EE-334
Digital System Design

Discussion for Lab 3
KeyLock in VHDL

Andreas Burg, A. Kristensen

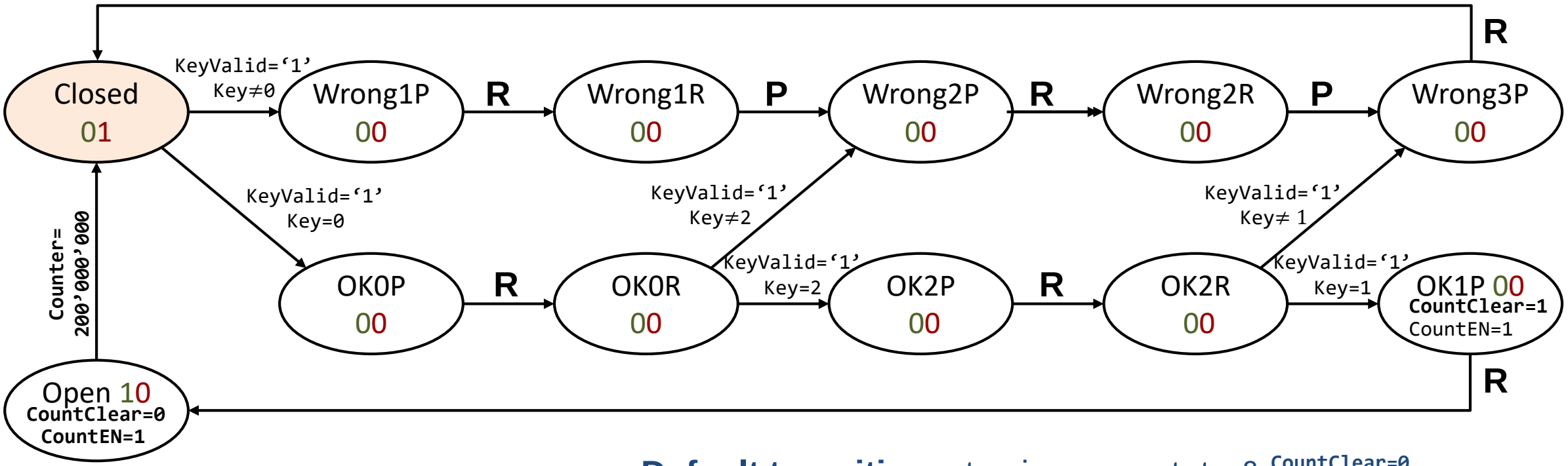
FSM for Key Lock with Moore FSM

```
KeyValid ∈ {‘0’, ‘1’}
Key      ∈ {0,1,2,3}

CountEN    ∈ {‘0’, ‘1’}
CountClear ∈ {‘0’, ‘1’}
```

```
GLED ∈ {‘0’, ‘1’}
RLED ∈ {‘0’, ‘1’}
Counter
```

	KeyValid	Key
P	‘1’	-
R	‘0’	-



Default transition: stay in same state & `CountClear=0` `CountEN=0`

Key Lock Top Level

- **Top level** with some basic signal translations, but generally not much functionality: **optional**, but good practice (often contains also PADs and circuits for clocking)
  - Provide FSM input signals `KEYxD` and `KeyVALIDxS` in the appropriate format
  - Instantiate the actual design unit

```
  -- Signals
  SIGNAL KeyVALIDxS : std_logic;
  SIGNAL KEYxD : unsigned(2-1 DOWNTO 0);

begin

  -- Translate one-hot keys into a Key code (0-3)
  KEYxD <= to_unsigned(0,2) WHEN Push0xSI = '1' ELSE
           to_unsigned(1,2) WHEN Push1xSI = '1' ELSE
           to_unsigned(2,2) WHEN Push2xSI = '1' ELSE
           to_unsigned(3,2) WHEN Push3xSI = '1' ELSE
           "--";

  -- KeyVALID should only be asserted if ONE key is pressed
  KeyVALIDxS <= '1' WHEN Push0xSI='1' AND Push1xSI='0' and Push2xSI='0' AND Push3xSI='0' ELSE
                '1' WHEN Push0xSI='0' AND Push1xSI='1' and Push2xSI='0' AND Push3xSI='0' ELSE
                '1' WHEN Push0xSI='0' AND Push1xSI='0' and Push2xSI='1' AND Push3xSI='0' ELSE
                '1' WHEN Push0xSI='0' AND Push1xSI='0' and Push2xSI='0' AND Push3xSI='1' ELSE
                '0';
```

```
  -- Instantiate FSM
  i_key_lock_timed: key_lock_timed
    PORT MAP (
      CLKxCI       => CLKxCI,
      RSTxR        => RSTxRI,
      KeyVALIDxSI  => KeyVALIDxS,
      KEYxDI       => KEYxD,
      RLEDxSO      => RLEDxSO,
      GLEDxSO      => GLEDxSO);
end rtl;
```

