# Registers: Counters

Prof. Daniele Gorla

## Counters

A *counter* is a register used to count the number of occurrences of a certain event, always modulo some natural number.

→ if it is made up by *n* FFs, it can count up to modulo $2^n$

Tipically, the countable events are clock's impulses or the occurrences of some Input values or sequences.

We have two kinds of counters:

- synchronous (all FFs of the counter have the same clock)
- asynchronous (in the same counters, FFs have different clocks)

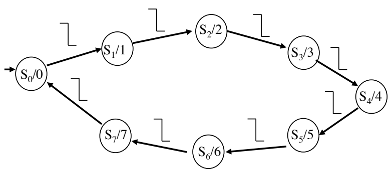
They can count upwise or downwise (or both)

They can be set to a value that does not respect the attended counting sequence.



## Synthesis of the upwise counter modulo 8 (1)

A *counter modulo 8* starts from 0 and at every descending wave front of the clock increments its value by 1, until it arrives at 7; then, it returns to 0 and starts again.




Binary encoding of the automaton:

- State $S_i$ is associated to the binary coding of *i* → 3 bits → 3 FFs
- There is no input alphabeth
- Output characters are codified with their normal binary coding.



## Synthesis of the upwise counter modulo 8 (2)

| State(t) | State(t+1) |
|---|---|
| $S_0$ | $S_1$ |
| $S_1$ | $S_2$ |
| $S_2$ | $S_3$ |
| $S_3$ | $S_4$ |
| $S_4$ | $S_5$ |
| $S_5$ | $S_6$ |

| $y_2$ $y_1$ $y_0$ | $Y_2$ $Y_1$ $Y_0$ | $J_2$ $K_2$ | $J_1$ $K_1$ | $J_0$ $K_0$ |
|---|---|---|---|---|
| 0 0 0 | 0 0 1 | 0 - | 0 - | 1 - |
| 0 0 1 | 0 1 0 | 0 - | 1 - | - 1 |
| 0 1 0 | 0 1 1 | 0 - | - 0 | 1 - |
| 0 1 1 | 1 0 0 | 1 - | - 1 | - 1 |
| 1 0 0 | 1 0 1 | - 0 | 0 - | 1 - |
| 1 0 1 | 1 1 0 | - 0 | 1 - | - 1 |
| 1 1 0 | 1 1 1 | - 0 | - 0 | 1 - |
| 1 1 1 | 0 0 0 | - 1 | - 1 | - 1 |

J2 map

| y2 y1 \ y0 | 0 | 1 |
|---|---|---|
| 00 | 0 | 0 |
| 01 | 0 | 1 |
| 11 | X | X |
| 10 | X | X |

J1 map

| y2 y1 \ y0 | 0 | 1 |
|---|---|---|
| 00 | 0 | 1 |
| 01 | X | X |
| 11 | X | X |
| 10 | 0 | 1 |

J0 map

| y2 y1 \ y0 | 0 | 1 |
|---|---|---|
| 00 | 1 | X |
| 01 | 1 | X |
| 11 | 1 | X |
| 10 | 1 | X |

K2 map

| y2 y1 \ y0 | 0 | 1 |
|---|---|---|
| 00 | X | X |
| 01 | X | X |
| 11 | 0 | 1 |
| 10 | 0 | 0 |

K1 map

| y2 y1 \ y0 | 0 | 1 |
|---|---|---|
| 00 | X | X |
| 01 | 0 | 1 |
| 11 | 0 | 1 |
| 10 | X | X |

K0 map

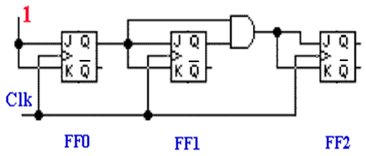
| y2 y1 \ y0 | 0 | 1 |
|---|---|---|
| 00 | X | 1 |
| 01 | X | 1 |
| 11 | X | 1 |
| 10 | X | 1 |

