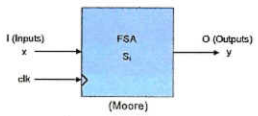
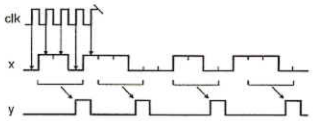
Detektor posloupnosti bitů '110' (FSA typu Moore)

Navrhněte synchronní konečný automat (FSA – Finite State Automaton), který v proudu vstupních bitů detekuje posloupnost '110'. Při detekci každé takové posloupnosti automat vyšle na výstupu impuls. Automat navrhněte s asynchronním nulováním.

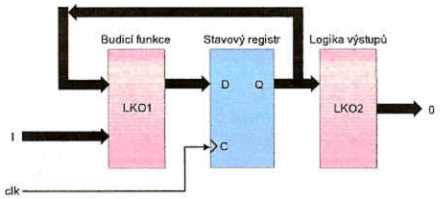


(Moore)



Detektor posloupnosti bitů '110' (FSA typu Moore)

Co máme navrhnout?



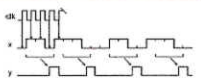
Detektor posloupnosti bitů '110' (FSA typu Moore)

Stavový diagram

I – Vstupy (Inputs)

O – Výstupy (Outputs)

S_i – i-tý stav



Tabulka přechodů

S_i	I_0	I_1
S_0	S_0	S_1
S_1	S_0	S_2
S_2	S_3	S_2
S_3	S_0	S_1

Tabulka výstupů

S_i	O_i
S_0	O_0
S_1	O_0
S_2	O_0
S_3	O_1

Detektor posloupnosti bitů '110' (FSA typu Moore)

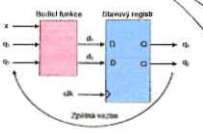
Tabulka přechodů

S_i	I_0	I_1
S_0	S_0	S_1
S_1	S_0	S_2
S_2	S_3	S_2
S_3	S_0	S_1

S_i I S_{i+1}

Kódování stavů

S_i	q_1	q_0	x	d_1	d_0	S_{i+1}
S_0	0	0	0	0	0	S_0
	0	0	1	0	1	S_1
S_1	0	1	0	0	0	S_0
	0	1	1	1	0	S_2
S_2	1	0	0	1	1	S_3
	1	0	1	1	0	S_2
S_3	1	1	0	0	0	S_0
	1	1	1	0	1	S_1



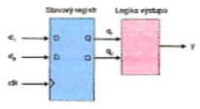
Detektor posloupnosti bitů '110' (FSA typu Moore)

Tabulka výstupů

S_i	O_i
S_0	O_0
S_1	O_1
S_2	O_2
S_3	O_3

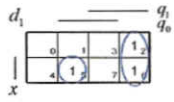
Kódování Výstupů

S_i	y
S_0	0
S_1	0
S_2	0
S_3	1

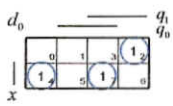


Detektor posloupnosti bitů '110' (FSA typu Moore)

Minimalizace



$$d_1 = q_1\overline{q_0} + \overline{q_1}q_0x$$

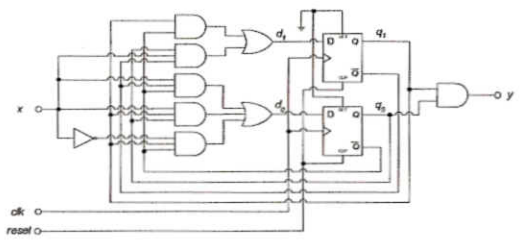


$$d_0 = \overline{q_1}\,\overline{q_0}x + q_1q_0x + q_1\overline{q_0}\,\overline{x} =$$
$$= x(\overline{q_1}\,\overline{q_0} + q_1q_0) + q_1\overline{q_0}\,\overline{x}$$

$$y = q_1q_0$$

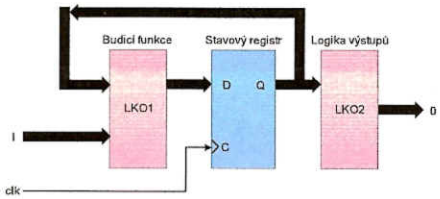
Detektor posloupnosti bitů '110' (FSA typu Moore)

Realizace



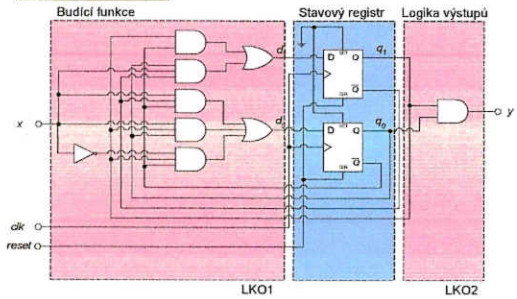
Detektor posloupnosti bitů '110' (FSA typu Moore)

Co jsme navrhli?