Timed Keylock Component

- **Signals, Types, Constants:**
  - FSM state with enumerated type
  - Signals for counter and FSM
  - Constants
- **Counter**: concurrent assignment & clocked process

```
ARCHITECTURE rtl OF key_lock_timed IS
  -- Declare Types for FSM
  TYPE KeyLockFSM_Type IS (Closed, Wrong1P, Wrong1R,
                           Wrong2P, Wrong2R, Wrong3P,
                           OK0P, OK0R, OK2P, OK2R, OK1P,
                           Opened);
  -------------------------------------
  -- Declares signals
  -------------------------------------
  -- FSM
  SIGNAL STATExDN, STATExDP : KeyLockFSM_Type;
  -- Counters
  CONSTANT DelCntMAX : integer := 250000000; -- 2 seconds
  SIGNAL CNTxDN, CNTxDP : unsigned(30-1 DOWNTO 0);
  SIGNAL CountCLRxS : std_logic;
  SIGNAL CountENxS : std_logic;
```

```
-- COUNTER
CNTxDN <= (OTHERS => '0') WHEN CountCLRxS = '1' ELSE
          CNTxDP + 1 WHEN CountENxS = '1' ELSE
          CNTxDP;
```

```
p_Cnt: PROCESS (CLKxCI, RSTxR) IS
BEGIN  -- PROCESS p_Cnt
  IF RSTxR = '1' THEN                       -- asynchronous reset (active high)
    CNTxDP <= (OTHERS => '0');
  ELSIF CLKxCI'event AND CLKxCI = '1' THEN  -- rising clock edge
    CNTxDP <= CNTxDN;
  END IF;
END PROCESS p_Cnt;
```

- **Finite State Machine**: one combinational & one clocked process (see next slide)

Timed Keylock FSM Comb. & Clocked Process

```
-- FSM
-- FSM single Combinational Process
p_FSMComb: PROCESS (CNTxDP, KEYxDI, KeyVALIDxSI, STATExDP) IS
BEGIN  -- PROCESS p_FSMComb
  STATExDN <= STATExDP;
  GLEDxSO <= '0';          -- default: LED off
  RLEDxSO <= '0';          -- default: LED off
  CountCLRxS <= '0';
  CountENxS <= '0';
```