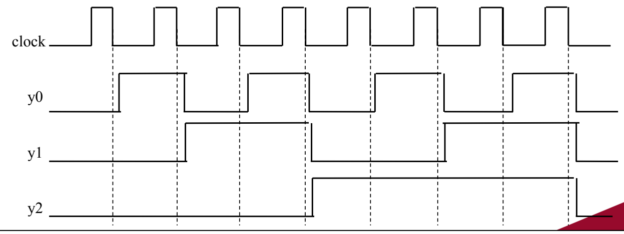
$J_0 = K_0 = 1$

$J_1 = K_1 = y_0$

$J_2 = K_2 = y_1 y_0$

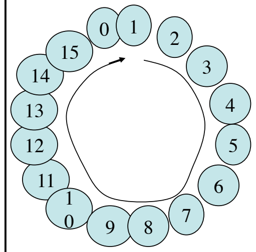
## The upwise counter modulo 8



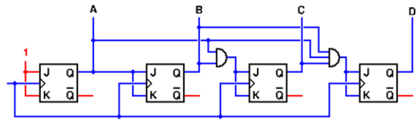



## The upwise counter modulo 16



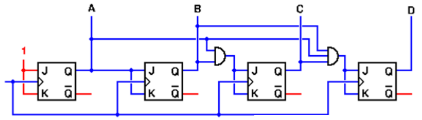

| y3 y2 y1 y0 | Y3 Y2 Y1 Y0 | J3 K3 | J2 K2 | J1 K1 | J0 K0 |
|---|---|---|---|---|---|
| 0 0 0 0 | 0 0 0 1 | 0 - | 0 - | 0 - | 1 - |
| 0 0 0 1 | 0 0 1 0 | 0 - | 0 - | 1 - | - 1 |
| 0 0 1 0 | 0 0 1 1 | 0 - | 0 - | - 0 | 1 - |
| 0 0 1 1 | 0 1 0 0 | 0 - | 1 - | - 1 | - 1 |
| 0 1 0 0 | 0 1 0 1 | 0 - | - 0 | 0 - | 1 - |
| 0 1 0 1 | 0 1 1 0 | 0 - | - 0 | 1 - | - 1 |
| 0 1 1 0 | 0 1 1 1 | 0 - | - 0 | - 0 | 1 - |
| 0 1 1 1 | 1 0 0 0 | 1 - | - 1 | - 1 | - 1 |
| 1 0 0 0 | 1 0 0 1 | - 0 | 0 - | 0 - | 1 - |
| 1 0 0 1 | 1 0 1 0 | - 0 | 0 - | 1 - | - 1 |
| 1 0 1 0 | 1 0 1 1 | - 0 | 0 - | - 0 | 1 - |
| 1 0 1 1 | 1 1 0 0 | - 0 | 1 - | - 1 | - 1 |
| 1 1 0 0 | 1 1 0 1 | - 0 | - 0 | 0 - | 1 - |
| 1 1 0 1 | 1 1 1 0 | - 0 | - 0 | 1 - | - 1 |
| 1 1 1 0 | 1 1 1 1 | - 0 | - 0 | - 0 | 1 - |
| 1 1 1 1 | 0 0 0 0 | - 1 | - 1 | - 1 | - 1 |

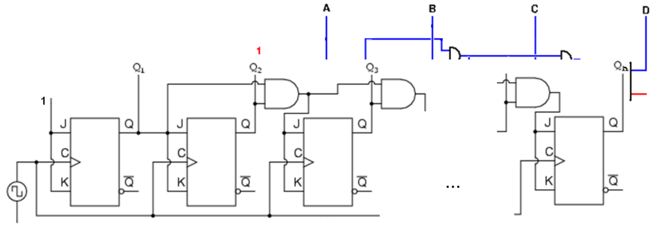
$J_0 = K_0 = 1$
$J_1 = K_1 = y_0$
$J_2 = K_2 = y_1y_0$
$J_3 = K_3 = y_2y_1y_0$






