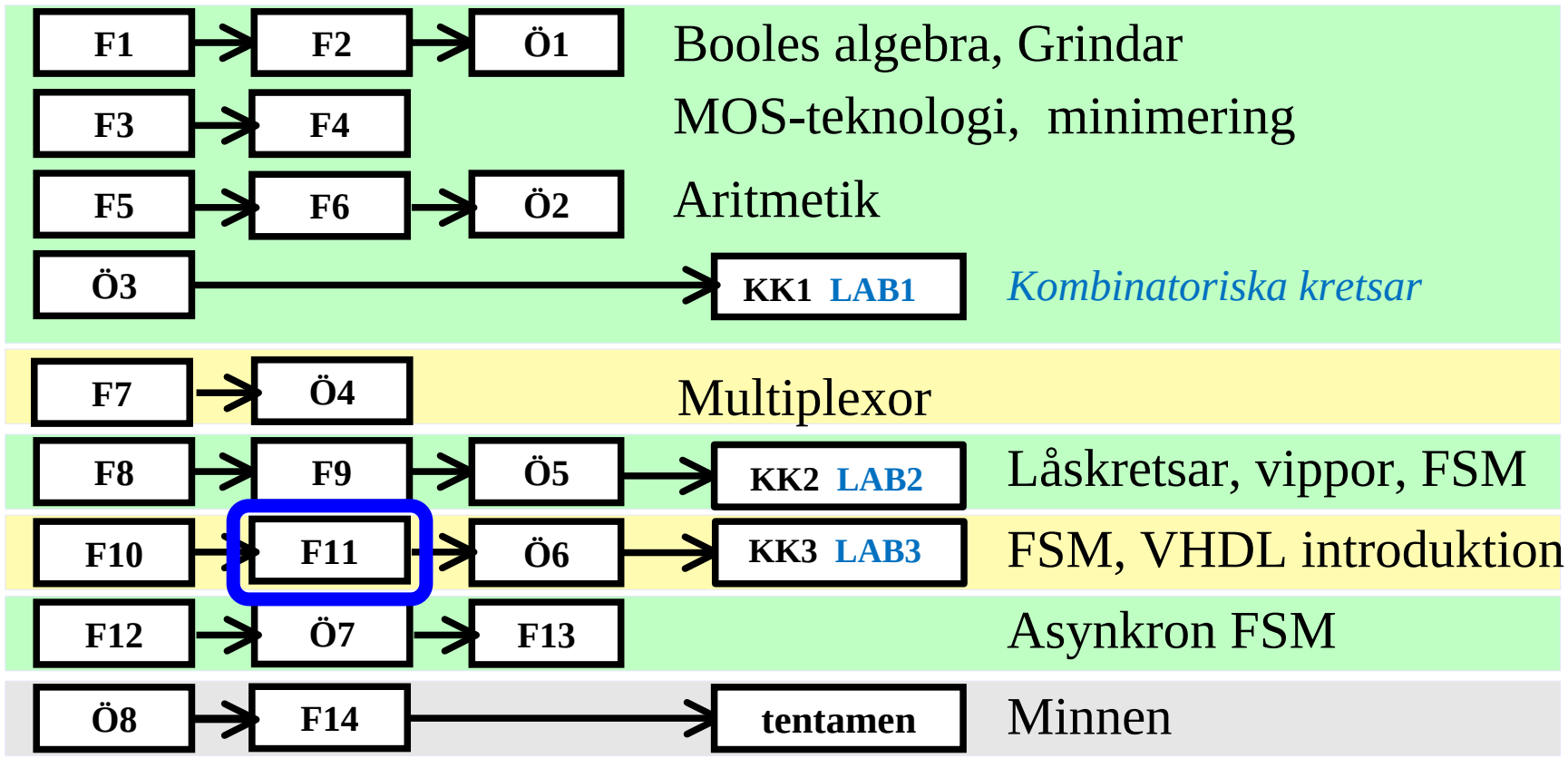
Digital Design IE1204

F11 Programmerbar logik
VHDL för sekvensnät

william@kth.se

IE1204 Digital Design



Föreläsningar och övningar bygger på varandra! Ta alltid igen det Du missat!
Läs på i förväg – delta i undervisningen – arbeta igenom materialet efteråt!

William Sandqvist william@kth.se

Detta har hänt i kursen …

Decimala, hexadecimala, oktala och binära talsystemen
AND OR NOT EXOR EXNOR Sanningstabell, mintermer Maxtermer PS-form
Booles algebra SP-form deMorgans lag Bubbelgrindar Fullständig logik
NAND NOR CMOS grindar, standardkretsar Minimering med Karnaugh-diagram 2, 3, 4, 5, 6 variabler
Registeraritmetik tvåkomplementrepresentation av binära tal
Additionskretsar Multiplikationskrets Divisionskrets
Multiplexorer och Shannon dekomposition Dekoder/Demultiplexor Enkoder
Prioritetsenkoder Kodomvandlare
VHDL introduktion
Vippor och Låskretsar SR-latch D-latch D-vippa JK-vippa T-vippa Räknare
Skiftregister Vippor i VHDL Moore-automat Mealy-automat Tillståndskod

Oanvända tillstånd Analys av sekvensnät Tillståndsminimering

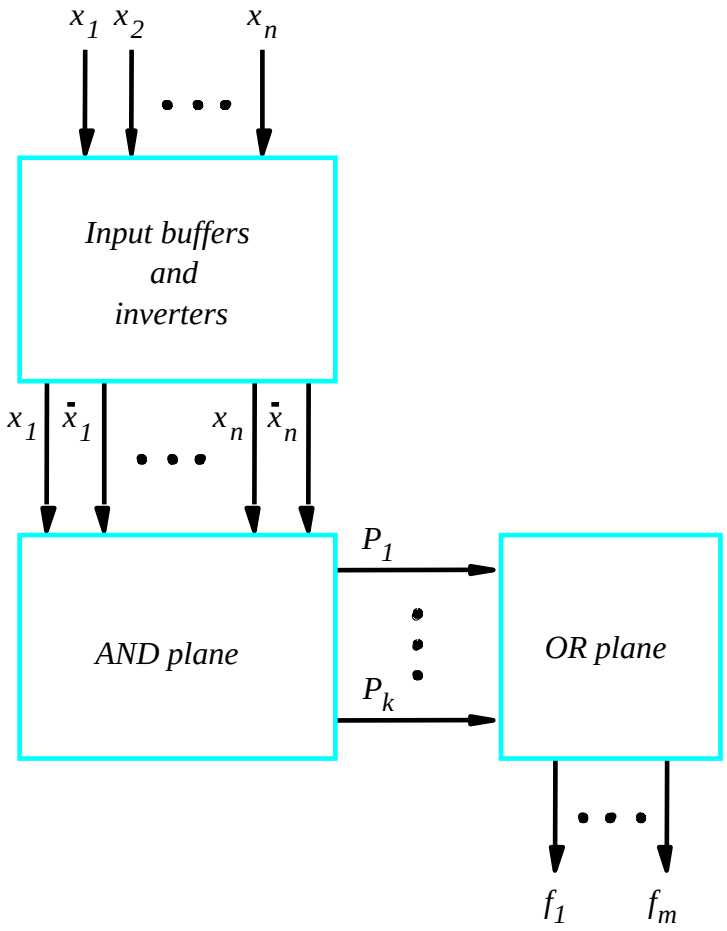
William Sandqvist william@kth.se

Programable Logic Devices

Under 1970-talet introducerades programmerbara logiska kretsar som betecknas programmable logic device (PLD) De bygger på en struktur med en AND-OR-matris som gör det enkelt att implementera SOP-uttryck

William Sandqvist william@kth.se

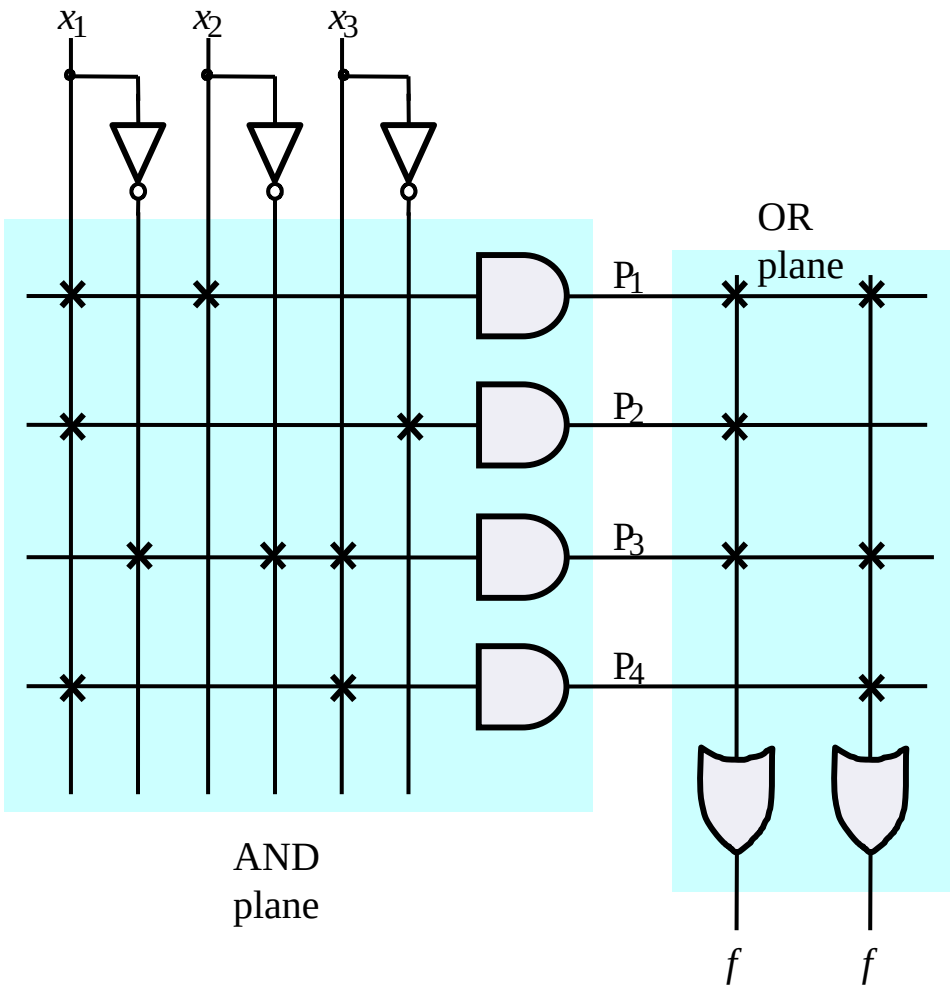
PLD struktur



William Sandqvist william@kth.se

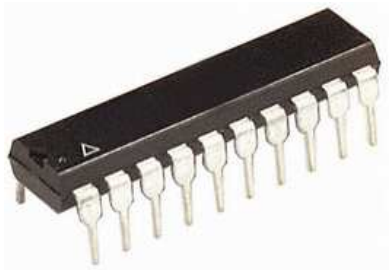
Programmable Logic Array (PLA)

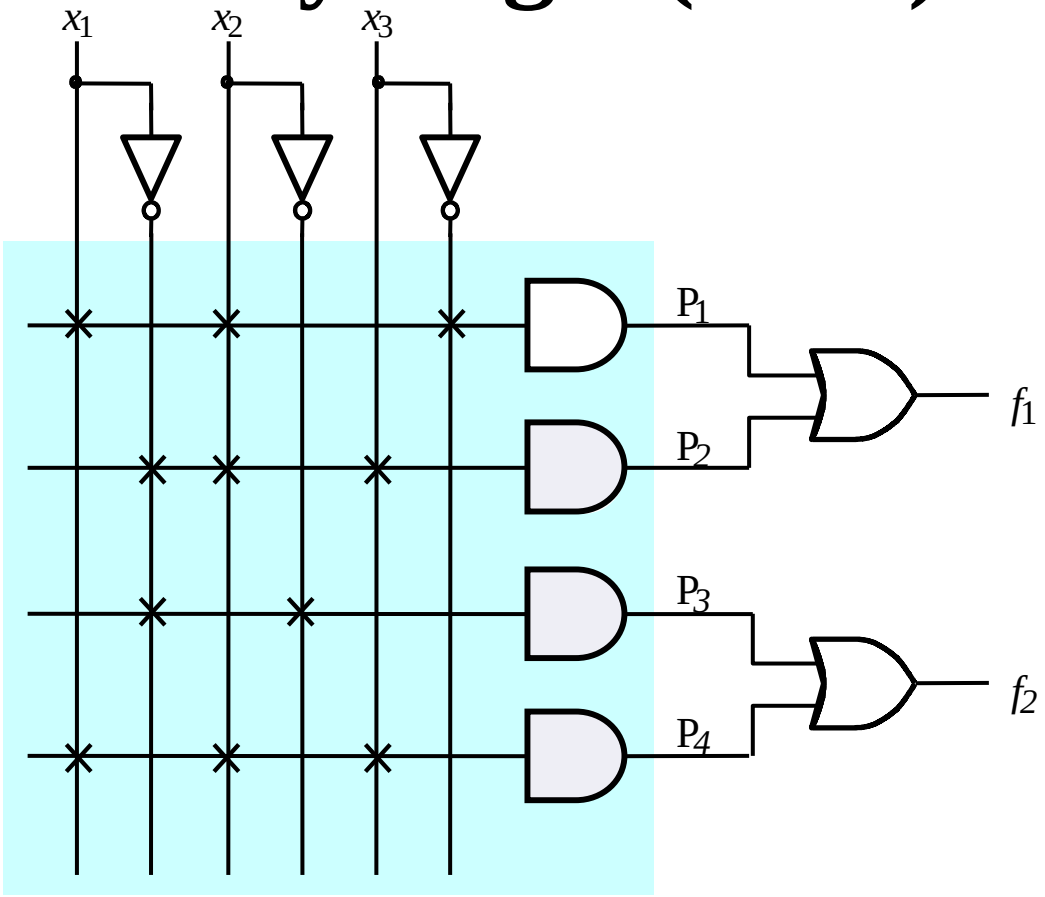
Både AND- och OR-matriserna är programmeringsbara



Programmable Array Logic (PAL)

