FPGA Design

Part I - Hardware Components

Thomas Lenzi

Vrije
Universiteit
Brussel

UNIVERSITÉ
LIBRE
DE BRUXELLES

Approach

- We believe that having knowledge of the hardware components that compose an FPGA allow for better firmware design.

- Being able to visualise the ressources that a code will exploit helps to design more efficiently and possibly with less bugs or unexpected behaviours.

- Therefore we will start from the bottom and work our way up.

Objective

- Our goal is to understand the functioning of an FPGA in order to program it.

- We will start by considering that the FPGA is a black box, an electronic component that sits on a PCB and is connected to other components, but nothing more.

- Then, step-by-step, we will go over what composes an FPGA an explain how it works.

Logic Gates, Multiplexers, LUTs, …

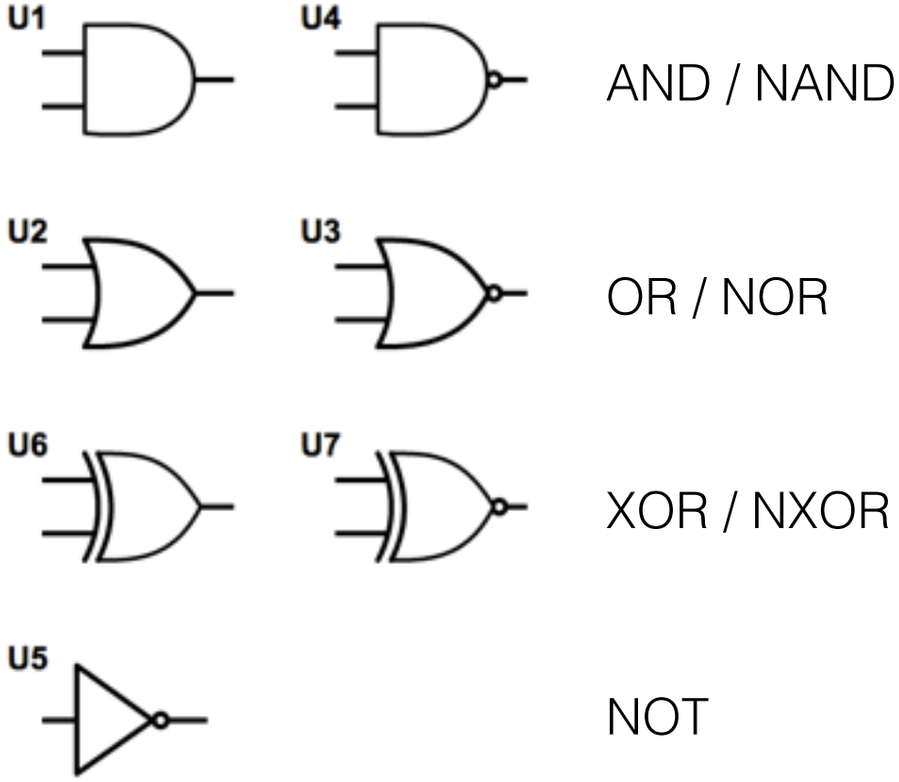
Combinatorial & Sequential Logic

- **Combinational** logic makes use of the **current** state of the inputs to define the value of the output.

- **Sequential** logic behaves according to the **current** and **past** values of the inputs by **registering** them.

Logic Gates

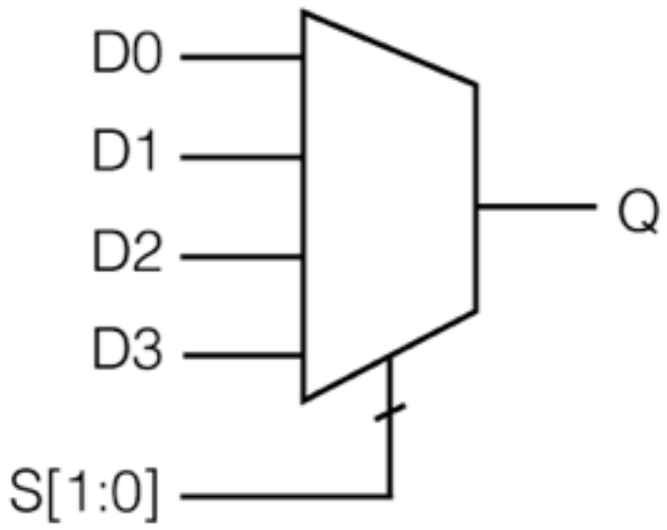
Logic gates implement boolean operations on one or more inputs.