## The upwise counter modulo $2^n$

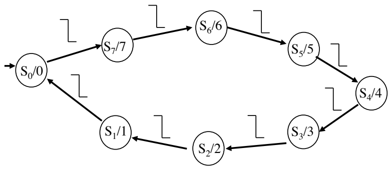



$J_0 = K_0 = 1 \qquad J_{i+1} = K_{i+1} = J_i \text{ AND } Q_i$

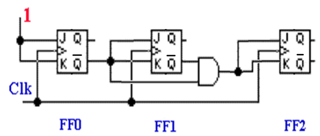


## The downwise counter modulo 8



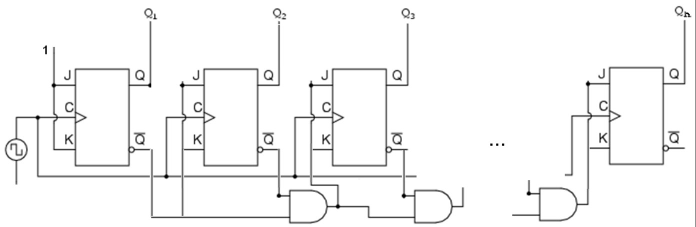

| $y_2$ $y_1$ $y_0$ | $Y_2$ $Y_1$ $Y_0$ | $J_2$ $K_2$ $J_1$ $K_1$ $J_0$ $K_0$ |
|---|---|---|
| 0 0 0 | 1 1 1 | 1 - 1 - 1 - |
| 0 0 1 | 0 0 0 | 0 - 0 - - 1 |
| 0 1 0 | 0 0 1 | 0 - - 1 1 - |
| 0 1 1 | 0 1 0 | 0 - - 0 - 1 |
| 1 0 0 | 0 1 1 | - 1 1 - 1 - |
| 1 0 1 | 1 0 0 | - 0 0 - - 1 |
| 1 1 0 | 1 0 1 | - 0 - 1 1 - |
| 1 1 1 | 1 1 0 | - 0 - 0 - 1 |

$J_0 = K_0 = 1$
$J_1 = K_1 = \overline{y}_0$
$J_2 = K_2 = \overline{y}_1\overline{y}_0$



## The downwise counter modulo $2^n$

SAPIENZA Università di Roma Dipartimento di Informatica

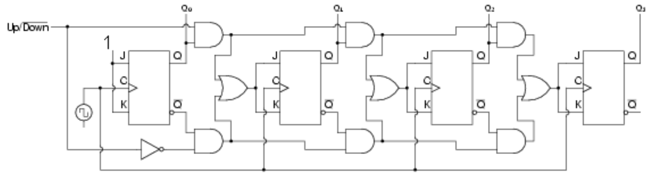



$J_0 = K_0 = 1 \qquad J_{i+1} = K_{i+1} = J_i \text{ AND } \overline{Q}_i$

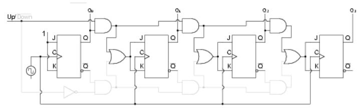


## Bidirectional Counter modulo $2^n$

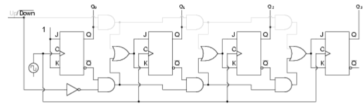
SAPIENZA Università di Roma Dipartimento di Informatica




Upwise Counter: Up = 1



Downwise Counter: Up = 0





## (Upwise) Counter modulo *k* (≠ $2^n$)

SAPIENZA Università di Roma Dipartimento di Informatica

2 ways:
1. a synthesis procedure for every *k*
2. a modular solution, that however uses FFs with asynchronous inputs (see later)

Ex.: counter modulo 5

| State(t) | State(t+1) |
|---|---|
| $S_0$ | $S_1$ |
| $S_1$ | $S_2$ |
| $S_2$ | $S_3$ |
| $S_3$ | $S_4$ |
| $S_4$ | $S_0$ |

$J_0 = \overline{y}_2$

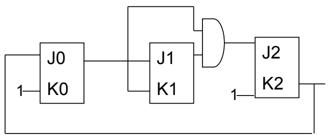
$K_0 = 1$

$J_1 = K_1 = y_0$

$J_2 = y_1 y_0$

$K_2 = 1$

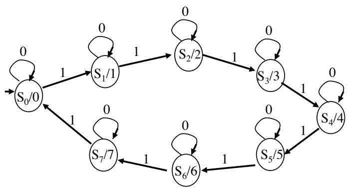
| $y_2\ y_1\ y_0$ | $Y_2\ Y_1\ Y_0$ | $J_2\ K_2\quad J_1\ K_1\quad J_0\ K_0$ |
|---|---|---|
| 0 0 0 | 0 0 1 | 0 - 0 - 1 - |
| 0 0 1 | 0 1 0 | 0 - 1 - - 1 |
| 0 1 0 | 0 1 1 | 0 - - 0 1 - |
| 0 1 1 | 1 0 0 | 1 - - 1 - 1 |
| 1 0 0 | 0 0 0 | - 1 0 - 0 - |
| 1 0 1 | - - - | - - - - - - |
| 1 1 0 | - - - | - - - - - - |
| 1 1 1 | - - - | - - - - - - |