Bara AND matrisen är programmeringsbar





AND plane

William Sandqvist william@kth.se

Registerutgångar

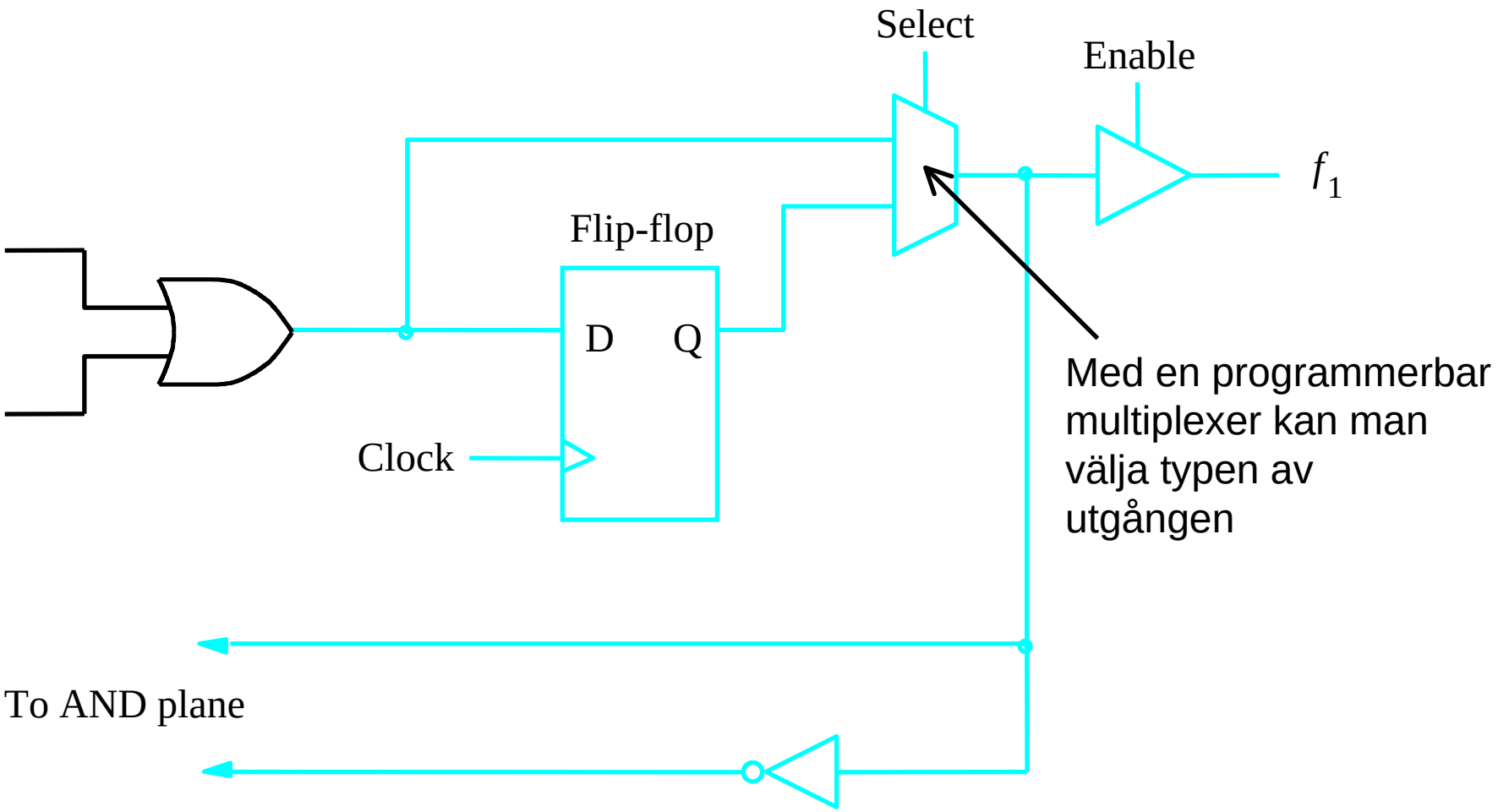
I de tidigare PLD-kretasarna fanns det

- kombinatoriska utgångar
- registerutgångar (utgångar med en vippa)

För varje krets fanns det ett **fast antal** kombinatoriska och registerutgångar

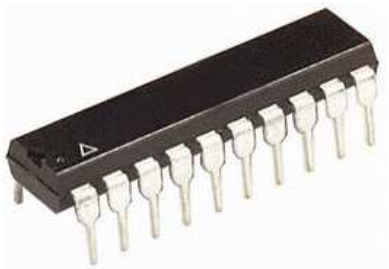
För att öka flexibiliteten introducerade man **makrocellen** där man kunde välja om en utgång skulle vara en kombinatorisk eller en registerutgång

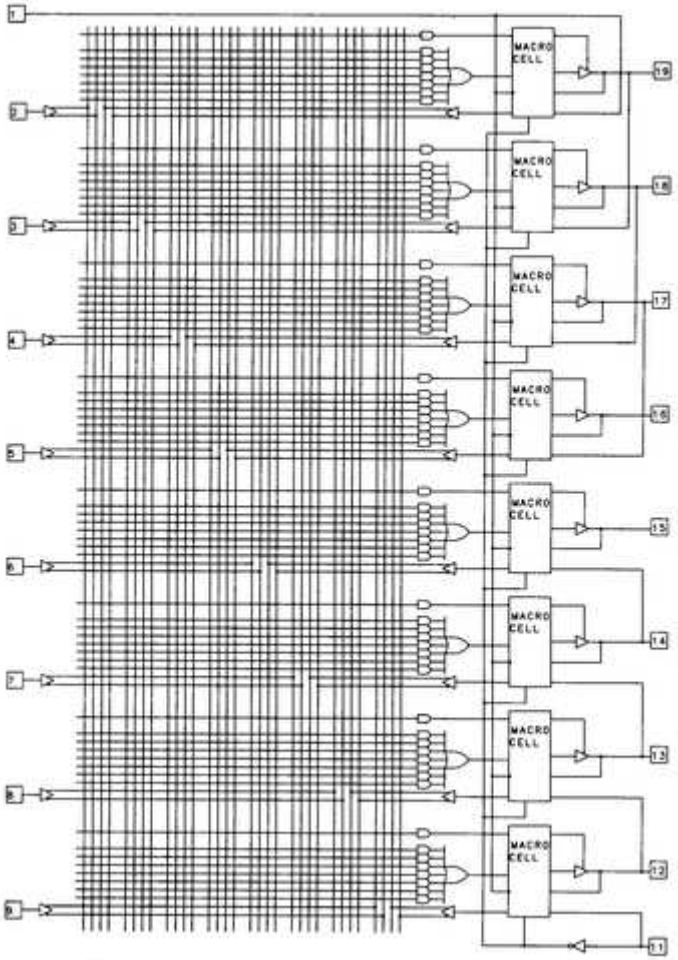
Makroceller i en PLD



William Sandqvist william@kth.se

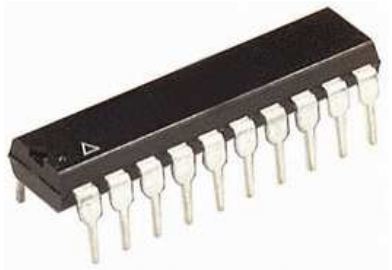
PAL





William Sandqvist william@kth.se

Programmering av PLD:er



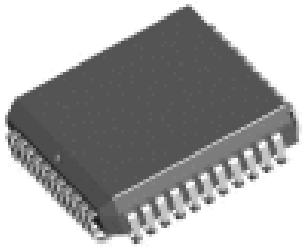
William Sandqvist william@kth.se

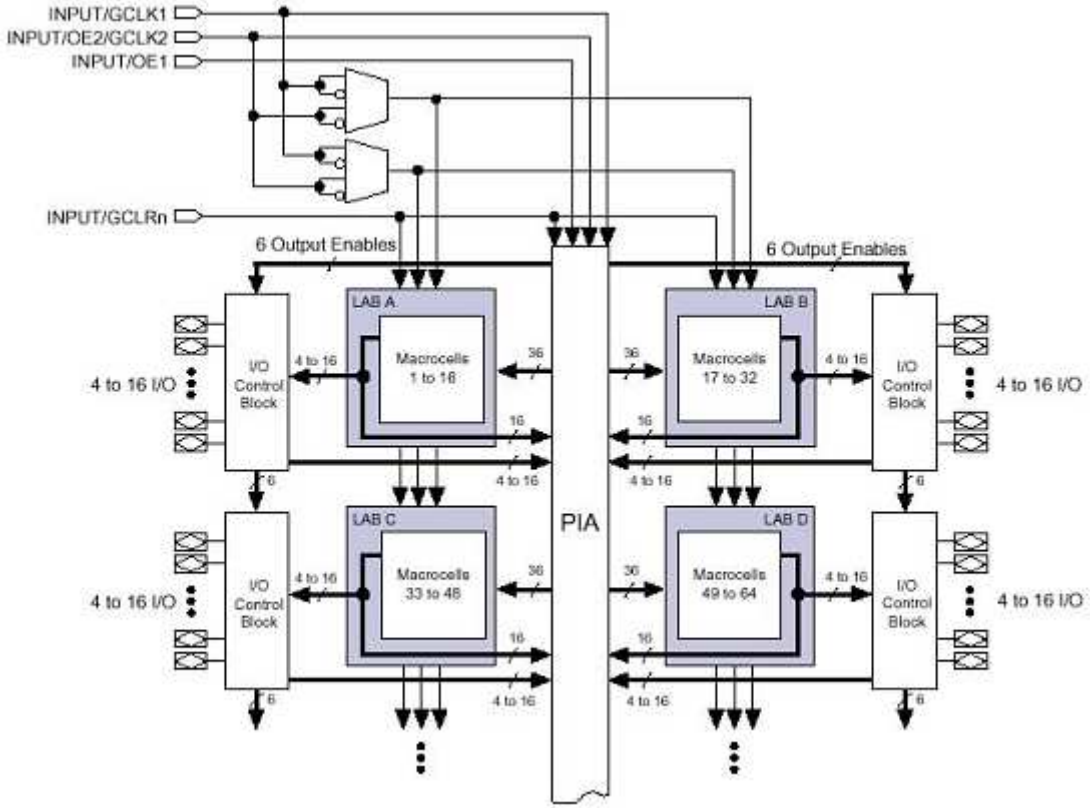
Komplexa PLD:er (CPLD)

PLD:erna var ganska små (PALCE 22V10 hade 10 vippor)

För att skapa större programmerbara kretsar utvecklade man en struktur bestående av flera PLD-liknande block

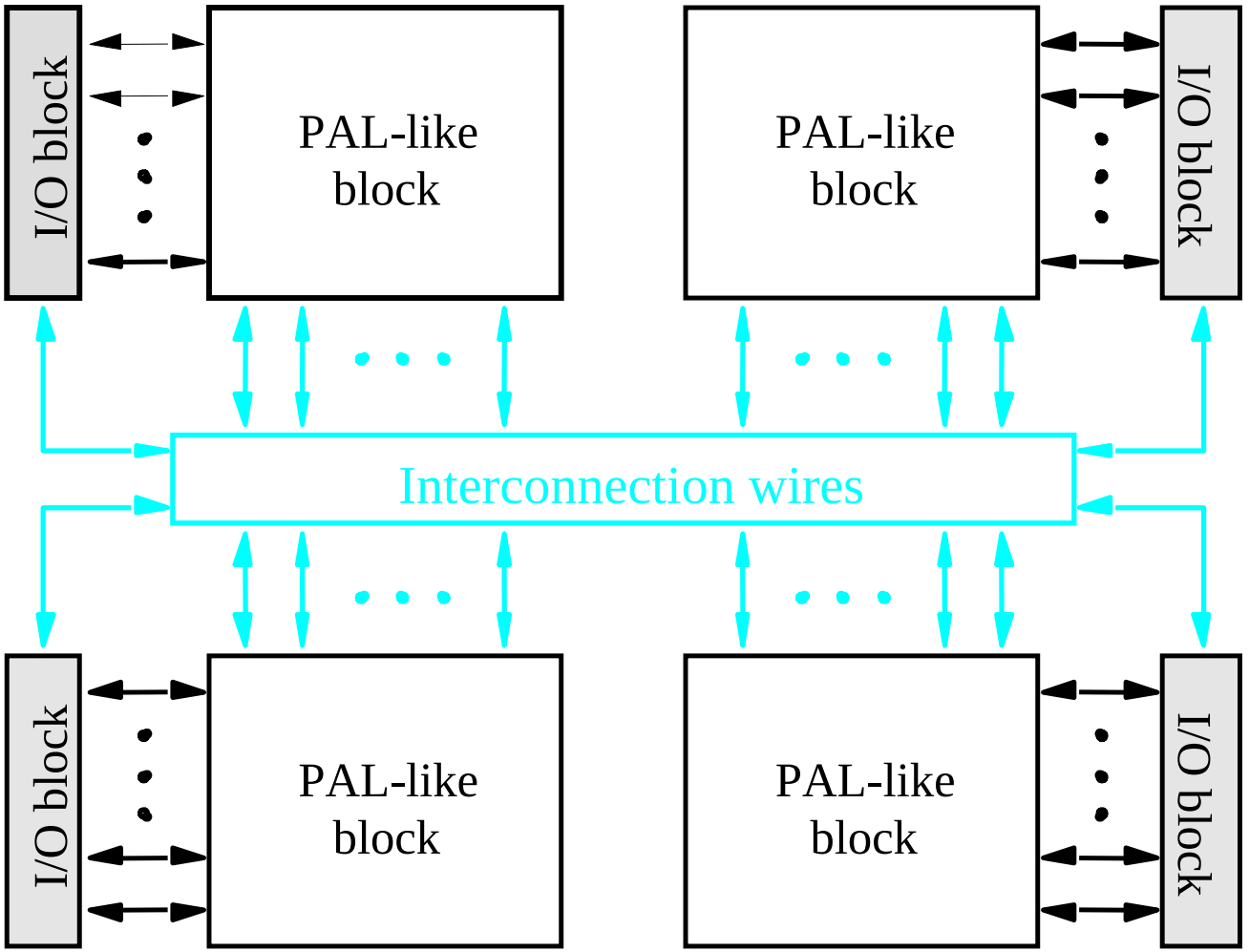
CPLD (MAX)





William Sandqvist william@kth.se

CPLD struktur



William Sandqvist william@kth.se

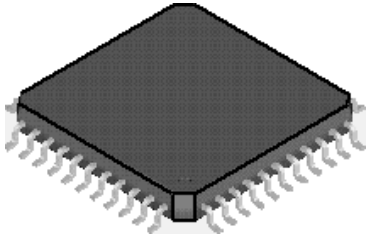
Programmering med JTAG

Moderna CPLD:er (och FPGA:er) kan programmeras genom att ladda ned kretsbekrivningen (programmerings-informationen) via en kabel