A	B	AND	NAND	OR	NOR	XOR	XNOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

Multiplexers

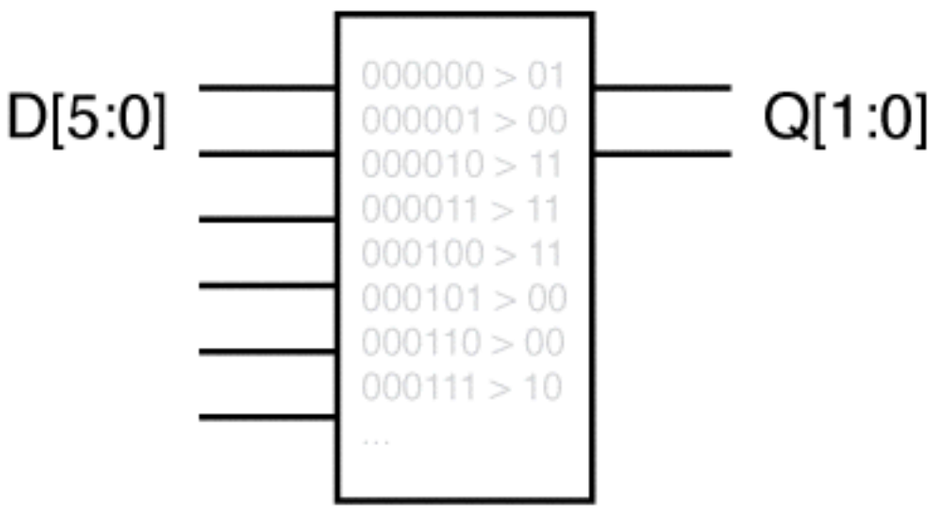
Multiplexers forward one out of several inputs to a single output according to a selection signal.



S[0]	S[1]	Q
0	0	D0
0	1	D1
1	0	D2
1	1	D3

LookUp-Tables (LUT)

LUTs are associative memories that return predefined output signals according to the state of the input signals. The values that will be returned are programmed beforehand.

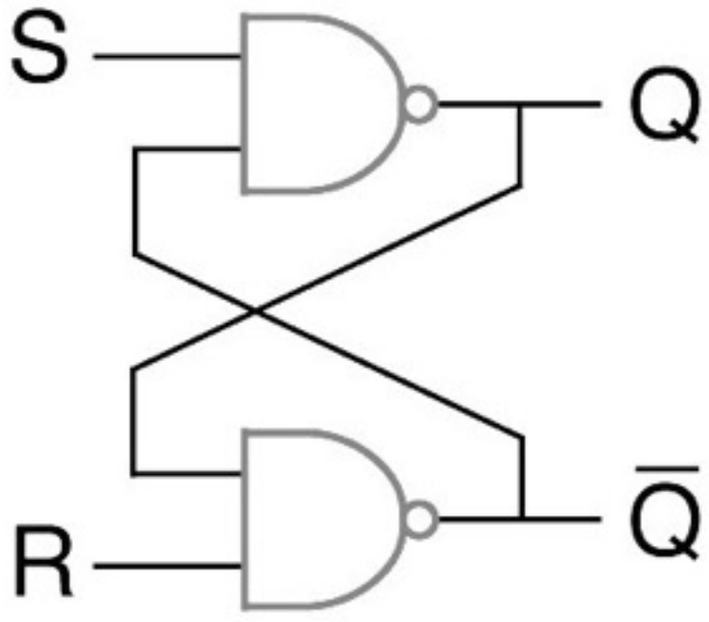


D	Q
000000	01
000001	00
000010	11
000011	11
…	…

Latches

Latches are components that maintain a defined state and are controlled by their inputs.

The SR flip-flop is the simplest example of this. It is an active low SET and RESET latch.