## Counter of input signals

SAPIENZA Università di Roma Dipartimento di Informatica




| x y2 y1 y0 | Y2 Y1 Y0 | J2 K2 | J1 K1 | J0 K0 |
|---|---|---|---|---|
| 0 0 0 0 | 0 0 0 | 0 - | 0 - | 0 - |
| 0 0 0 1 | 0 0 1 | 0 - | 0 - | - 0 |
| 0 0 1 0 | 0 1 0 | 0 - | - 0 | 0 - |
| 0 0 1 1 | 0 1 1 | 0 - | - 0 | - 0 |
| 0 1 0 0 | 1 0 0 | - 0 | 0 - | 0 - |
| 0 1 0 1 | 1 0 1 | - 0 | 0 - | - 0 |
| 0 1 1 0 | 1 1 0 | - 0 | - 0 | 0 - |
| 0 1 1 1 | 1 1 1 | - 0 | - 0 | - 0 |
| 1 0 0 0 | 0 0 1 | 0 - | 0 - | 1 - |
| 1 0 0 1 | 0 1 0 | 0 - | 1 - | - 1 |
| 1 0 1 0 | 0 1 1 | 0 - | - 0 | 1 - |
| 1 0 1 1 | 1 0 0 | 1 - | - 1 | - 1 |
| 1 1 0 0 | 1 0 1 | - 0 | 0 - | 1 - |
| 1 1 0 1 | 1 1 0 | - 0 | 1 - | - 1 |
| 1 1 1 0 | 1 1 1 | - 0 | - 0 | 1 - |
| 1 1 1 1 | 0 0 0 | - 1 | - 1 | - 1 |

*The same as the one for the counter modulo 8*

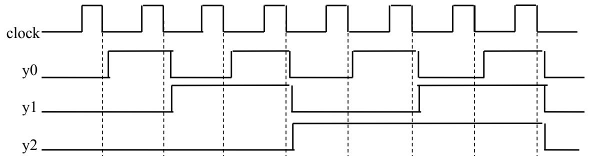
$J_0 = K_0 = x1$

$J_1 = K_1 = xy_0$

$J_2 = K_2 = xy_1y_0$

## Asynchronous Counter MOD 8

SAPIENZA Università di Roma Dipartimento di Informatica




OBS.: FF0 commutes at every descending wave front of the clock;
FF1 commutes at every descending wave front of FF0;
FF2 commutes at every descending wave front of FF1.

We can then design a different counter MOD 8 where

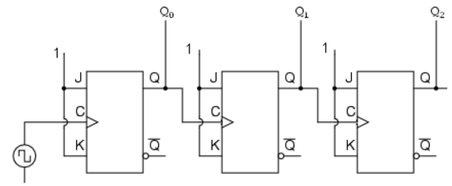
- all FFs are in toggle modality (J = K = 1)
- FF0 uses as clock the *clock* signal;
- FF1 uses as clock $y_0$;
- FF2 uses as clock $y_1$.

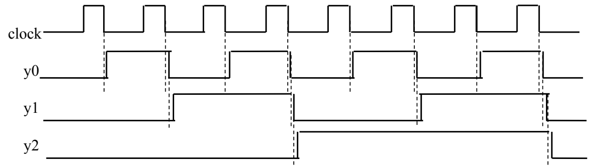
We call such a counter *asynchronous* because the FFs are not synchronized on the same clock (notice however that **this is still a synchronous circuit**, because a clock is present and FFs commute only at precise moments in time).