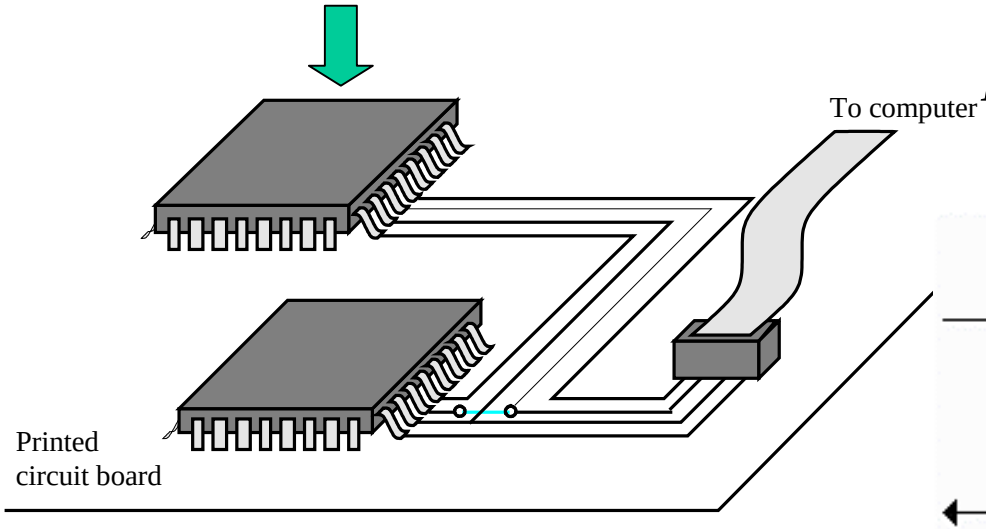
Nedladdningen använder oftast en standardiserad port: ***JTAG-porten***

William Sandqvist william@kth.se

JTAG programmering

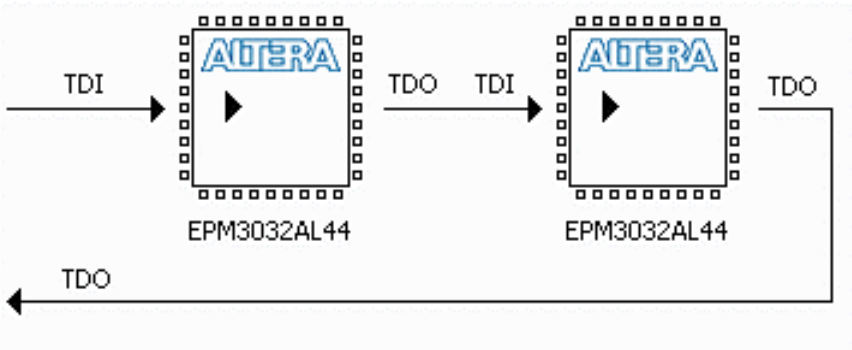


(a) CPLD in a Quad Flat Pack (QFP) package



(b) JTAG programming

Man kan programmera chippen när de är fastlödda på kretskortet – innifrån programmeraren kan man välja ut vilket chip man vill programmera från JTAG-kontakten.



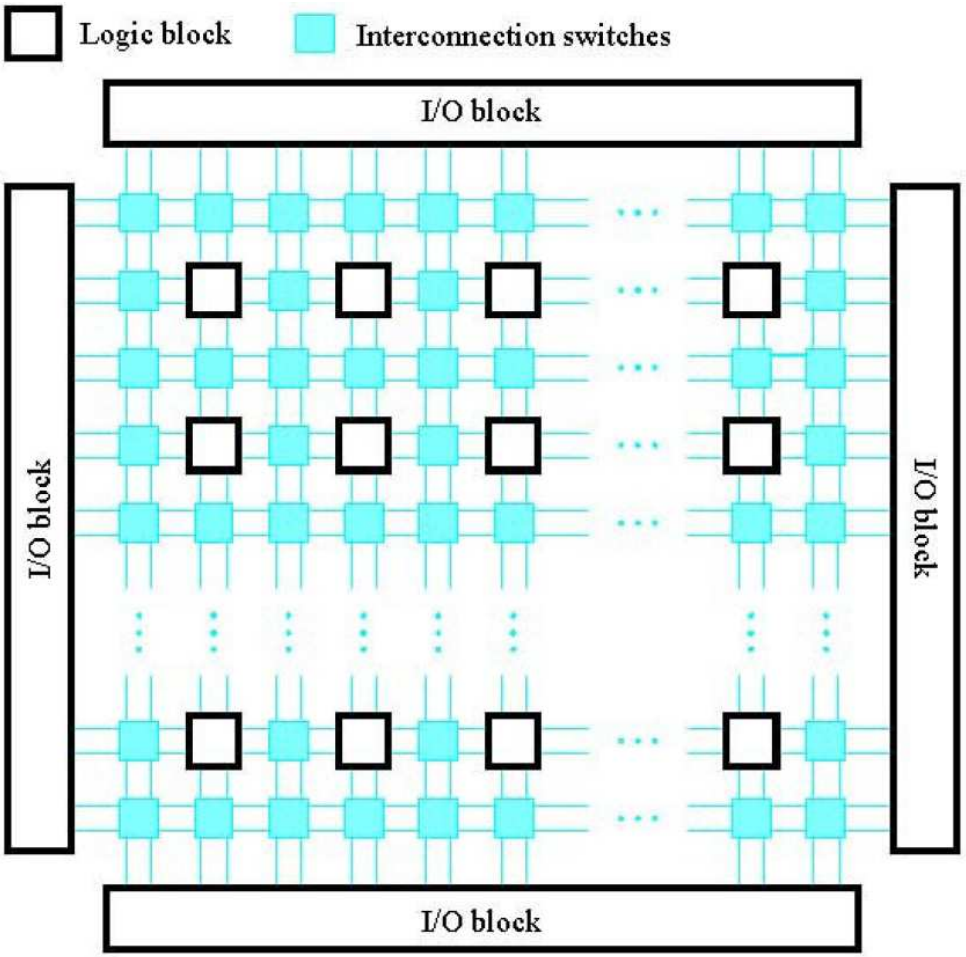
William Sandqvist william@kth.se

FPGA kretsar

CPLD:er baseras på AND-OR-matrisen och det blir svårt att göra riktigt stora kretsar

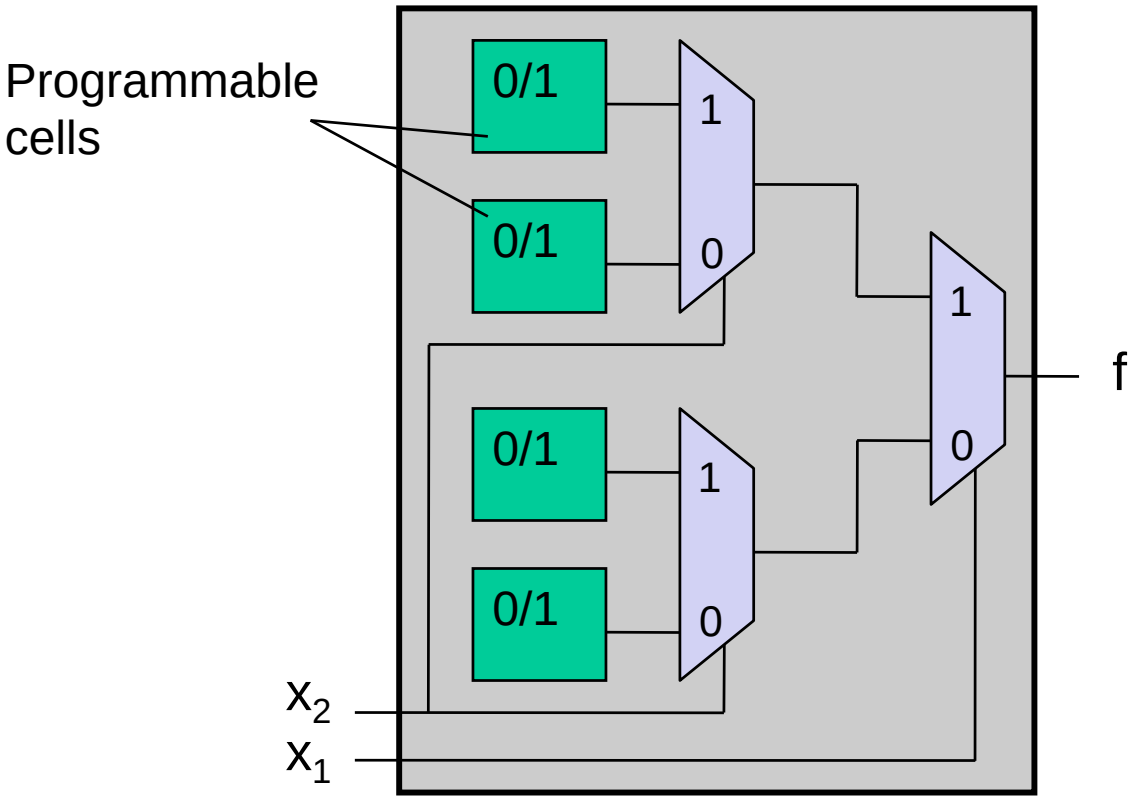
FPGA (Field Programmable Gate Array) kretsarna använder en annan koncept som baseras på ***logiska block***

FPGA-struktur



William Sandqvist william@kth.se

LUT-LookUp-Table

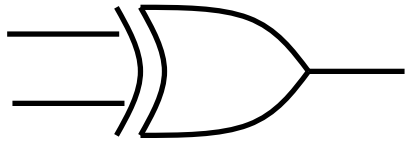


Two-input LUT

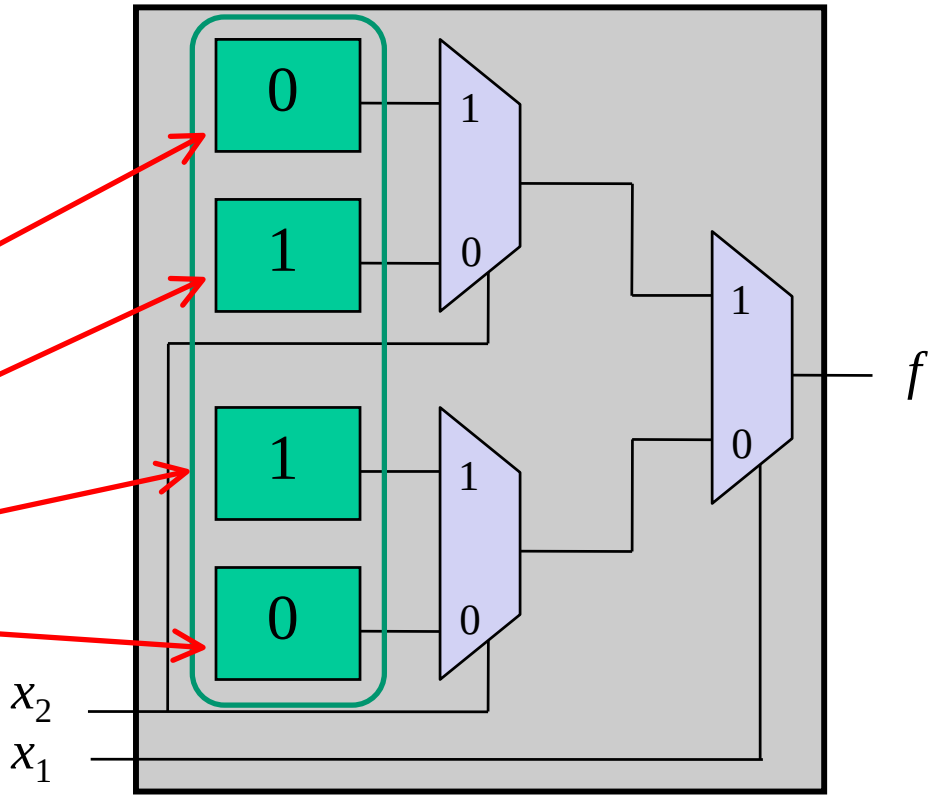
A LUT with *n* inputs can realize ***all*** combinational functions with *n* inputs.
The usual size in an FPGA is *n*=4