Detektor posloupnosti bitů '110' (FSA typu Moore)

Co jsme navrhli?



Detektor posloupnosti bitů '110' (FSA typu Moore)

"C"

```
int cBitStream3Decoder(int x_in, int reset){
  // Moore type FSA,
  // Inputs: x_in, reset, Outputs: y_out
  enum {s0,s1,s2,s3};
  static int stateReg=s0, nextState=s0, y_out;
  if(reset == TRUE){
    stateReg = s0; nextState = s0; x_in = 0;
  }
  y_out = 0;
  stateReg = nextState;
  switch(stateReg){
    case s0:
      if(x_in == 0);
      if(x_in == 1)
        nextState = s1;
      break;
    case s1:
      if(x_in == 0)
        nextState = s0;
      if(x_in == 1)
        nextState = s2;
      break;
    case s2:
      if(x_in == 0)
        nextState = s3;
      if(x_in == 1);
      break;
    case s3:
      y_out = 1;
      if(x_in == 0)
        nextState = s0;
      if(x_in == 1)
        nextState = s1;
      break;
    default: // Error section
      y_out = 0;
      nextState = s0;
  }// switch() END
  return(y_out);
}// cBitStream3Decoder() END
```


Detektor posloupnosti bitů '110' (FSA typu Moore)

Java

```
class JBitStream3Decoder {
  final int s0 = 0, s1 = 1, s2 = 2, s3 = 3;
  int stateReg = s0, nextState = s0;
  int yOut = 0;

  public JBitStream3Decoder() {// Constructor
    // empty
  }

  void setFsaReset (boolean reset){
    stateReg = s0;
    nextState = s0;
    yOut = 0;
  }

  int jBitStream3Decoder(int xIn) {
    // Moore type FSA
    // Inputs: xIn, reset, Outputs: y_out
    yOut = 0;
    stateReg = nextState;
    switch (stateReg) {
      case s0:
        if (xIn == 0);
        if (xIn == 1)
          nextState = s1;
        break;
      case s1:
        if (xIn == 0)
          nextState = s0;
        if (xIn == 1)
          nextState = s2;
        break;
      case s2:
        if (xIn == 0)
          nextState = s3;
        if (xIn == 1),
        break;
      case s3:
        yOut = 1;
        if (xIn == 0)
          nextState = s0;
        if (xIn == 1)
          nextState = s1;
        break;
      default: // Error section
        yOut = 0;
        nextState = s0;
    }// switch() END
    return (yOut);
  }// jBitStream3Decoder() END

}// JBitStream3Decoder class END
```

Detektor posloupnosti bitů '110' (FSA typu Moore)

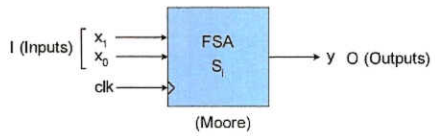
VHDL

```
entity vFlitStream3Decoder is
  Port ( clk : in  STD_LOGIC;
    x_in : in  STD_LOGIC;
    y_out : out  STD_LOGIC;
    reset : in  STD_LOGIC;
    q : out std_logic_vector(1 downto 0)
    );
end vFlitStream3Decoder;

architecture Behavioral of vFlitStream3Decoder is
  type states is (s1,s2,s3,s4);
  signal stateReg, nextState: states := s1;
begin -- FSA - Finite State Machine
  process(clk, reset)
    begin
      if reset = '1' then
        stateReg <= s1;
      elsif
        clk'event and clk = '1' then
        stateReg <= nextState;
      end if;
  end process;

process(stateReg, x_in) -- State diagram
    definition
    begin
      nextState <= stateReg;
        case stateReg is
          when s1 =>
            if x_in = '1' then
              nextState <= s2;
            end if;
          when s2 =>
            if x_in = '0' then
              nextState <= s1;
            elsif x_in = '1' then
              nextState <= s3;
            end if;
          when s3 =>
            if x_in = '1' then
              nextState <= s3;
            elsif x_in = '0' then
              nextState <= s4;
            end if;
          when s4 =>
            if x_in = '1' then
              nextState <= s2;
            elsif x_in = '0' then
              nextState <= s1;
            end if;
          when others =>
            nextState <= stateReg;
        end case;
      end process;

      process(stateReg) -- Output
    function
        begin
          case stateReg is
            when s1 =>
              y_out <= '0';
            when s2 =>
              y_out <= '0';
            when s3 =>
              y_out <= '0';
            when s4 =>
              y_out <= '1';
            when others => null;
          end case;
        end process;
end Behavioral;
```