## Implementation and temporal diagram of the asynchronous counter MOD 8

SAPIENZA Università di Roma Dipartimento di Informatica

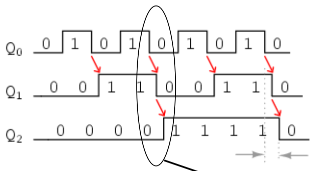


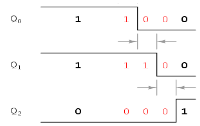



## Delay propagation

SAPIENZA Università di Roma Dipartimento di Informatica




Since FFs are not syncrhonized on the same clock, commutation delays sum one with the other and yield, for very few moments, to sequences out from the normal counting (*false counts*). This phenomenon is called *ripple*.

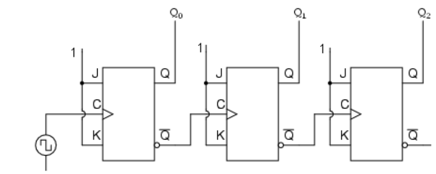


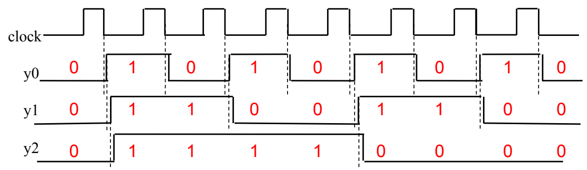

In almost all applications, this effect is negligible, since delays are very small.



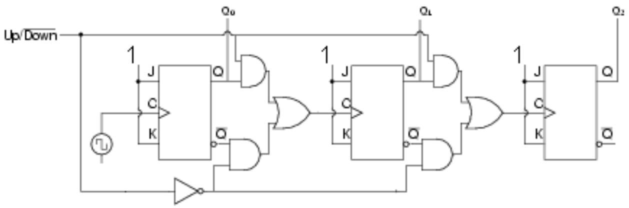
## Downwise Asynchronous Counter MOD 8

SAPIENZA Università di Roma Dipartimento di Informatica



## Bidirectional Asynchronous Counter

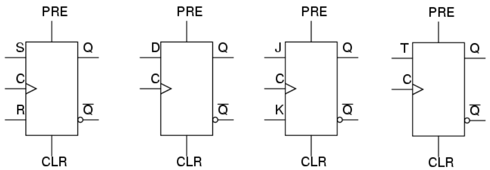







## FF with asynchronous inputs

Sometimes, FFs are equipped with two further inputs, called PRESET and CLEAR, that work in an *asynchronous* way w.r.t. the clock: i.e., they are used to set ot reset the FF in an instantaneous way (independently from the usual inputs and from the *clock*).




Behaviour:

- PRESET = CLEAR = 0: usual FF;
- PRESET = 1, CLEAR = 0: immediate set of the FF;
- PRESET = 0, CLEAR = 1: immediate reset of the FF;
- PRESET = CLEAR = 1: not used.

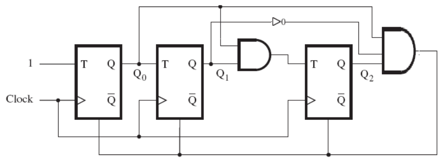


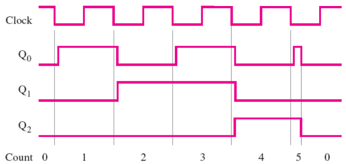


## A modular counter MOD *k* (≠ $2^n$)

Idea: when passing from *k*–1 to *k*, we reset the counter through CLEARs
→ as soon as the counter stores (the binary coding of) *k*, we activate the CLEAR of all FFs for a very small time interval

Ex.: counter MOD 5



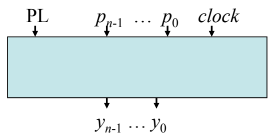

Same idea can be used for downwise, bidirectional or asynchronous counters





## Presettable Counters

A second important use of FFs with asynchronous inputs is to build *presettable counters*, where we can force (and maintain) a value out of the normal counting sequence, independently of the inputs and the clock.




Behaviour:

- If PL (= *parallel load*) holds 0, then it behaves like a normal counter MOD $2^n$;
- If PL = 1, it immediately stores the values on $p_{n-1},\dots,p_0$ (*parallel data inputs*) in the respective FFs;
- To store a value, we can simply keep that value on the parallel data inputs and keep PL = 1.

## A presettable upwards counter MOD 8

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA