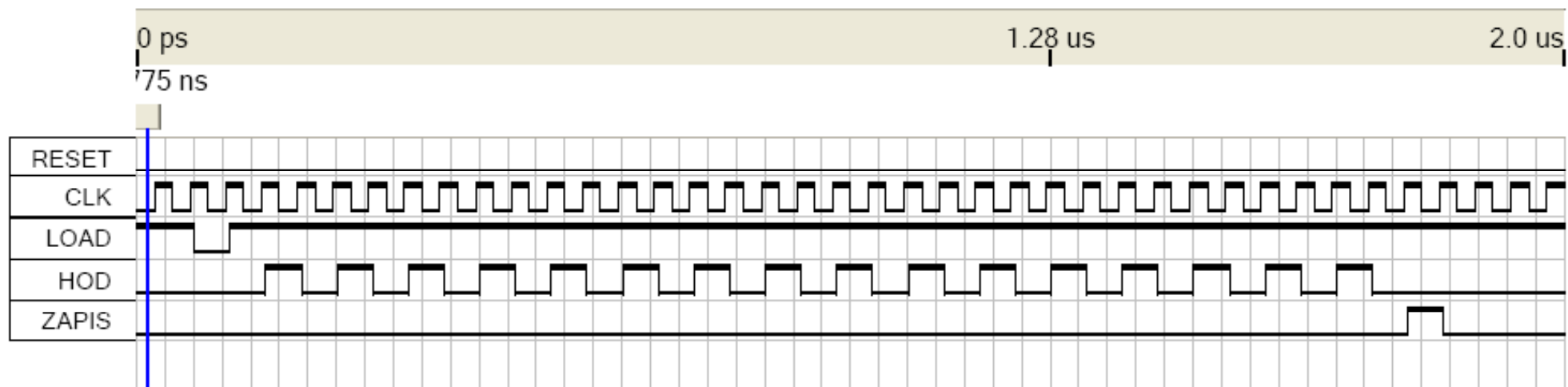
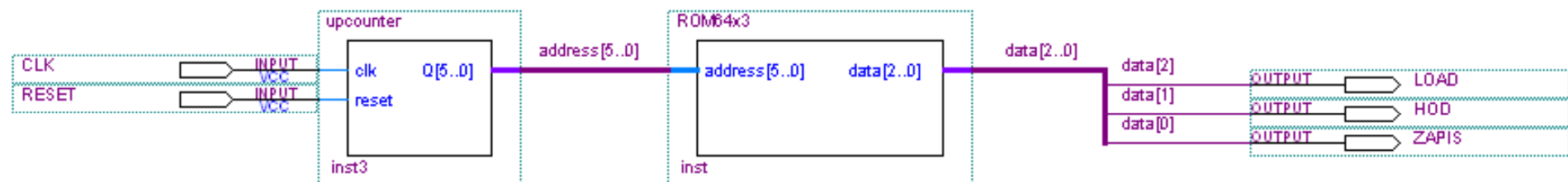
```
  ("100","100","000","100","110","100","110","100","110","100","110","100",  
   "110","100","110","100","110","100","110","100","110","100","110","100",  
   "110","100","110","100","110","100","110","100","110","100","110","100",  
   "101","100","100","100","100","100","100","100","100","100","100","100",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



REALIZACE ŘADIČE PRO DYNAMICKY OVLÁDANÝ DISPLEJ - VAR.3