Detektor posloupnosti bitů '110' (FSA typu Moore)

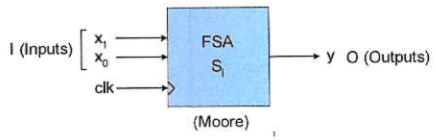
Příklad návrhu

Navrhněte synchronní konečný automat (FSA – Finite State Automaton), který v proudu vstupních bitů detekuje posloupnost '110'. Detekci každé takové posloupnosti automat indikuje na výstupu blikáním LED diody. V ostatních stavech automatu LED dioda svítí nepřerušovaně. Náběžná hrana x_i indikuje na vstupu '1', náběžná hrana x_0 pak '0'. Automat realizujte programovými prostředky a využitím hardwareové podpory. Použijte jednohladinový systém přerušení a časovač. Vstupy x_1 a x_0 se zadávají tlačítky s mechanickým kontaktem a odskoky při změně hodnoty. Program automatu musí odskoky filtrovat, sestavte a použijte algoritmus pro „debounce circuit".

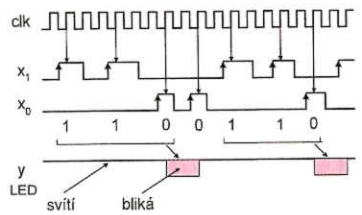


(Moore)

Detektor posloupnosti bitů '110' (FSA typu Moore)

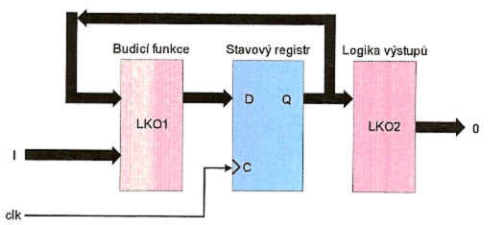


(Moore)



Detektor posloupnosti bitů '110' (FSA typu Moore)

Co máme navrhnout?



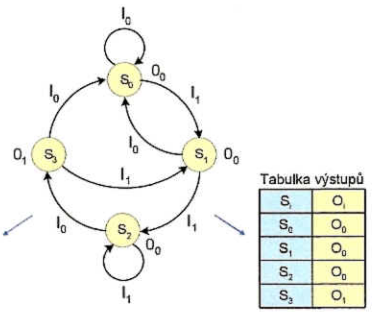
Detektor posloupnosti bitů '110' (FSA typu Moore)

Stavový diagram

I – Vstupy (Inputs)

O – Výstupy (Outputs)