William Sandqvist william@kth.se

Ex. LUT för XOR-grind



x_2	x_1	f
1	1	0
0	1	1
1	0	1
0	0	0

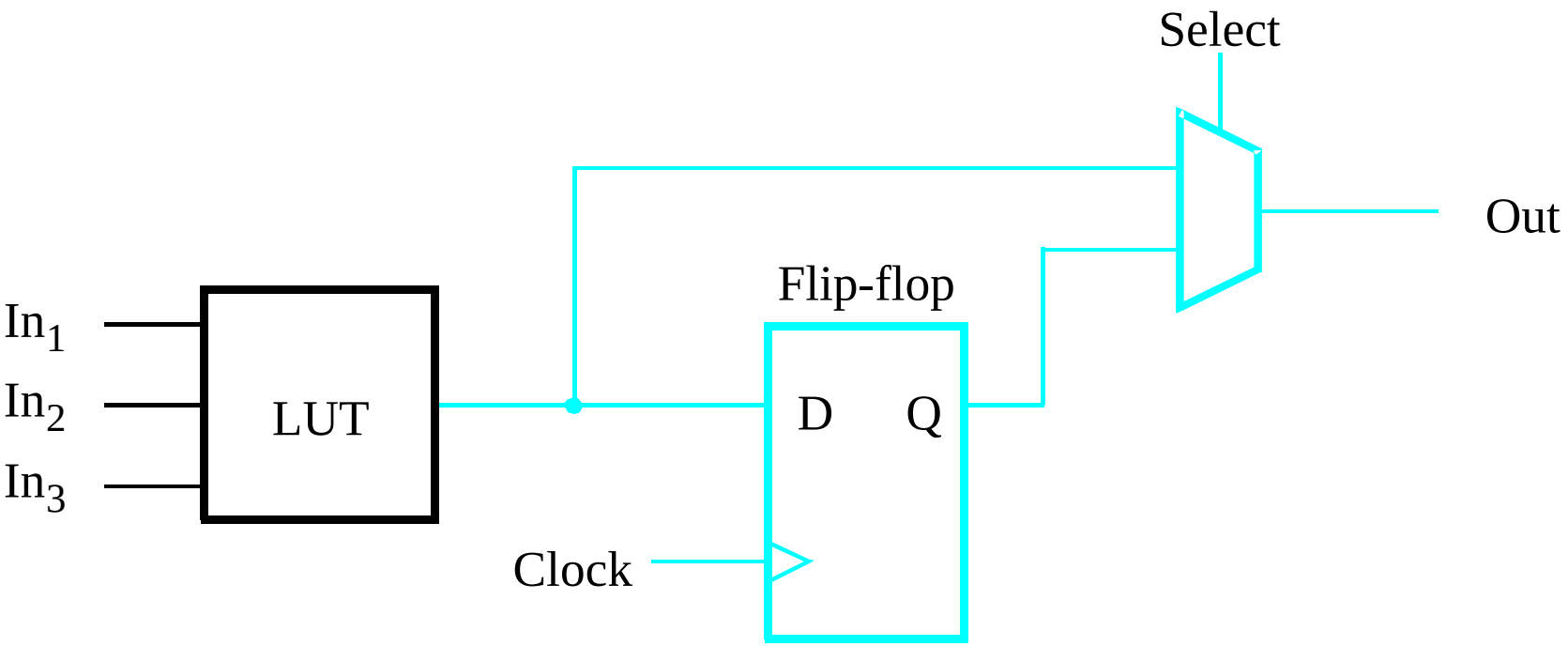


Two-input LUT

William Sandqvist william@kth.se

Logiskt block i FPGA

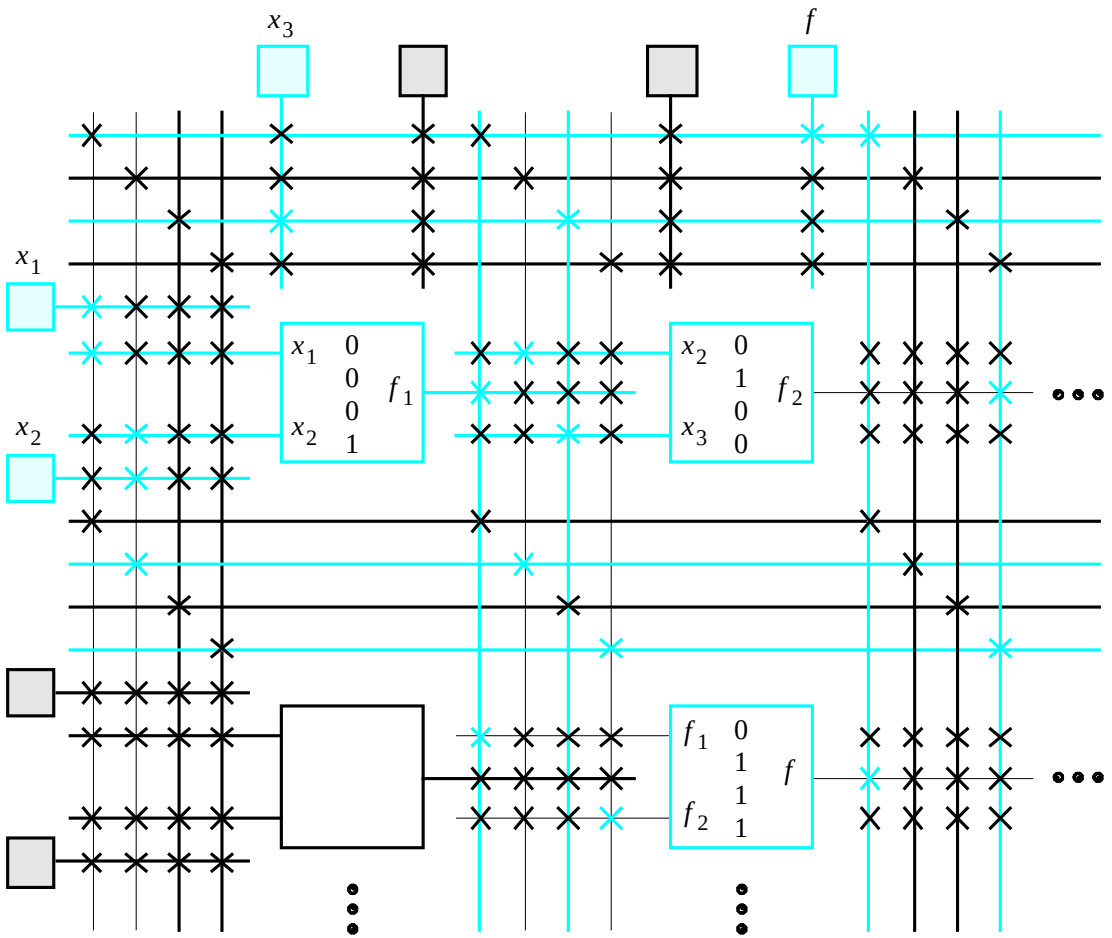
Ett logiskt block i en FPGA består av en LUT, en vippa, och en mux för val av registerutgång.



William Sandqvist william@kth.se

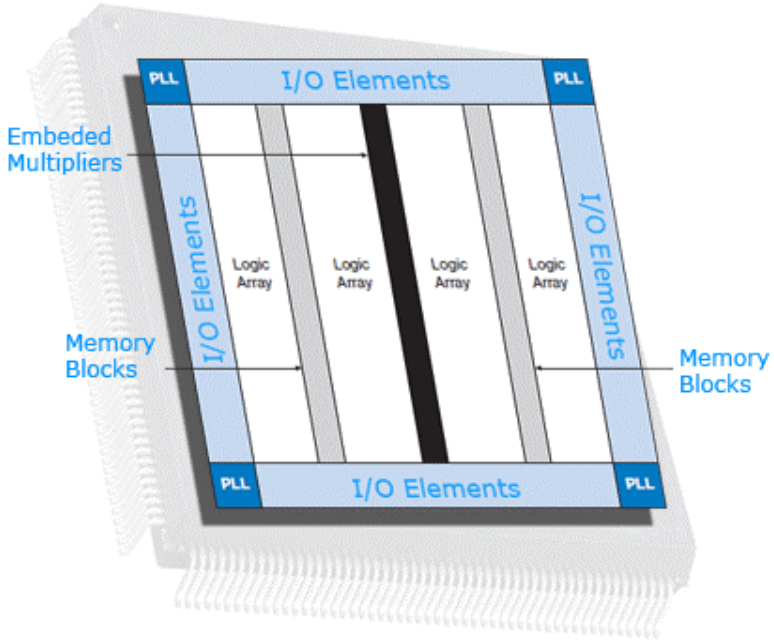
Förbindelsematris i FPGA

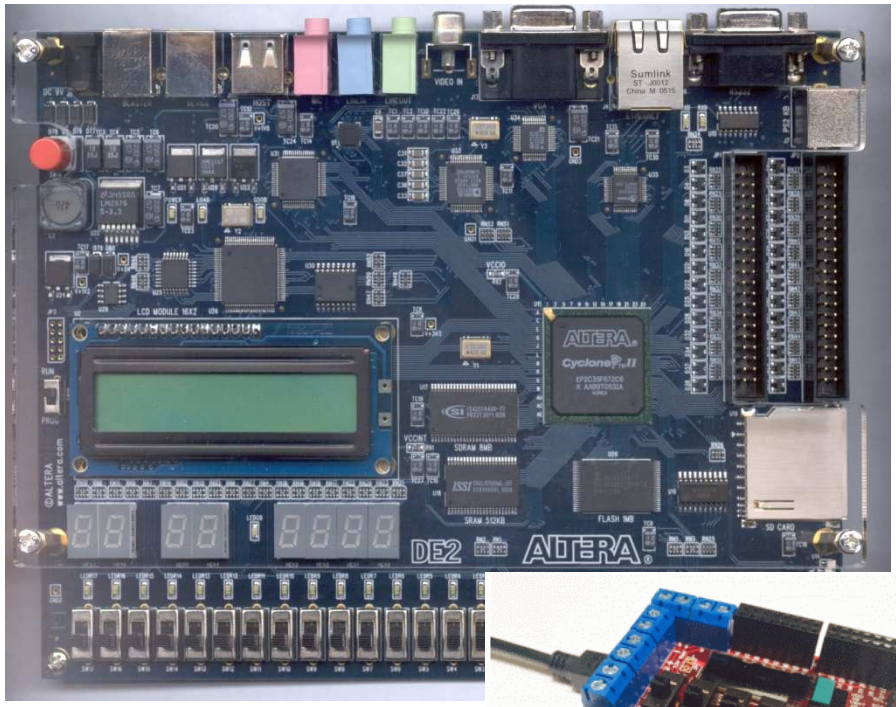
- Blå kryss: Förbindelsen är programmerad
- Svart kryss: Förbindelsen är inte programmerad



William Sandqvist william@kth.se

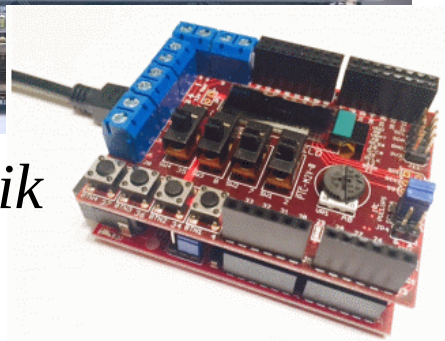
DE2 University Board





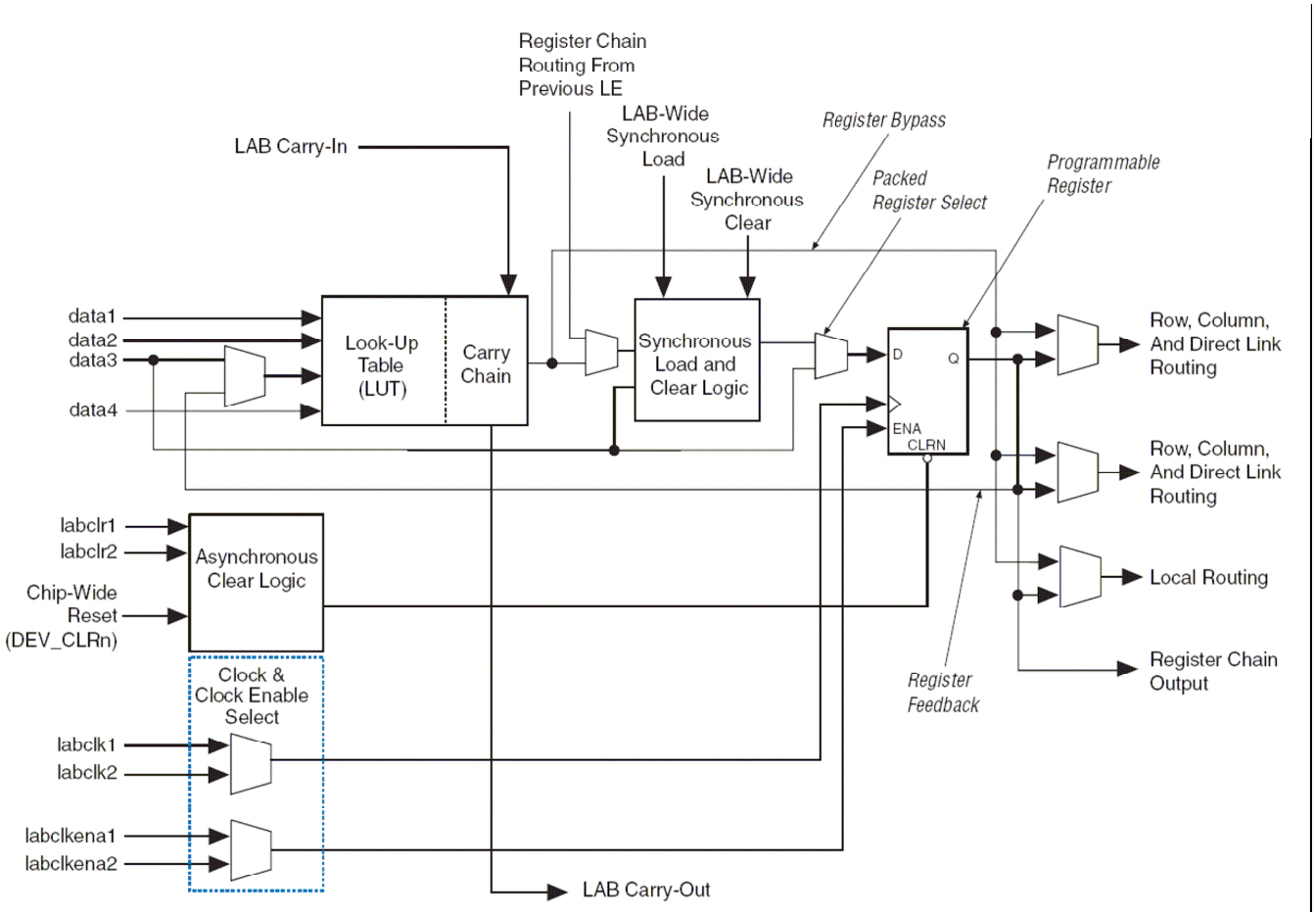
Cyclone II EP2C35 FPGA – in Master program

Dator teknik PIC32!



William Sandqvist william@kth.se

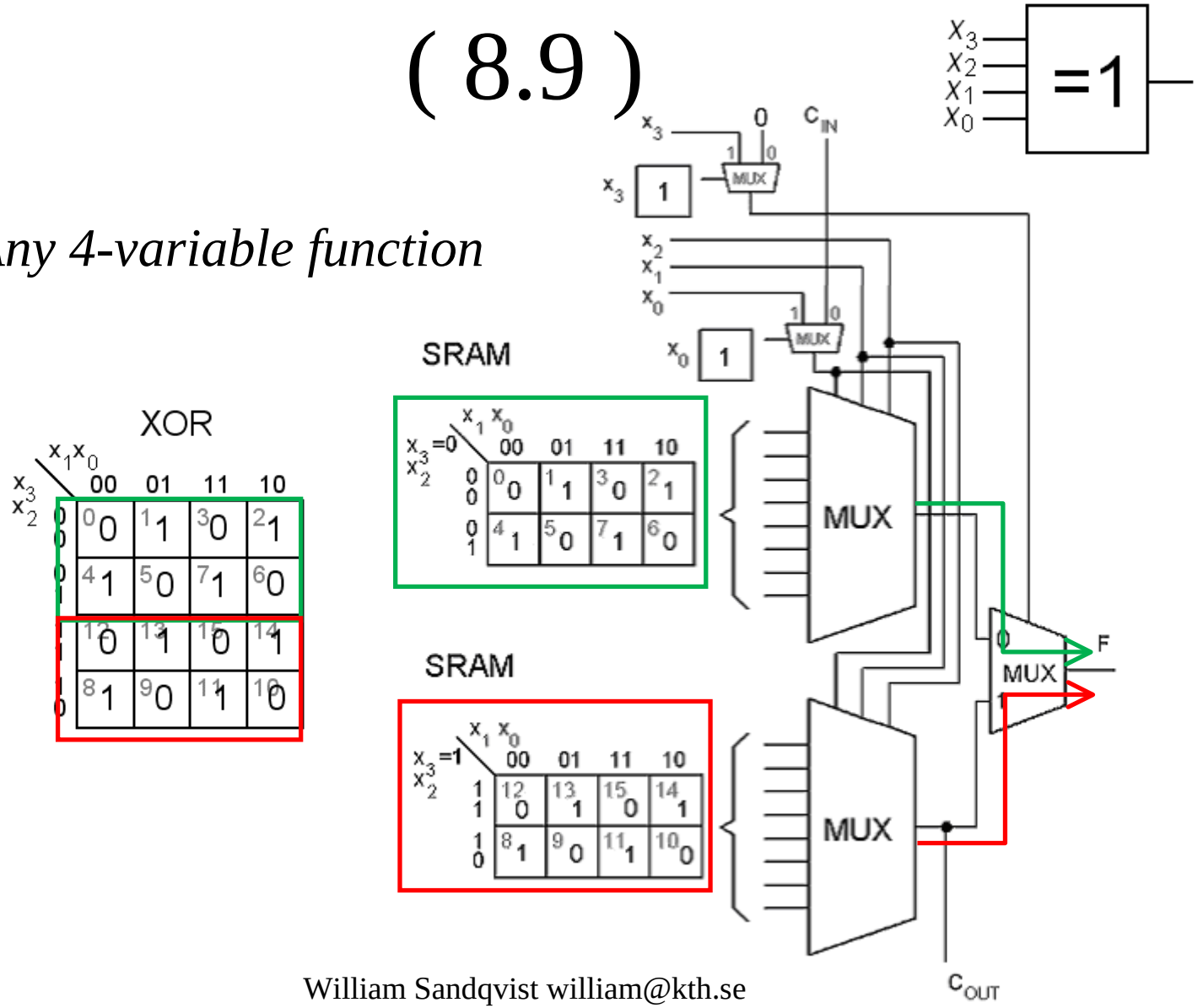
Cyclone II logic element

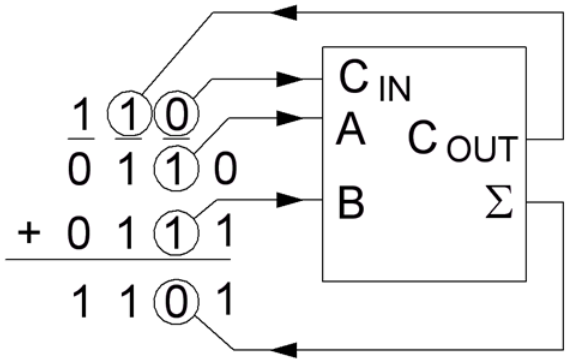


William Sandqvist william@kth.se

(8.9)