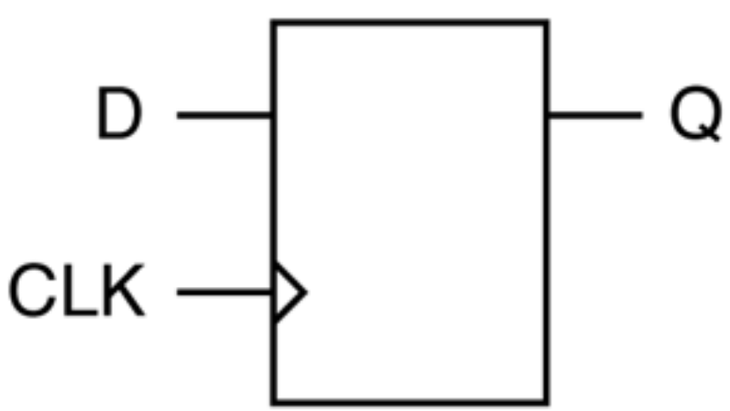


S	R	Q
0	0	Forbidden
1	0	0
0	1	1
1	1	Unchanged

Registers

Registers are latches that store “data”.

The D flip-flop (DFF) for example changes state only on the rising edge of the clock. It can therefore be seen as a sample-and-hold register.



D	CLK	Q
X	↑	X
X	↓	Unchanged
X	1	Unchanged
X	0	Unchanged

Other components

- **Buffers** isolate their input and output pins and allow for high fanout of the signal.

- **Shift registers** shift the input stream of data by a given number of bits.

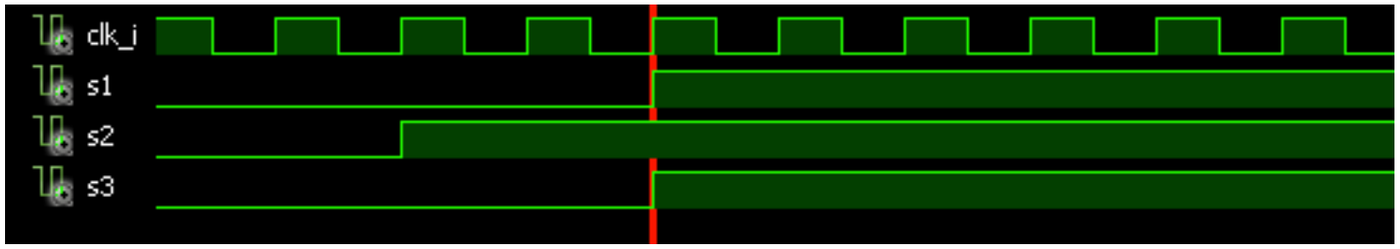
- **Serialisers / deserialisers** transform parallel (serial) data into serial (parallel) data.

- **Adders** and **Multipliers** are dedicated components that are optimised to perform mathematical operations.

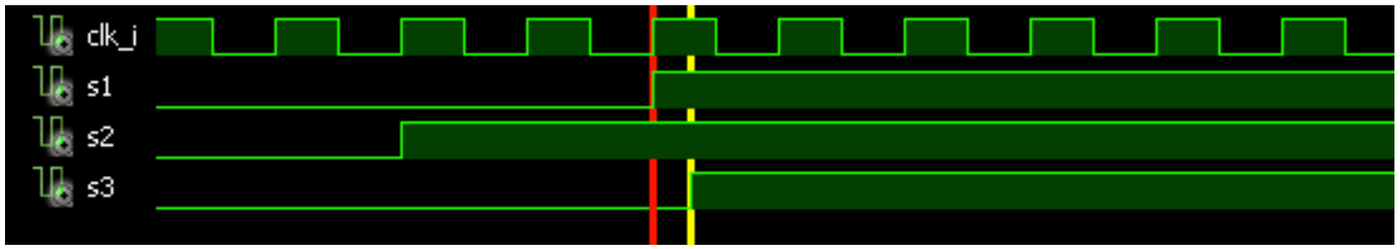
Delays

- Every component in the design adds delay to the signal.