S_i – i-tý stav



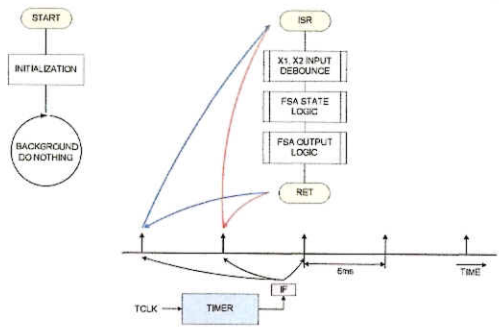
Tabulka přechodů

S_i	I_0	I_1
S_0	S_0	S_1
S_1	S_0	S_2
S_2	S_3	S_2
S_3	S_0	S_1

Tabulka výstupů

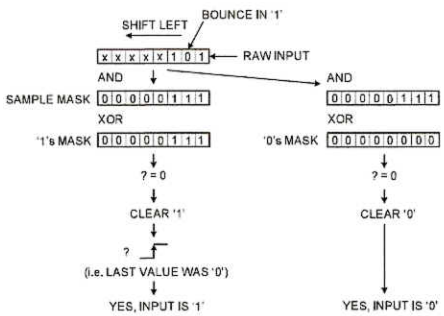
S_i	O_i
S_0	O_0
S_1	O_0
S_2	O_0
S_3	O_1

Dekompozice problému - 1

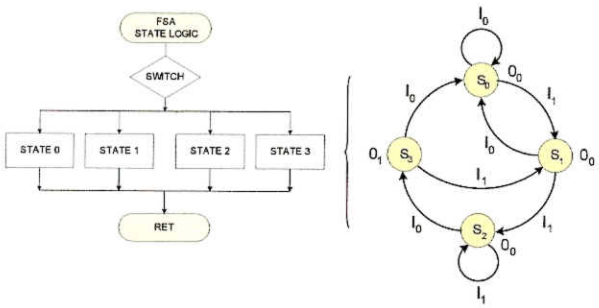


Dekompozice problému - 2

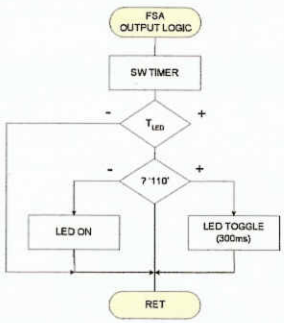
DEBOUNCE ALGORITHM PRINCIPLE



Dekompozice problému - 3



Dekompozice problému - 4

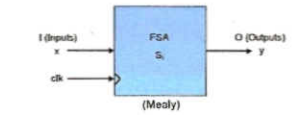


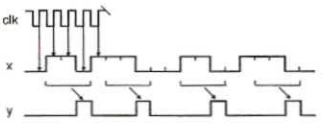
Detektor '110' – Hlavička programu

```
; File:          BSD3_110.asm
; Date:          04.04.2010
; Version:       1.00
; HW:            Explorer pic18F87J11
; IDE:           MPLAB 7.50
; Author:        J.Z.;  Institute: CTU in Prague, EEF Dept.13114
; Function: Bitstream '110' detector, Moore type FSA
; Pushbutton S1=input'1', pushbutton S2=input'0'
; Program detects leading edge of S1,S2 pushbuttons
; FSA outputs:
; LED1,0 = FSA state_register
; LED7 = ON - no '110' bitstream detected
; LED7 = TOGGLE (approx 300ms] - '110' bitstream detected
; Note: Pushbuttons S1,S2 are debounced (3 conscutive equal bits required
; to be correct value detected)
; Debounce S1,S2 pushbuttons routines detect leading edge only
```

Detektor posloupnosti bitů '110' (FSA typu Mealy)

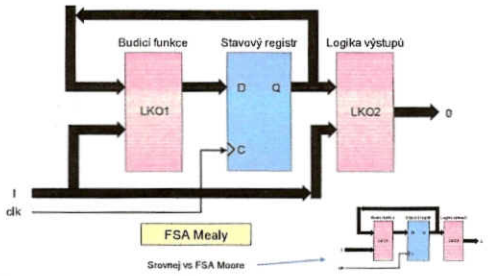
Navrhněte synchronní konečný automat (FSA – Finite State Automaton), který v proudu vstupních bitů detekuje posloupnost '110'. Při detekci každé takové posloupnosti automat vyšle na výstupu impuls. Automat navrhněte s asynchronním nulováním.





Detektor posloupnosti bitů '110' (FSA typu Mealy)

Co máme navrhnout?



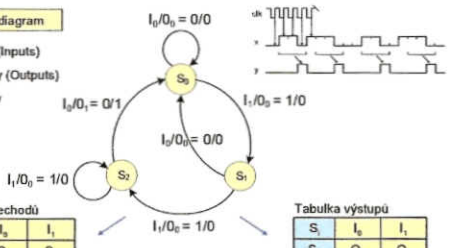
Detektor posloupnosti bitů '110' (FSA typu Mealy)

Stavový diagram

I – Vstupy (Inputs)

O – Výstupy (Outputs)

S_i – i-tý stav



Tabulka přechodů

S_i	I_0	I_1
S_0	S_0	S_1
S_1	S_0	S_2
S_2	S_0	S_2

Tabulka výstupů

S_i	I_0	I_1
S_0	O_0	O_2
S_1	O_0	O_0
S_2	O_1	O_0

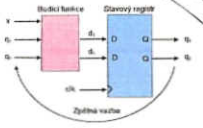
Detektor posloupnosti bitů '110' (FSA typu Mealy)