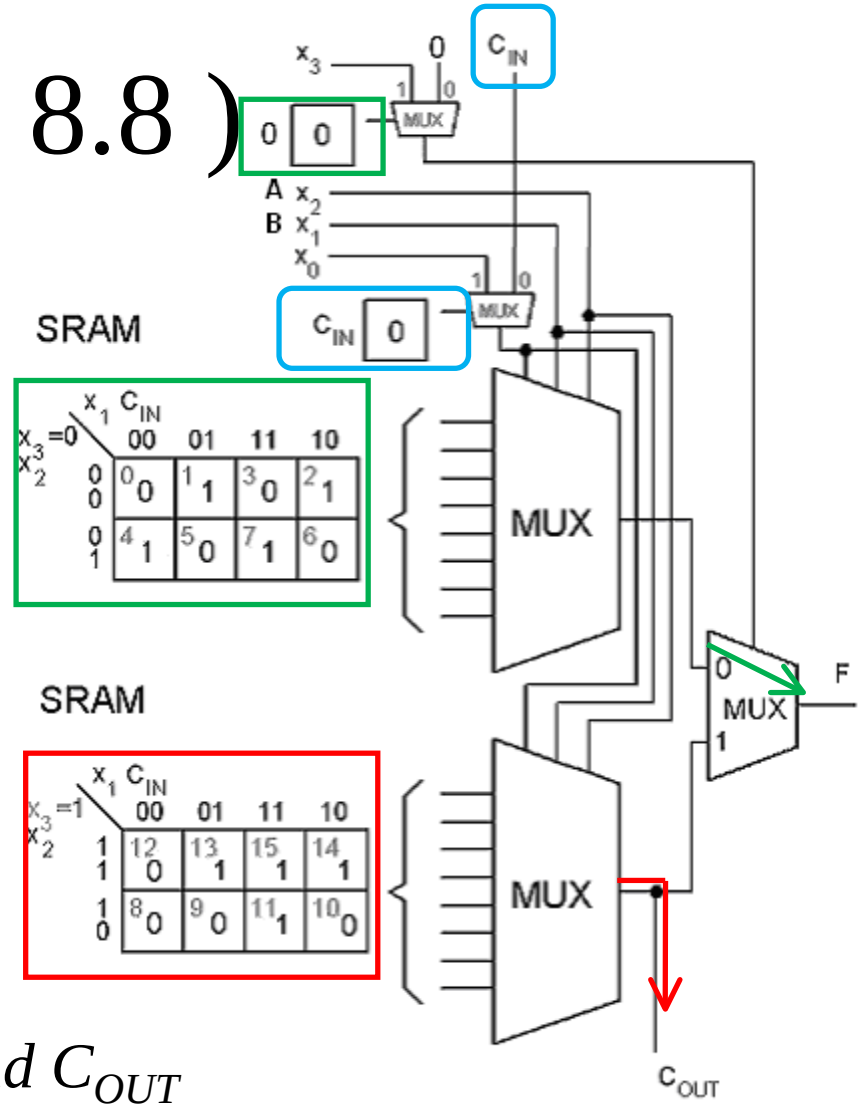
- *Any 4-variable function*





(8.8)

x_2	x_1	x_0		
A	B	C_{IN}	Σ	C_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



- *Or a Full Adder with S and C_{OUT}*

Cyclone II Family

Table 1–1. Cyclone II FPGA Family Features

Feature	EP2C5	EP2C8 *(2)*	EP2C15 *(1)*	EP2C20 *(2)*	EP2C35	EP2C50	EP2C70
LEs	4,608	8,256	14,448	18,752	33,216	50,528	68,416
M4K RAM blocks (4 Kbits plus 512 parity bits	26	36	52	52	105	129	250
Total RAM bits	119,808	165,888	239,616	239,616	483,840	594,432	1,152,000
Embedded multipliers *(3)*	13	18	26	26	35	86	150
PLLs	2	2	4	4	4	4	4
Maximum user I/O pins	158	182	315	315	475	450	622

DE2

(3) Total Number of 18x18 Multipliers

Stratix III Family

Table 1–1. Stratix III FPGA Family Features

	Device/ Feature	ALMs	LEs	M9K Blocks	M144K Blocks	MLAB Blocks	Total Embedded RAM Kbits	MLAB RAM Kbits *(2)*	Total RAM Kbits *(3)*	18×18-bit Multipliers (FIR Mode)	PLLs
Stratix III Logic Family	EP3SL50	19K	47.5K	108	6	950	1,836	297	2,133	216	4
	EP3SL70	27K	67.5K	150	6	1,350	2,214	422	2,636	288	4
	EP3SL110	43K	107.5K	275	12	2,150	4,203	672	4,875	288	8
	EP3SL150	57K	142.5K	355	16	2,850	5,499	891	6,390	384	8
	EP3SL200	80K	200K	468	36	4,000	9,396	1,250	10,646	576	12
	EP3SE260	102K	255K	864	48	5,100	14,688	1,594	16,282	768	12
	EP3SL340	135K	337.5K	1,040	48	6,750	16,272	2,109	18,381	576	12
Stratix III Enhanced Family	EP3SE50	19K	47.5K	400	12	950	5,328	297	5,625	384	4
	EP3SE80	32K	80K	495	12	1,600	6,183	500	6,683	672	8
	EP3SE110	43K	107.5K	639	16	2,150	8,055	672	8,727	896	8
	EP3SE260 *(1)*	102K	255K	864	48	5,100	14,688	1,594	16,282	768	12

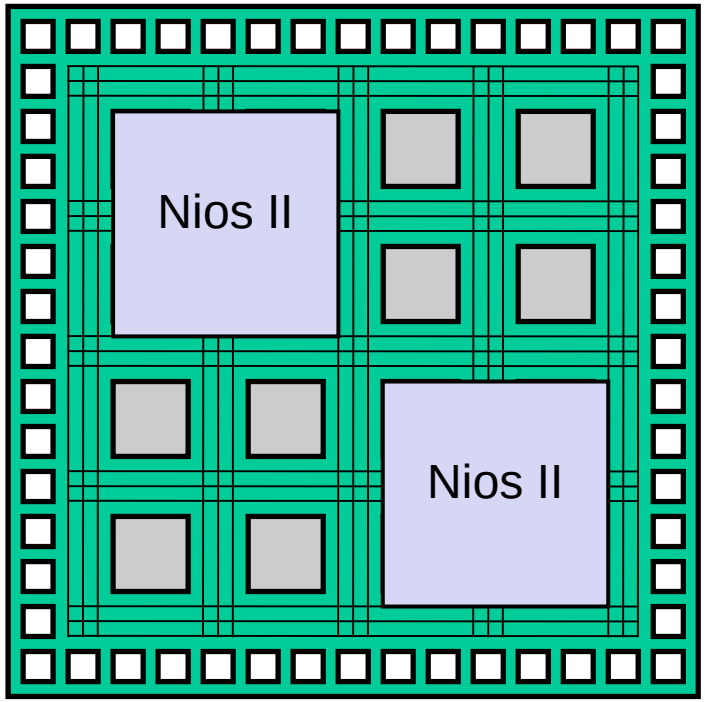
DE3 Board

Used in Master programs

William Sandqvist william@kth.se

Flera processorer på en FPGA

- Nios II är en så kallad ‘soft-processor’ (32-bit) som kan implementeras på Altera FPGA-kretsen
- Dagens FPGA-kretsar är så stora att flera processorer får plats på en enda FPGA-krets



Mycket kraftfulla multiprocessor system kan skapas på en FPGA!

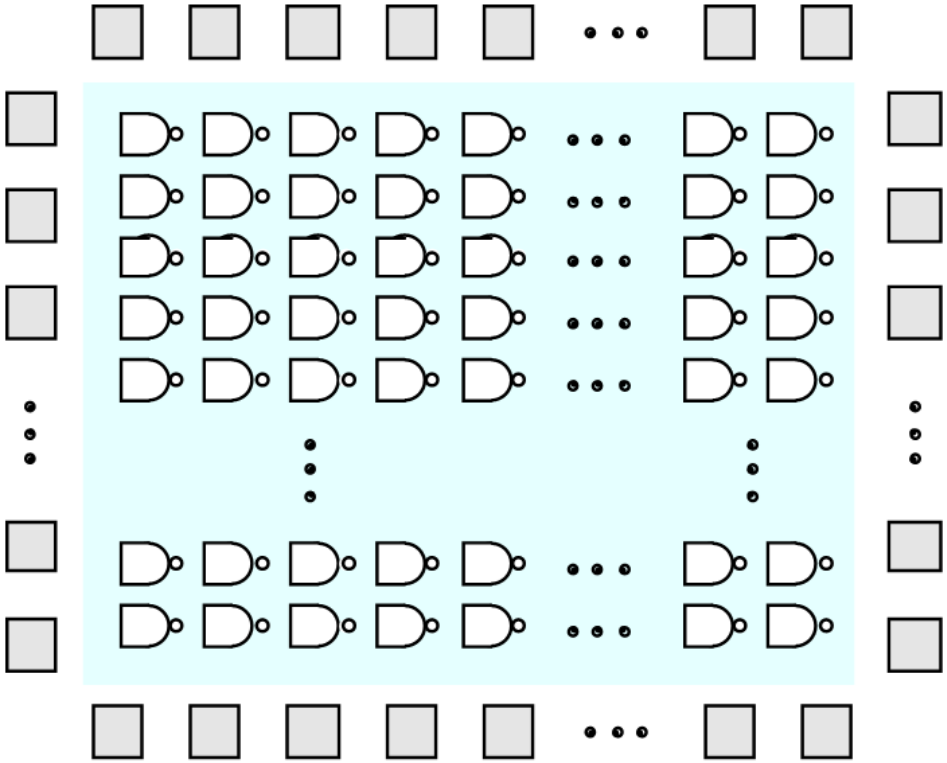
William Sandqvist william@kth.se

ASIC

- En **ASIC** (Application Specific Integrated Circuit) är en krets som görs i en halvledarfabrik
- I en ***full custom*** integrerad krets skräddarsyr man i princip hela kretsen
- I en **ASIC** har vissa arbetssteg redan gjorts för att minska design-tiden och kostnaden

ASIC, gate array

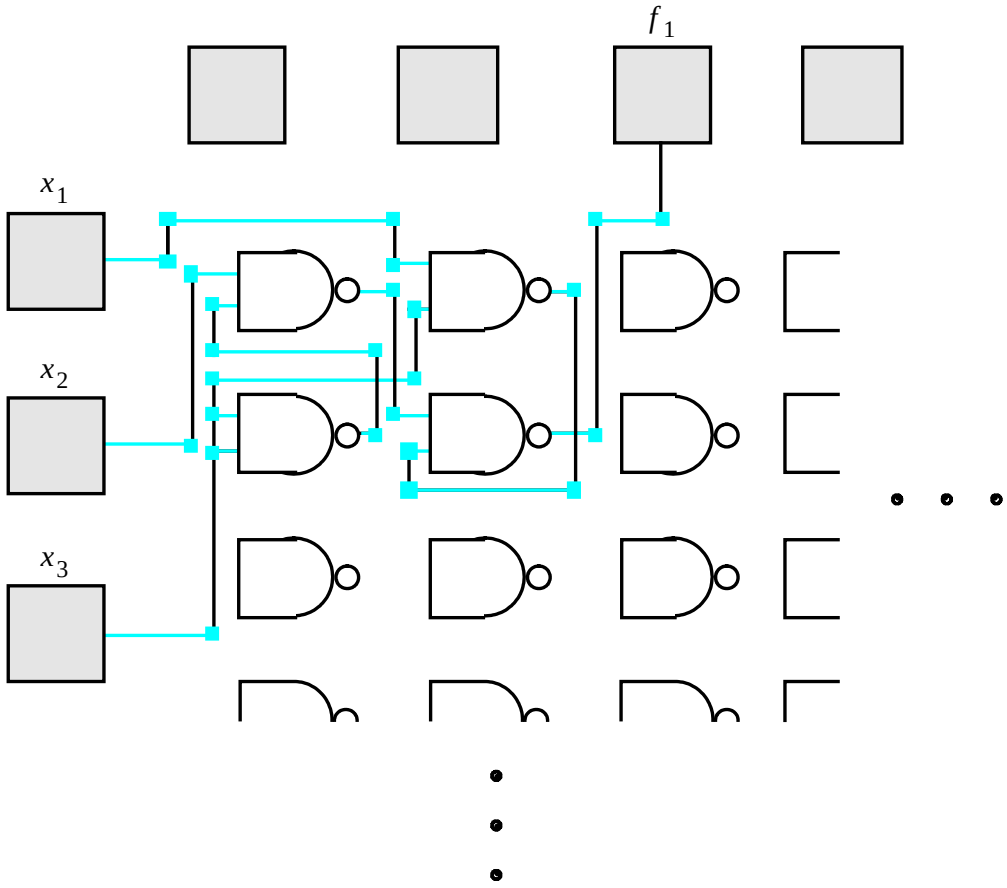
I en Gate Array finns redan grindarna (eller transistorerna) på kiseln