Default assignments

```
  CASE STATExDP IS
    WHEN Closed =>
      RLEDxSO <= '1';
      IF KeyVALIDxSI = '1' THEN
        IF KEYxDI = 0 THEN
          STATExDN <= OK0P;
        ELSE
          STATExDN <= Wrong1P;
        END IF;
      END IF;

    -- States where code was wrong
    WHEN Wrong1P =>
      IF KeyVALIDxSI = '0' THEN
        STATExDN <= Wrong1R;
      END IF;
    WHEN Wrong1R =>
      IF KeyVALIDxSI = '1' THEN
        STATExDN <= Wrong2P;
      END IF;
    WHEN Wrong2P =>
      IF KeyVALIDxSI = '0' THEN
        STATExDN <= Wrong2R;
      END IF;
    WHEN Wrong2R =>
      IF KeyVALIDxSI = '1' THEN
        STATExDN <= Wrong3P;
      END IF;
    WHEN Wrong3P =>
      IF KeyVALIDxSI = '0' THEN
        STATExDN <= Closed;
      END IF;

    -- States where code is still OK
    WHEN OK0P =>
      IF KeyVALIDxSI = '0' THEN
        STATExDN <= OK0R;
      END IF;
    WHEN OK0R =>
      IF KeyVALIDxSI = '1' THEN
        IF KEYxDI = 2 THEN
          STATExDN <= OK2P;
        ELSE
          STATExDN <= Wrong2P;
        END IF;
      END IF;
    WHEN OK2P =>
      IF KeyVALIDxSI = '0' THEN
        STATExDN <= OK2R;
      END IF;
    WHEN OK2R =>
      IF KeyVALIDxSI = '1' THEN
        IF KEYxDI = 1 THEN
          STATExDN <= OK1P;
        ELSE
          STATExDN <= Wrong3P;
        END IF;
      END IF;
    WHEN OK1P =>
      IF KeyVALIDxSI = '0' THEN
        STATExDN <= Opened;
        CountCLRxS <= '1';
        CountENxS <= '1';
      END IF;

    WHEN Opened =>
      GLEDxSO <= '1';
      CountCLRxS <= '0';
      CountENxS <= '1';
      IF CNTxDP = DelCntMAX THEN
        STATExDN <= Closed;
      END IF;

    WHEN OTHERS => NULL;
  END CASE;
END PROCESS p_FSMComb;
```

```
-- FSM Sequential Process
p_FSMSeq: PROCESS (CLKxCI, RSTxR) IS
BEGIN  -- PROCESS p_FSMSeq
  IF RSTxR = '1' THEN
    STATExDP <= Closed;
  ELSIF CLKxCI'event AND CLKxCI = '1' THEN
    STATExDP <= STATExDN;
  END IF;
END PROCESS p_FSMSeq;
```

Common Mistakes & Potential for Improvement

```
signal next_state, present_state : states;
signal CNTxP,CNTxN: unsigned(28-1 downto 0):= (others=>'0');

begin



FSM : process (ClKxCI,present_state,Key) is
begin
present_state <= next_state;
GLED <= '0';
RLED <= '1';

if rising_edge(ClKxCI) then
Case present_state is

when Closed =>
    if KeyValid = '1' and Key /= std_logic_vector(to_unsigned(1,4)) then
        next_state <= Wrong1P;
        GLED <= '0';
        RLED <= '0';

    elsif KeyValid = '1' and Key = std_logic_vector(to_unsigned(1,4)) then
        next_state <= OK0P;
        GLED <= '0';
        RLED <= '0';

    end if;
--Wrong path
when Wrong1P =>
    if KeyValid='0' then
    next_state <=Wrong1R;
```

Common Mistakes & Potential for Improvement

Real Issues

```
    -- Counter --

PROCESS(CLKxCI, count_cl, count_en) IS
BEGIN
    IF count_cl = '1' THEN
        count <= (OTHERS => '0');
    ELSIF CLKxCI'EVENT AND CLKxCI = '1' AND count_en = '1' THEN
        count <= count + 1;
    END IF;
END PROCESS;


cnt : PROCESS (CLKxCI, CNT_CLEARxSO, CNT_ENxSO) IS
BEGIN --Threshold
    IF CNT_CLEARxSO = '1' THEN
        counter <= (OTHERS => '0');
    ELSIF CLKxCI 'EVENT AND CLKxCI = '1' THEN
        IF CNT_ENxSO = '1' THEN
            counter <= STD_LOGIC_VECTOR(UNSIGNED(counter) + 1);
        END IF;
    END IF;
END PROCESS cnt;
```

Could be more elegant, but OK

```
--Counter
CNTxDN <= CNTxDP+1;
p_cnt: PROCESS(CLKxCI,RSTxRI) IS
BEGIN
    IF RSTxRI = '1' THEN
        CNTxDP <= (OTHERS=>'0');
    ELSIF CLKxCI'EVENT AND CLKxCI = '1' THEN
        IF CNT_CLRxS = '1' THEN
            CNTxDP <= (OTHERS=>'0');
        ELSIF CNT_ENxS = '1' THEN
            CNTxDP <= CNTxDN;
        END IF;
    END IF;
END PROCESS p_cnt;
```