Tabulka přechodů

S_i	I_0	I_1
S_0	S_0	S_1
S_1	S_0	S_2
S_2	S_0	S_2

Kódování stavů

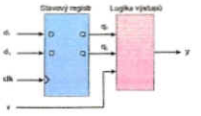
S_i	q_1	q_0	x	d_1	d_0	S_{i+1}
S_0	0	0	0	0	0	S_0
	0	0	1	0	1	S_1
S_1	0	1	0	0	0	S_0
	0	1	1	1	0	S_2
S_2	1	0	0	0	0	S_0
	1	0	1	1	0	S_2



Detektor posloupnosti bitů '110' (FSA typu Mealy)

Tabulka výstupů

S_i	I_0	I_1
S_0	O_0	O_0
S_1	O_0	O_0
S_2	O_1	O_0

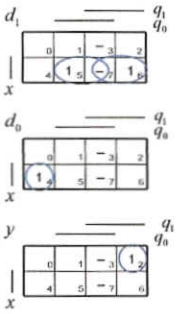


Kódování výstupů

S_i	q_1	q_0	x	y
S_0	0	0	0	0
	0	0	1	0
S_1	0	1	0	0
	0	1	1	0
S_2	1	0	0	1
	1	0	1	0

Detektor posloupnosti bitů '110' (FSA typu Mealy)

Minimalizace



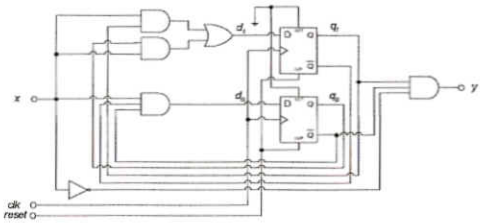
$$d_1 = q_1 x + q_0 x$$

$$d_0 = \overline{q_1}\,\overline{q_0}\,x$$

$$y = q_1\,\overline{q_0}\,\overline{x}$$

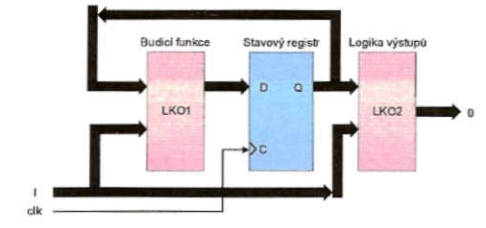
Detektor posloupnosti bitů '110' (FSA typu Mealy)

Realizace



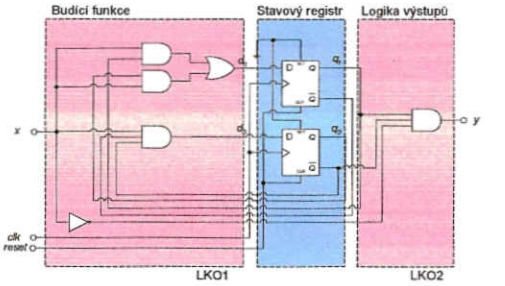
Detektor posloupnosti bitů '110' (FSA typu Mealy)

Co jsme navrhli?



Detektor posloupnosti bitů '110' (FSA typu Mealy)

Co jsme navrhli?



Detektor '110' – Konfigurace a konstanty

```
#define  DBG_MODE      ; Conditional translation (simulator dbg)

#include P18F87J11.inc
   CONFIG  FOSC=HS
   CONFIG  WDTEN=OFF
   CONFIG  XINST=OFF

#define F                1
#define BIT0             0
#define TIMER0_FOSC_PRESC8_16bit  b'00000010'
#define TIMER0_FOSC_PRESC64_8bit  b'01000101' ; 6,6ms, FOSC=10MHz
```


Detektor '110' – Proměnné a konstanty

```
   UDATA_ACS
; Program variables
; Save context registers
wreg_tmp:       res     1
status_tmp:     res     1
bsr_tmp:        res     1
reg:            res     1
; FSA Moore variables and constants
#define STATE_0                  0
#define STATE_1                  1
#define STATE_2                  2
#define STATE_3                  3
#define FSA_INPUT_BIT_VALUE      0
#define FSA_OUTPUT_LED_ON_TIME   45     ; t=6.6ms * 45 = 300ms
#define FSA_OUTPUT_LED_OFF_TIME  25     ; 170ms
```


Detektor '110' – Proměnné

```
state_register          res     1
next_state              res     1
fsa_input               res     1
fsa_output              res     1
fsa_output_timer        res     1

; Pushbutton s1,s2 debounce and leading edge detect variables
portb_dbg:              res     1
porta_dbg:              res     1
pb_s1_debounce:         res     1
pb_s1_status:           res     1
pb_s2_debounce:         res     1
pb_s2_status:           res     1
```


MOORE

Detektor '110' – Konstanty (vstup FSA)

```
; Pushbutton s1,s2 debounce and leading edge detection constants
#define  PB_ON           0               ;Pb sx on status bit position
#define  PB_OFF          1               ;Pb sx off status bit position
#define  DEBOUNCE_MASK   b'00000111'     ; 3 equal bits required
#define  PB_xTIMES_ON    b'00000111'
#define  PB_xTIMES_OFF   b'00000000'
```


Detektor '110' – Začátek programu a tab.vektoru přerušení

```
   code
   ORG          0x00000

Reset_vector:                   ; Program start
   goto  Start
   ORG          0x00008

Interrupt_vector:               ; Interrupt vector table (one item here)
   goto  Timer0_int
```


Detektor '110' – Inicializace

```
   ORG          0x00050
Start:                          ; Background Start
   clrf     reg                 ; reg = 0
   clrf     WREG                ; WREG = 0
   movwf    TRISD               ; PORTD -> all bits outputs
   movwf    PORTD               ; PORTD = 00h
   bsf      TRISB,BIT0          ; PORTB.bit0=input, push button S1
   bsf      TRISA,RA5
   bsf      WDTCON,ADSHR
   bsf      ANCON0,PCFG4        ; RA5 - digital input
   bcf      WDTCON,ADSHR
   call     Pushbutton_s1_init  ; Initialize s1 debounce routine
   call     Pushbutton_s2_init  ; Initialize s2 debounce routine
   call     Bitstream_detector_init
   call     Bitstream_detector_output_init
```


Detektor '110' – Inicializace a pozadí (prázdné)

```
   movlw    TIMER0_FOSC_PRESC64_8bit
   movwf    T0CON               ; Configure Timer0
   bsf      T0CON,TMR0ON        ; Timer0 ON
   ;
   bsf      INTCON,TMR0IE       ; Timer0 interrupt enable
   bsf      INTCON,GIE          ; Non priority mode (IPEN=0 by default)
Loop:
   nop                          ; Do nothing, background
   bra      Loop
;
; Background END
```


Detektor '110' – Inicializace (vstupní filtr tlačítek S1, S2)

```
; Init debounce pushbutton S1 (RB0)
Pushbutton_s1_init:
   clrf   pb_s1_debounce
   clrf   pb_s1_status
   return
; Pushbutton_s1_init END

; Init debounce pushbutton S2 (RA5)
Pushbutton_s2_init:
   clrf   pb_s2_debounce
   clrf   pb_s2_status
   return
; Pushbutton_s1_init END
```


Detektor '110' – Vstupní filtr tlačítka S1

```
Pushbutton_s1:                  ; Debounce pushbutton S1 (RB0)
   movlw  DEBOUNCE_MASK
   andwf  pb_s1_debounce,F
   rlncf  pb_s1_debounce,F      ; Rotate one bit left
#ifdef DBG_MODE
   btfss  portb_dbg,RB0         ; dbg ? Push button s1 pressed (0 -> on)
#else
   btfss  PORTB,RB0             ; ? Push button s1 pressed (0 -> on)
#endif
Pushbutton_s1_read_as_on:
   bsf    pb_s1_debounce,BIT0   ; Yes, insert '1' to LSB
Pb_s1_test_off:
   movf   pb_s1_debounce,W
   andlw  DEBOUNCE_MASK
   xorlw  PB_xTIMES_OFF
   bnz    Pb_s1_test_on
```

Detektor '110' – Vstupní filtr tlačítka S1

```
Pb_s1_set_off:                  ; Ok, s1 debounced, read off
    bcf     pb_s1_status,PB_ON  ; Clear pushbutton s1 on
    bsf     pb_s1_status,PB_OFF ; Set pushbutton s1 off
    bra     Pb_s1_exit
Pb_s1_test_on:
    movf    pb_s1_debounce, W
    andlw   DEBOUNCE_MASK
    xorlw   PB_xTIMES_ON
    bnz     Pb_s1_exit          ; s1 bounces yet
Pb_s1_detect_leading_edge:
    btfss   pb_s1_status,PB_OFF ; Detect leading edge
    bra     Pb_s1_exit
Pb_s1_set_pb_on:
    bcf     pb_s1_status,PB_OFF
    bsf     pb_s1_status,PB_ON  ; Set pushbutton s1 on
Pb_s1_exit: return              ; Pushbutton_s1 END
```


Detektor '110' – Vstupní filtr tlačítka S2

```
Pushbutton_s2:                  ; Debounce pushbutton S2 (RA5)
    movlw   DEBOUNCE_MASK
    andwf   pb_s2_debounce,F
    rlncf   pb_s2_debounce,F    ; Rotate one bit left
#ifdef DBG_MODE
    btfss   porta_dbg,RA0       ; dbg ?Push button s2 pressed (0->on)
#else
    btfss   PORTA,RA5           ; ? Push button s1 pressed (0->on)
#endif
Pushbutton_s2_read_as_on:
    bsf     pb_s2_debounce,BIT0 ; Yes, insert '1' to LSB
Pb_s2_test_off:
    movf    pb_s2_debounce,W
    andlw   DEBOUNCE_MASK
    xorlw   PB_xTIMES_OFF
    bnz     Pb_s2_test_on
```


Detektor '110' – Vstupní filtr tlačítka S2

```
Pb_s2_set_off:                  ; Ok, s2 debounced, read off
    bcf     pb_s2_status,PB_ON  ; Clear pushbutton s2 on
    bsf     pb_s2_status,PB_OFF ; Set pushbutton s2 off
    bra     Pb_s2_exit
Pb_s2_test_on:
    movf    pb_s2_debounce,W
    andlw   DEBOUNCE_MASK
    xorlw   PB_xTIMES_ON
    bnz     Pb_s2_exit          ; s2 bounces yet
Pb_s2_detect_leading_edge:
    btfss   pb_s2_status,PB_OFF ; Detect leading edge
    bra     Pb_s2_exit
Pb_s2_set_pb_on:
    bcf     pb_s2_status,PB_OFF
    bsf     pb_s2_status,PB_ON  ; Set pushbutton s2 on
Pb_s2_exit: return              ; Pushbutton_s2 END
```


MOORE

Detektor '110' – Inicializace (FSA)

```
; Bitstream detector initialization
Bitstream_detector_init:
    movlw   STATE_0
    movwf   state_register
    movwf   next_state
    bcf     fsa_input,FSA_INPUT_BIT_VALUE
    return
; Bitstream_detector_init END
```


Detektor '110' – Vlastní konečný automat FSA

```
; Bitstream detector '110', FSA Moore
Bitstream_detector:
    ; Test FSA input value ('0' or '1' or none)
    btfsc   pb_s1_status, PB_ON             ; S1 -> input '1'
    bra     Bd_input_1
    btfss   pb_s2_status, PB_ON             ; S2 -> input '0'
    bra     Bd_exit                         ; No input bit
Bd_input_0:
    bcf     pb_s2_status,PB_ON              ; Consume input '0'
    bcf     fsa_input,FSA_INPUT_BIT_VALUE   ; Input = '0'
    bra     Bd_state_switch
Bd_input_1:
    bcf     pb_s1_status,PB_ON              ; Consume input '1'
    bsf     fsa_input,FSA_INPUT_BIT_VALUE   ; Input = '1'
```


Detektor '110' – Vlastní konečný automat FSA

```
; In Java .... switch (state_register){ }
Bd_state_switch:
    movf    state_register,W
    sublw   STATE_0
    bz      Bd_state_0
    movf    state_register,W
    sublw   STATE_1
    bz      Bd_state_1
    movf    state_register,W
    sublw   STATE_2
    bz      Bd_state_2
    bra     Bd_state_3              ; There is no more states
    ;
    movff   state_register, next_state
```