William Sandqvist william@kth.se

ASIC, gate array

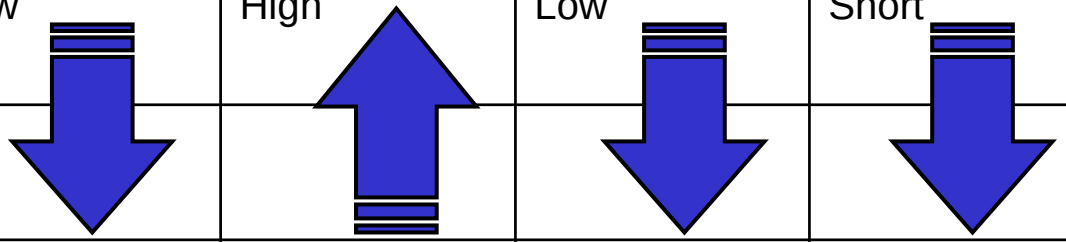
Man skapar bara förbindelserna mellan ingångarna, grindarna, och utgångarna



William Sandqvist william@kth.se

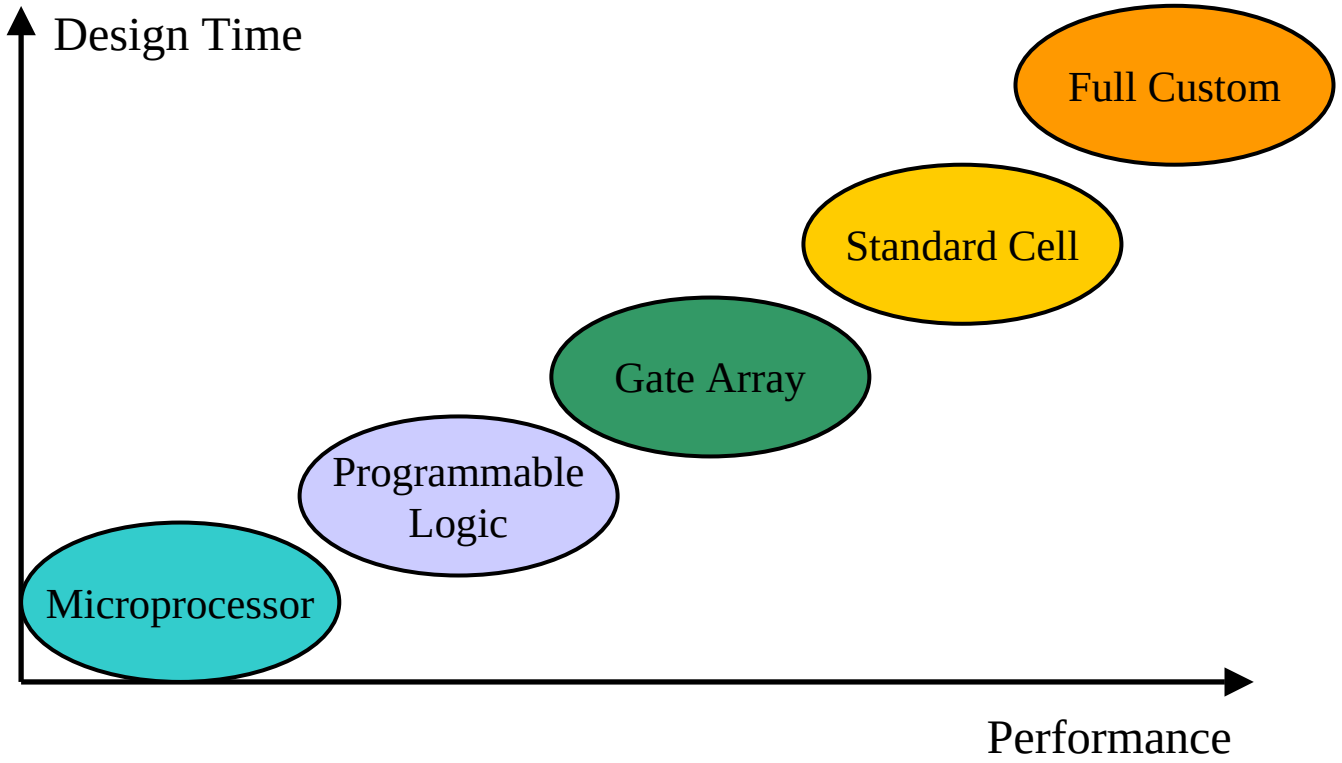
Jämförelser ASIC, FPGA

	Initial Cost	Cost per part	Performance	Fabrication Time
FPGA	Low	High	Low	Short
Gate Array (ASIC)				
Standard Cell (ASIC)	High	Low	High	Long



William Sandqvist william@kth.se

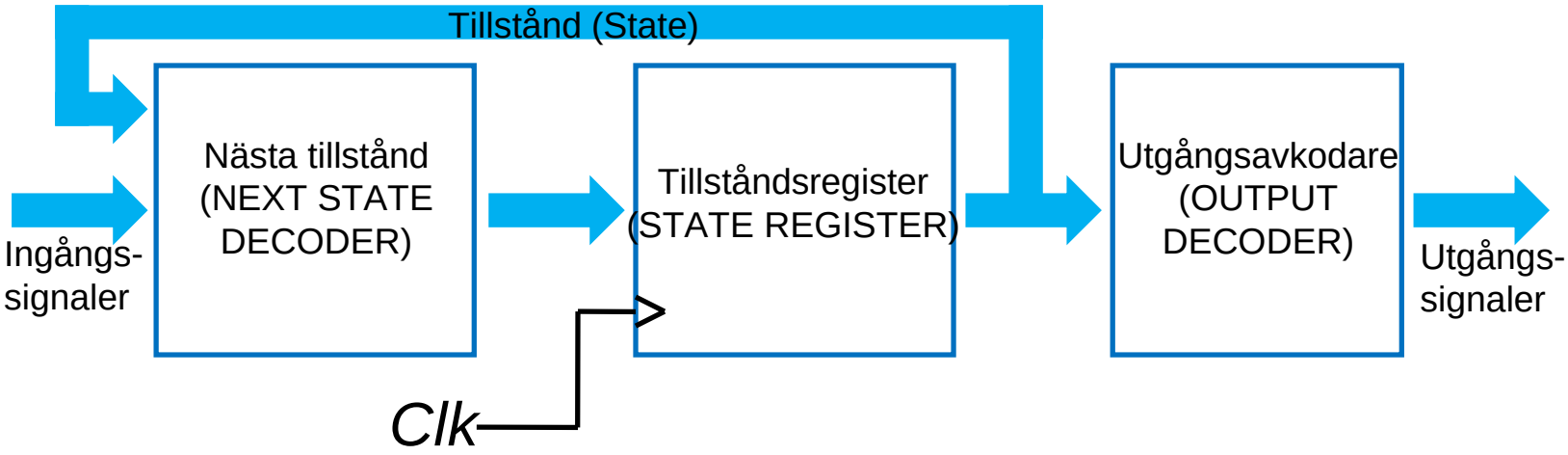
Design Trade-Offs



William Sandqvist william@kth.se

William Sandqvist william@kth.se

Sekvenskretsar med VHDL



Moore-automat

William Sandqvist william@kth.se

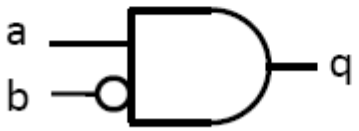
Modellera Statemachine i VHDL

- I en Moore-automat har vi tre block
  - Nästa-tillståndsavkodare
  - Utgångsavkodare
  - Tillståndsregister
- Dessa block exekverars parallellt

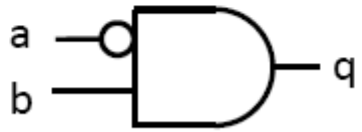
Snabbfråga

Vilken logisk grind motsvarar följande VHDL kod?

```
q <= a and (not b);
```



Alt: A



Alt: B

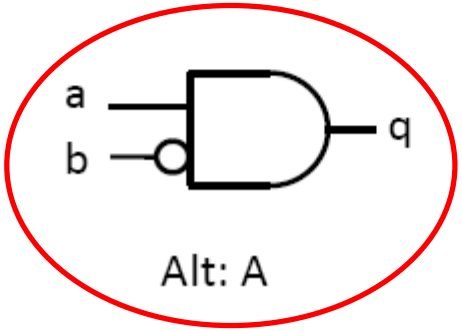
Alt: C

William Sandqvist william@kth.se

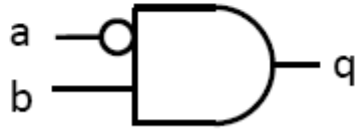
Snabbfråga

Vilken logisk grind motsvarar följande VHDL kod?

```
q <= a and (not b);
```



Alt: A



Alt: B

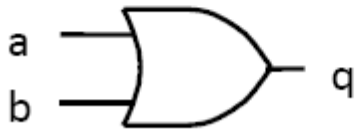
Alt: C

William Sandqvist william@kth.se

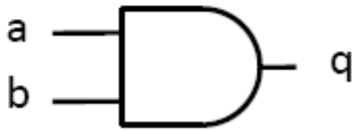
Snabbfråga

Vilken logisk grind motsvarar följande VHDL kod?

```
if (a /= b) then
  q <= '1';
else
  q <= '0';
end if;
```



Alt: A



Alt: B

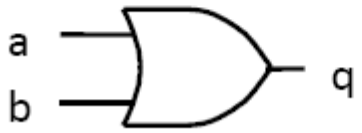
Alt: C

William Sandqvist william@kth.se

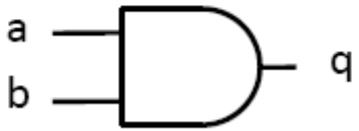
Snabbfråga

Vilken logisk grind motsvarar följande VHDL kod?

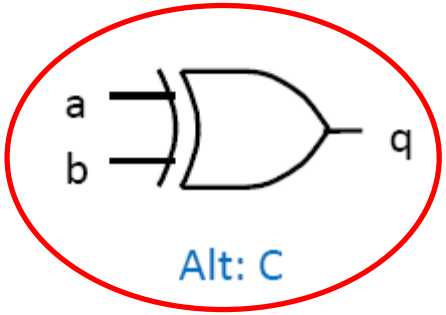
```
if (a /= b) then
  q <= '1';
else
  q <= '0';
end if;
```



Alt: A



Alt: B



Alt: C

William Sandqvist william@kth.se

Processer i VHDL

- En *architecture* i VHDL kan innehåller flera processer
- Processer exekveras parallelt
- En process är skriven som ett sekvensiellt program

William Sandqvist william@kth.se

Moore-automatens processer

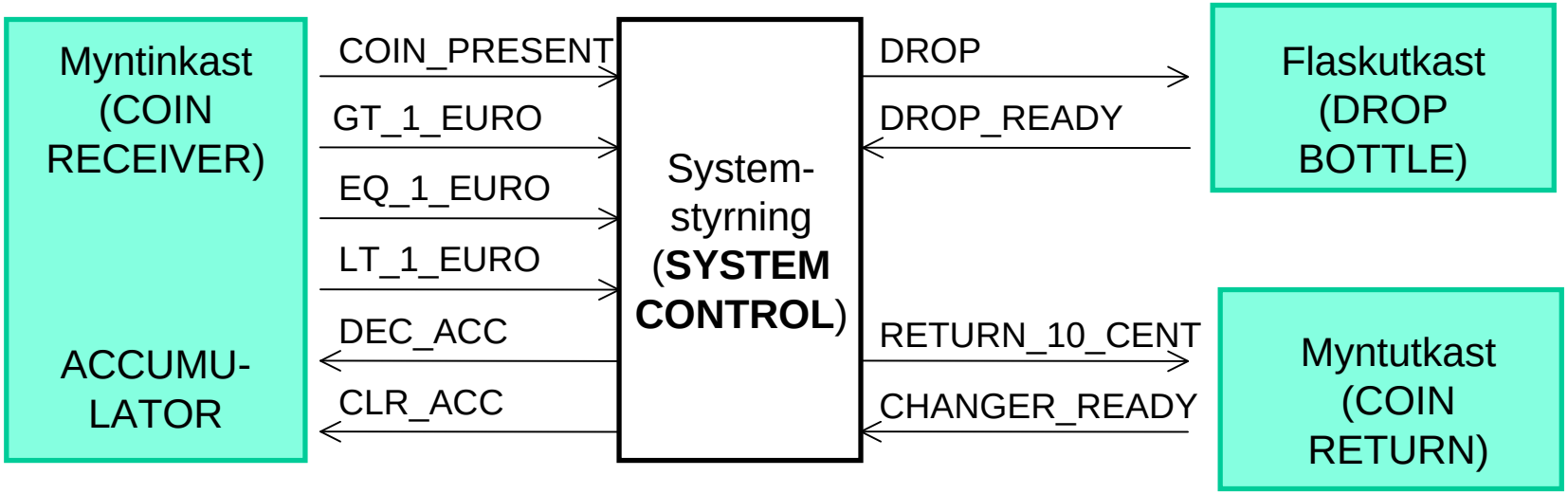
- För en Moore-automat kan vi skapa tre processer för
  - Nästa-tillståndsavkodare
  - Utgångsavkodare
  - Tillståndsregister

Interna signaler

- Moore-automaten innehåller interna signaler för
  - Nästa tillstånd
  - Nuvarande tillstånd
- Dessa signaler deklareras i *architecture*-beskrivningen

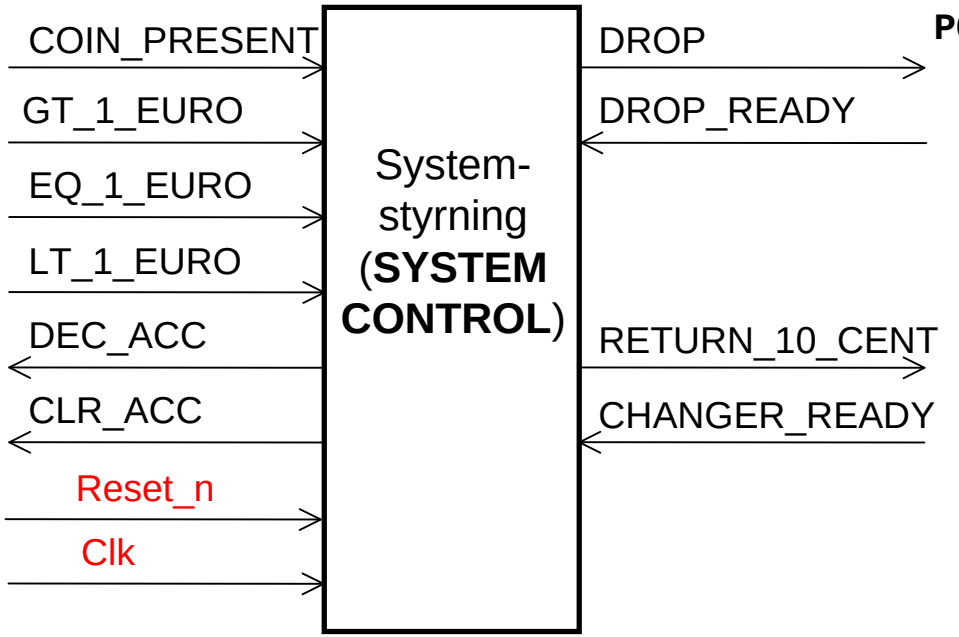
Flaskautomaten i VHDL

Vi använder flaskautomaten (systemstyrningen) från förra föreläsningen som konkret VHDL-exempel



William Sandqvist william@kth.se

Flaskautomatens **entity**



Det behövs även
Clk och Reset (aktiv låg)

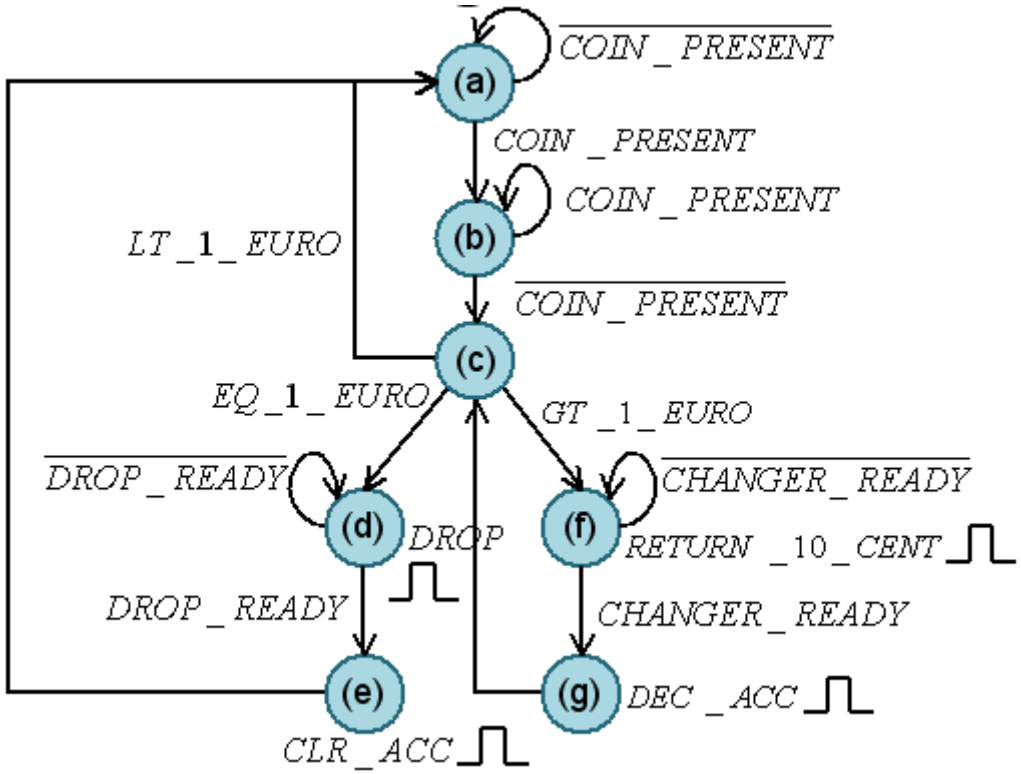
```
ENTITY Vending_Machine IS
   PORT (
      -- Inputs
      coin_present   : IN std_logic;
      gt_1_euro      : IN std_logic;
      eq_1_euro      : IN std_logic;
      lt_1_euro      : IN std_logic;
      drop_ready     : IN std_logic;
      changer_ready  : IN std_logic;
      reset_n        : IN std_logic;
      clk            : IN std_logic;
      -- Outputs
      dec_acc        : OUT std_logic;
      clr_acc        : OUT std_logic;
      drop           : OUT std_logic;
      return_10_cent : OUT std_logic);
END Vending_Machine;
```

William Sandqvist william@kth.se

Flaskautomatens **architecture**

- Arkitekturen beskriver funktionen av automaten
- Vi definierar
  - interna signaler för nuvarande och nästa tillstånd
  - tre processer för nästa-tillstånds-, utgångs-avkodare och tillståndsregister

Tillståndsdiagram



(a) Vänta på myntinkast

(b) Registrering av myntinkast

(c) Myntinkast är registrerat (3 fall)

(d) Flaskutmatning

(e) Nollställ summan

(f) Retur 10 Cent

(g) Minska summan med 10 Cent

William Sandqvist william@kth.se

Interna signaler

- Vi måste skapa en datatyp för den interna signalen
- Eftersom vi beskriver tillstånden använder vi en uppräkningstyp med värdena a,b,c,d,e,f,g
- Vi deklarerar en variabel för nuvarande tillstånd (`current_state`) och en för nästa tillstånd (`next_state`)

```
ARCHITECTURE Moore_FSM OF Vending_Machine IS
   TYPE   state_type IS (a, b, c, d, e, f, g);
   SIGNAL current_state, next_state      : state_type;
BEGIN  -- Moore_FSM
```

…

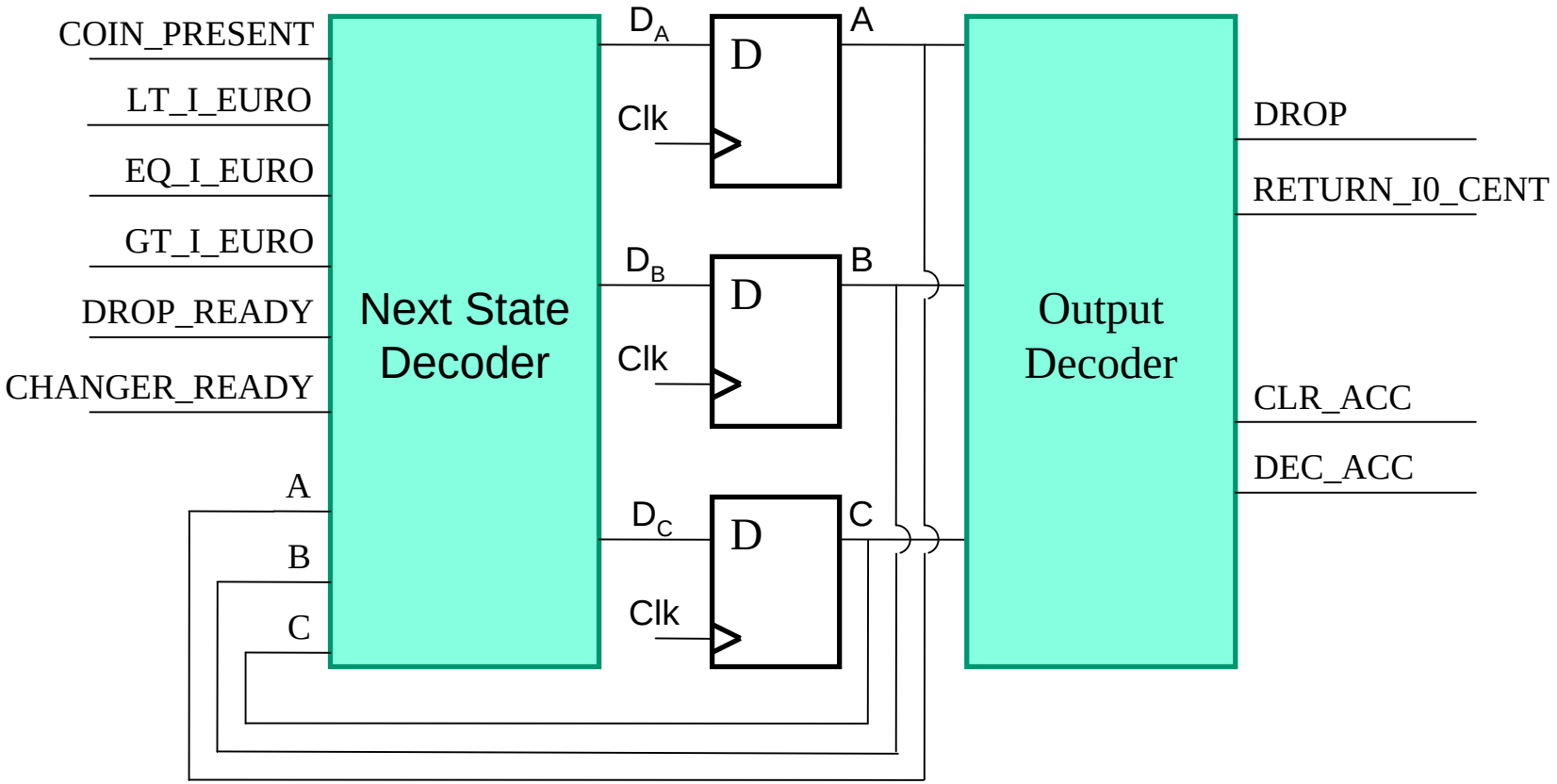
Vi vill behålla vår ”fiffiga” tillståndskod

- Om vi inte specificerar tillståndskodningen så väljer syntesverktyget kodningen
- Vi kan tvinga den till en viss kodning med attributer (**OBS! Attributer är beroende på syntesverktyget och därmed inte portabel!)**

```
ARCHITECTURE Moore_FSM OF Vending_Machine IS
   TYPE   state_type IS (a, b, c, d, e, f, g);
   -- We can use state encoding according to BV 8.4.6
   -- to enforce a particular encoding (for Quartus)
   ATTRIBUTE enum_encoding : string;
   ATTRIBUTE enum_encoding OF state_type : TYPE IS "000
  001 011 110 111 100 101";
   SIGNAL current_state, next_state      : state_type;
BEGIN  -- Moore_FSM
```

…

Blockschema



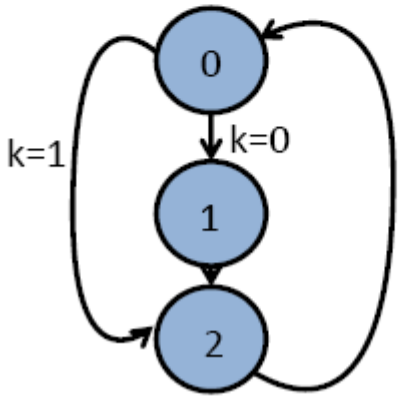
- Signalerna A,B,C beskriver nuvarande tillstånd
- Signalerna D_A, D_B, D_C beskriver nästa tillstånd

William Sandqvist william@kth.se

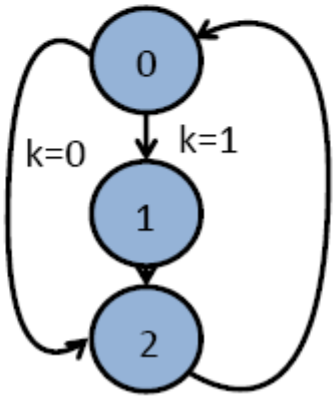
Snabbfråga

Vilken statemaskin motsvarar följande VHDL kod?

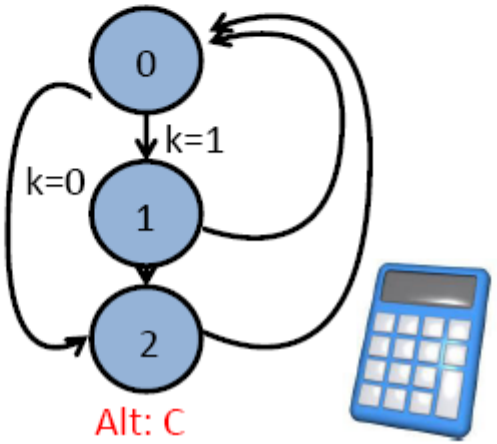
```
case state is
  when 0 =>
    if (k = '1') then
      nextstate <= 1;
    else
      nextstate <= 2;
    end if;
  when 1 => nextstate <= 2;
  when others => nextstate <= 0;
end case;
```



Alt: A



Alt: B



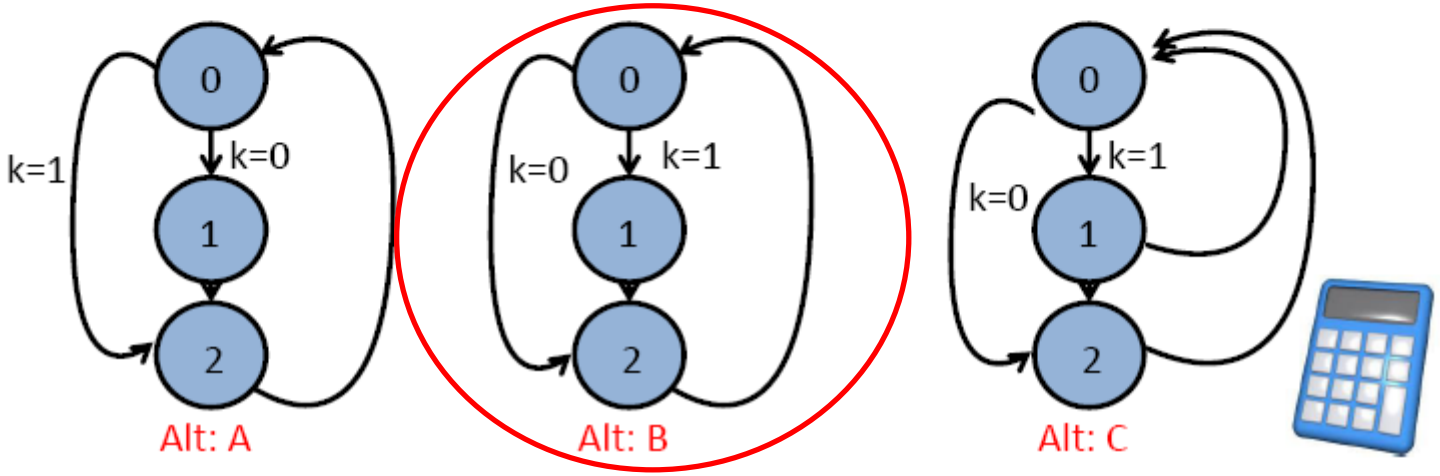
Alt: C

William Sandqvist william@kth.se

Snabbfråga

Vilken statemaskin motsvarar följande VHDL kod?

```
case state is
  when 0 =>
    if (k = '1') then
      nextstate <= 1;
    else
      nextstate <= 2;
    end if;
  when 1 => nextstate <= 2;
  when others => nextstate <= 0;
end case;
```



William Sandqvist william@kth.se

Next-State-Decoder

- Next-State-Decoder beskrivs som process
- Sensitivity list innehåller alla insignaler som ‘aktiverar’ processen

Next-State-Decoder

- I vanliga fall innehåller sensitivity listan alla ingångar till processen

```
NEXTSTATE : PROCESS (current_state, coin_present,
   gt_1_euro, eq_1_euro, lt_1_euro, drop_ready,
   changer_ready) -- Sensitivity List
    BEGIN  -- PROCESS NEXT_STATE
        …
```

Next-State-Decoder

- Vi använder nu en CASE-sats för att beskriva för varje tillstånd villkoren för tillståndsändring till nästa tillstånd

```
...
CASE current_state IS
   WHEN a => IF coin_present = '1' THEN
                next_state <= b;
             ELSE
                next_state <= a;
             END IF;
   WHEN b => IF coin_present = '0' THEN
                next_state <= c;
             ELSE
                next_state <= b;
             END IF;
```

Next-State-Decoder

- Vi kan förenkla beskrivningen genom att ange ett default-värde för nästa tillstånd

```
…
next_state <= current_state;
CASE current_state IS
   WHEN a => IF coin_present = '1' THEN
                next_state <= b;
             END IF;
   WHEN b => IF coin_present = '0' THEN
                next_state <= c;
             END IF;
…
```

Det är viktigt att vi anger alla alternativ för `next_state` signalen. Annars får vi implicit en sats `next_state <= next_state` som genererar en latch.

Next-State-Decoder

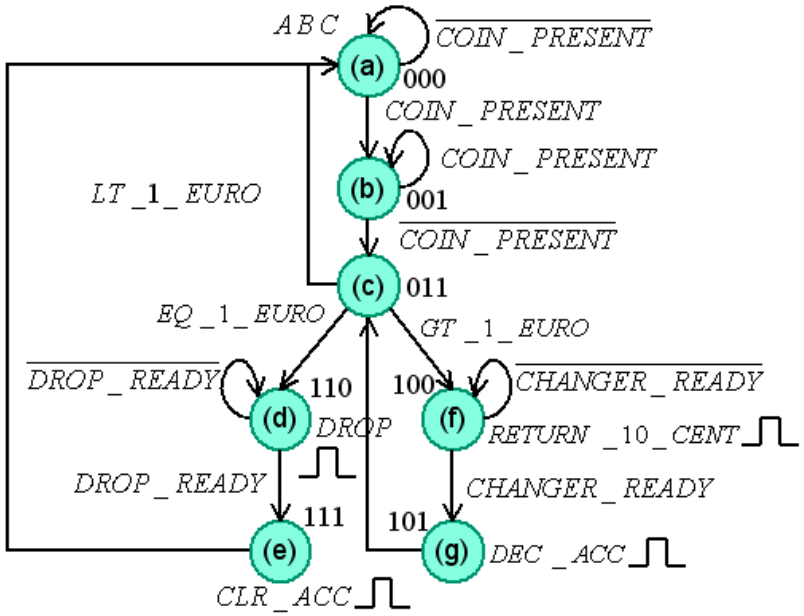
- Vi avsluta CASE-satsen med en WHEN OTHERS sats. Här anger vi att vi ska gå till tillstånd a om vi hamnar i ett ospecificierat tillstånd

```
    …
        WHEN g      => next_state <= c;
        WHEN OTHERS => next_state <= a;
    END CASE;
END PROCESS NEXTSTATE;
```

Utgångsavkodaren

- Utgångsavkodaren beskrivs som en egen process
- Sensitivity-listan innehåller bara nuvarande tillstånd eftersom utgångarna bara är beroende av tillståndet

Utgångsavkodaren




```
OUTPUT : PROCESS (current_state)
   BEGIN  -- PROCESS OUTPUT
      drop           <= '0';
      clr_acc        <= '0';
      dec_acc        <= '0';
      return_10_cent <= '0';
      CASE current_state IS
         WHEN d      => drop           <= '1';
         WHEN e      => clr_acc        <= '1';
         WHEN f      => return_10_cent <= '1';
         WHEN g      => dec_acc        <= '1';
         WHEN OTHERS => NULL;
      END CASE;
   END PROCESS OUTPUT;
```

Tillståndsregistret

- Tillståndsregistret modelleras som en synkron process med asynkron reset (aktiv låg)

```
CLOCK : PROCESS (clk, reset_n)
   BEGIN  -- PROCESS CLOCK
      IF reset_n = '0' THEN -- asynchronous reset
   (active low)
         current_state <= a;
      ELSIF clk'event AND clk = '1' THEN  --
   rising clock edge
         current_state <= next_state;
      END IF;
   END PROCESS CLOCK;
```

Mealy-automat?

- En Mealy-automat kan modelleras på samma sätt som Moore-automaten
- Skillnaden är att utgångsavkodaren också är beroende av insignalerna
- Processen som modellerar utgångssignalerna behöver också ha insignalerna i sin sensitivity list!

Mer om VHDL

- Kodexemplet för flaskautomaten finns på kurshemsidan
- Titta på studiematerialet om “VHDL-syntes” på kurshemsidan
- Både Brown/Vranesic- och Hemert-boken innehåller kodexempel

William Sandqvist william@kth.se

Inför Lab 3 VHDL-intro

ALTERA Quartus II och ModelSim

Observera! Det är ett mycket omfattande förberedelsearbete inför lab 3, börja arbetet i tid!

PLD_CPLD_FPGA.pdf

- *Installera programmen på din dator (låna USB med installationsprogram)*
  (Drivrutin för USB-blaster)
- *VHDL-program med Quartus*
- *VHDL för ett kodlås*
- *Pin-planering i Quartus*
- *Chip-programmering med Quartus*

CodelockTutorial.pdf *codelockVHDL.pdf*

- *Simulera med ModelSim*
- *Testbänk i ModelSim*

testbench.pdf

William Sandqvist william@kth.se

Inför Lab 3 VHDL-intro

Förbered Labprogrammet hemma (VHDL). Kontrollera att det går att kompilera. Tag med dig koden till skolan på något sätt, tex.:

- Maila koden som text till dig själv.
- Ta med dig ett USB-minne med koden som en textfil.
- Har Du möjlighet att simulera koden hemma ökar Du sannolikheten för att Du har en korrekt kod att utgå från i skolan.

Tiden vid laborationen i skolan kommer *inte* att räcka till att skriva program-koden från "scratch"!

William Sandqvist william@kth.se

Kodlås – klassiskt exempel!

Moore *– ”Gedanken Experiments” on Sequential Machines* ***1956***

The only way to make this give a 1 output is putting it into state q_n and this will be said to be unlocking the combination lock. If the combination lock is originally in state q_1, it can be unlocked only by giving it exactly the proper input sequence for the last n-1 steps before unlocking it. This input sequence is, of course, called the combination of the lock. The machine H is an example of a combination lock having the combination 0,1,0:

Machine H

Present State		
Previous State	Previous Input	
	0	1
q_1	q_2	q_1
q_2	q_1	q_3
q_3	q_4	q_1
q_4	q_1	q_1

Present State	Present Input
q_1	0
q_2	0
q_3	0
q_4	1

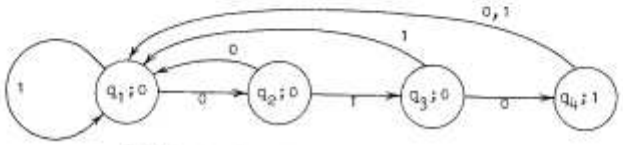


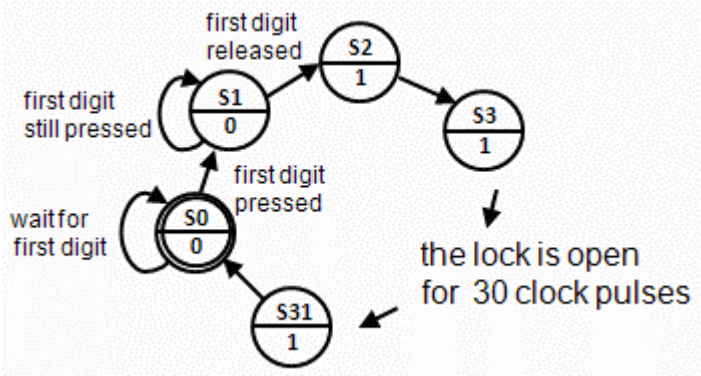
FIGURE 9. Transition Diagram of Machine H

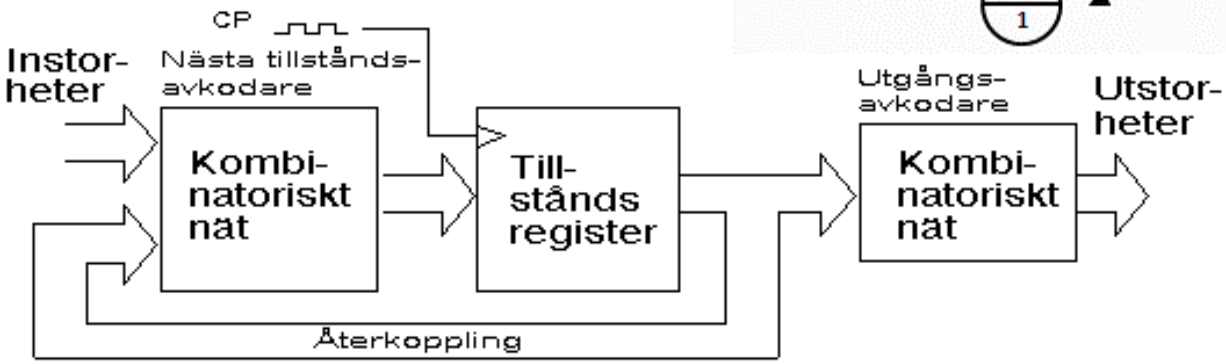
Kombinationslås 0-1-0
det exemplet finns med i Moores klassiska uppsats från 1956.

William Sandqvist william@kth.se

Laborationsuppgift - kodlås

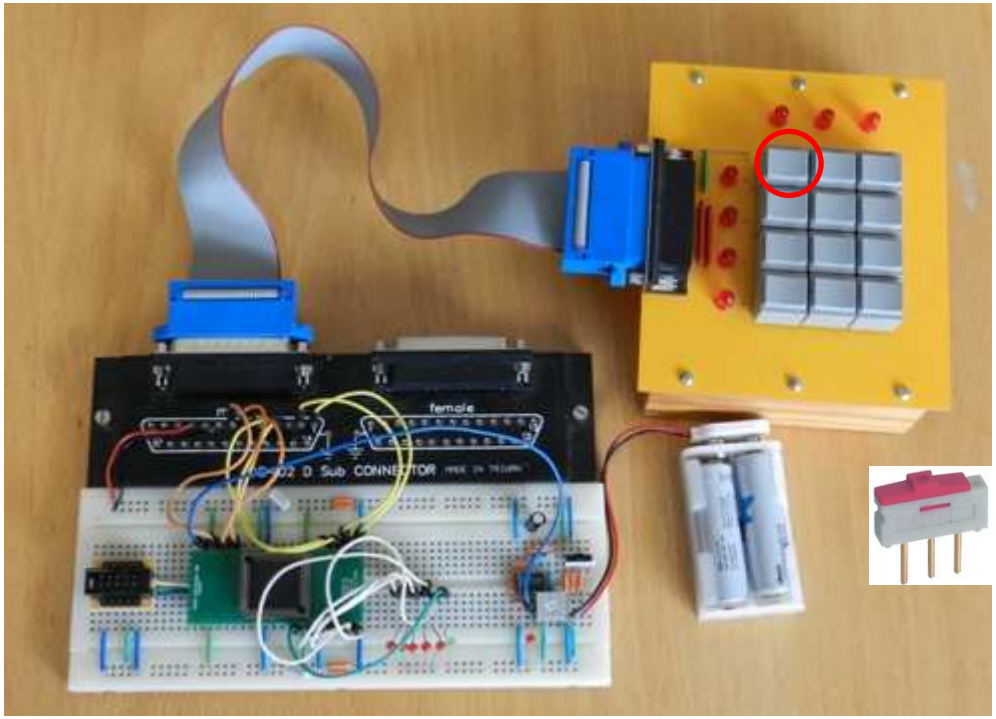
- ***Uppgift:*** *att skriva VHDL kod för ett kodlås som öppnas med koden ”de fyra sista siffrorna i ditt personnummer”.*
- ***Ledning:*** *en VHDL ”mall” för ett förenklat kodlås som öppnas med koden ”siffran ett”.*





William Sandqvist william@kth.se

Mall-program



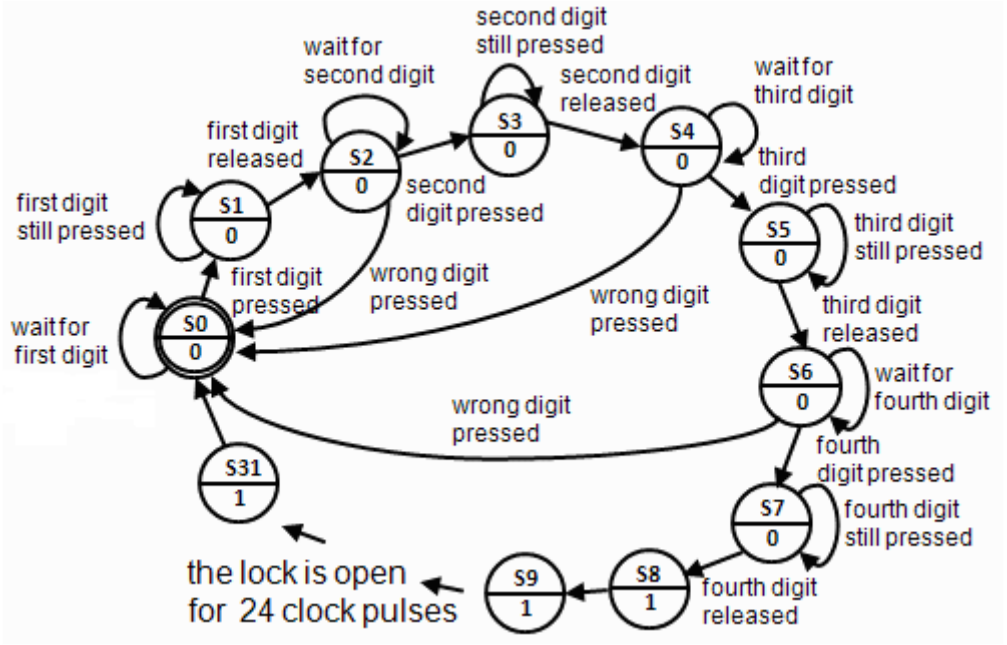
Power On/Off

Mall-programmet gäller ett förenklat kodlås som öppnar för tangenten ”1”, lite väl enkelt kan nog tyckas … !

William Sandqvist william@kth.se

Öppna låset med ditt personnummer!

- ***Nu är det dags att skriva om VHDL-koden så att låset öppnar för de fyra sista siffrorna i ditt personnummer!***



(*Om Du förbereder koden för ditt personnummer, så kan två i en laborationsgrupp bidraga med hälften av koden var vid laborationen*).

William Sandqvist william@kth.se

William Sandqvist william@kth.se