Detektor '110' – Vlastní konečný automat FSA

```
Bd_state_0: ; FSA state diagram logic (See state diagram)
    btfss   fsa_input, FSA_INPUT_BIT_VALUE  ; ? '1'
    bra     Bd_set_next_state               ; No, no change
    movlw   STATE_1
    movwf   next_state                      ; Goto STATE_1
    bra     Bd_set_next_state
Bd_state_1:
    btfss   fsa_input, FSA_INPUT_BIT_VALUE  ; ? '1'
    bra     Bd_state_1_1
    movlw   STATE_2                         ; Yes
    movwf   next_state
    bra     Bd_set_next_state
Bd_state_1_1:
    movlw   STATE_0                 ; No
    movwf   next_state
    bra     Bd_set_next_state
```


Detektor '110' – Vlastní konečný automat FSA

```
Bd_state_2:
    btfsc   fsa_input,FSA_INPUT_BIT_VALUE ; ? '0'
    bra     Bd_set_next_state             ; No, no change
    movlw   STATE_3                       ; Yes
    movwf   next_state
    bra     Bd_set_next_state
Bd_state_3:
    btfss   fsa_input,FSA_INPUT_BIT_VALUE ; ? '1'
    bra     Bd_state_3_1
    movlw   STATE_1                       ; Yes
    movwf   next_state
    bra     Bd_set_next_state
Bd_state_3_1:
    movlw   STATE_0                       ; No
    movwf   next_state
    bra     Bd_set_next_state
```


Detektor '110' – Vlastní konečný automat FSA

```
Bd_set_next_state:
    movff   next_state, state_register
Bd_exit:
    return
; Bitstream_detector END
```

Detektor '110' – Inicializace (FSA – logika výstupů)

```
; Bitstream detector output initializtion
;
Bitstream_detector_output_init:
    movlw  FSA_OUTPUT_LED_ON_TIME
    movwf  fsa_output_timer
    clrf   fsa_output
    return
;
; Bitstream_detector END
```


Detektor '110' – FSA - logika výstupů

```
; STATEs 0,1,2 -> LED7 ON
; STATE 3 -> sequence detected -> LED7 toggle, T= approx 400ms
Bitstream_detector_output:
    movlw  STATE_3
    cpfseq state_register
    bra    Bdo_nothing_detected
Bdo_bitstream_detected:
    btfsc  LATD,RD7                ; Send state_register, no delay
    bsf    WREG,RD7
    movff  WREG,LATD
Bdo_testif_bit_toggle:
    decfsz fsa_output_timer
    bra    Bdo_exit
Bdo_toggle_led:
    btfss  LATD,RD7
    bra    Bdo_set_ledtime_on
```


Detektor '110' – FSA - logika výstupů

```
Bdo_set_ledtime_off:
    movlw  FSA_OUTPUT_LED_OFF_TIME
    movwf  fsa_output_timer
    bra    Bdo_send_output
Bdo_set_ledtime_on:
    movlw  FSA_OUTPUT_LED_ON_TIME
    movwf  fsa_output_timer
Bdo_send_output:
    movff  state_register, WREG
Bdo_preset_led_on:
    bsf    WREG,RD7
    btfsc  LATD,RD7
```


Detektor '110' – FSA - logika výstupů

```
Bdo_set_led_off:
    bcf    WREG,RD7
Bdo_send_new_output:
    movff  WREG, LATD
    bra    Bdo_exit
;
Bdo_nothing_detected:
    movff  state_register, WREG
    bsf    WREG, RD7
    movff  WREG, LATD
Bdo_exit:
    return
;
; Bitstream_detector END
```


Detektor '110' - Timer0 ISR (Interrupt Service Routine)

```
Timer0_int:  ; Timer0 ISR (Interrupt Service Routine), T = 6.6ms
             ; Save context (example, background empty-no context)
    movff  STATUS, status_tmp
    movff  WREG, wreg_tmp
    movff  BSR, bsr_tmp
    ;
    bcf    INTCON, TMR0IF          ; Clear intrerrupt flag
    ;
    call   Pushbutton_s1           ; Test s1 on (input '1')
    call   Pushbutton_s2           ; Test s2 on (input '0')
    call   Bitstream_detector      ; FSA state logic
    call   Bitstream_detector_output ; FSA output logic
    ;
    movff  bsr_tmp, BSR            ; Restore context
    movff  wreg_tmp, WREG
    movff  status_tmp, STATUS
    retfie         ; Return from interrupt, reenable interrupt
    END
```


MOORE