Ideal behaviour

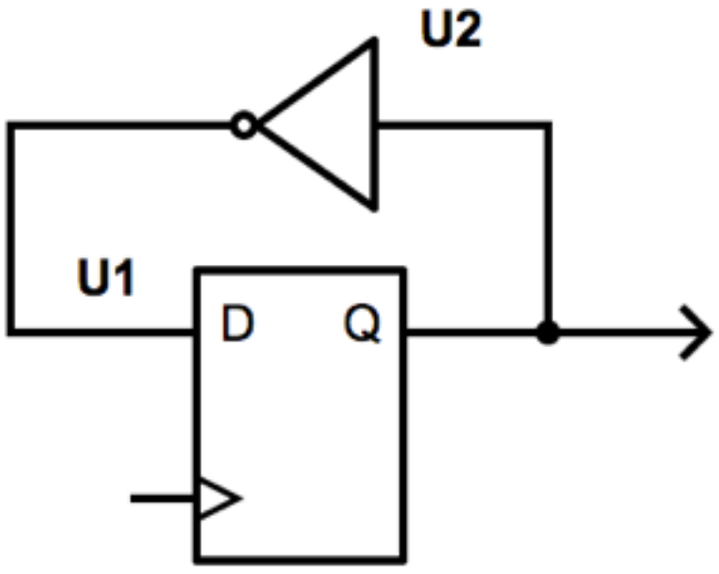


Real behaviour

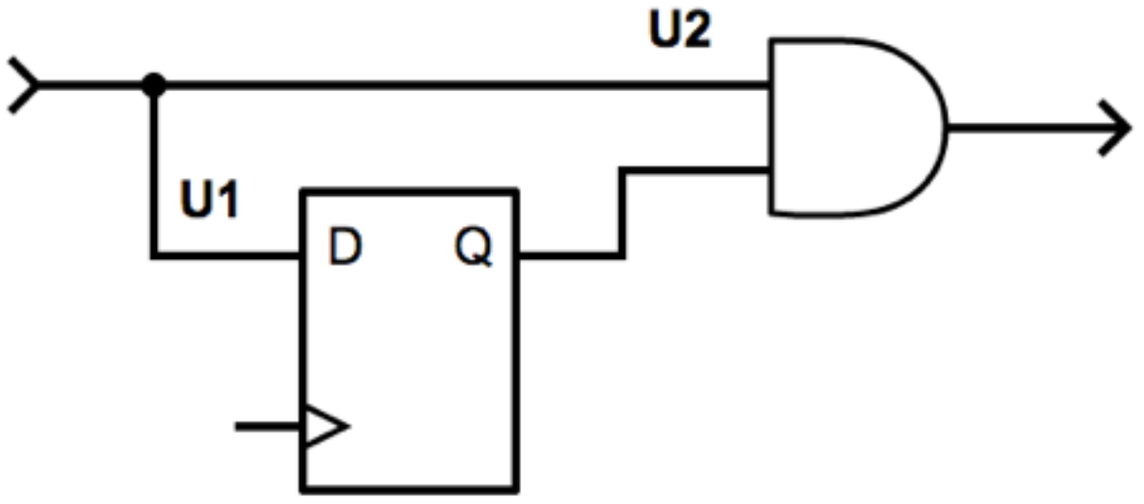


- This is OK as long as you are aware of it and take it into account when you design your firmware.

Examples

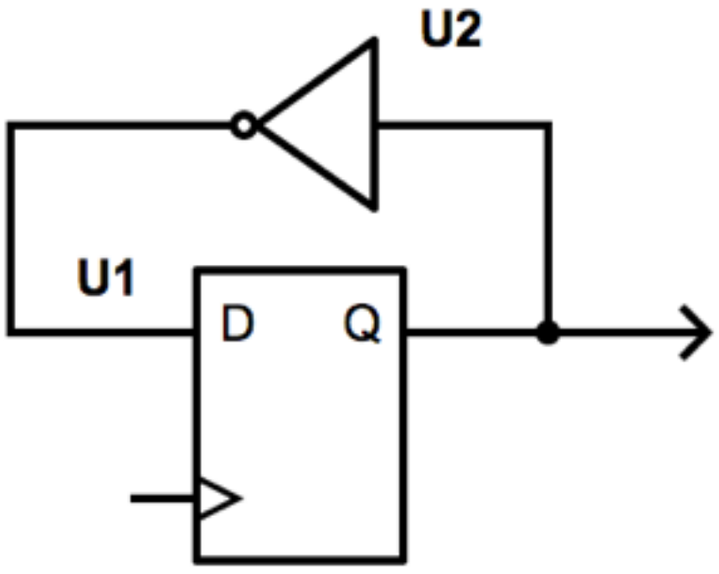


What is the output of this circuit ?

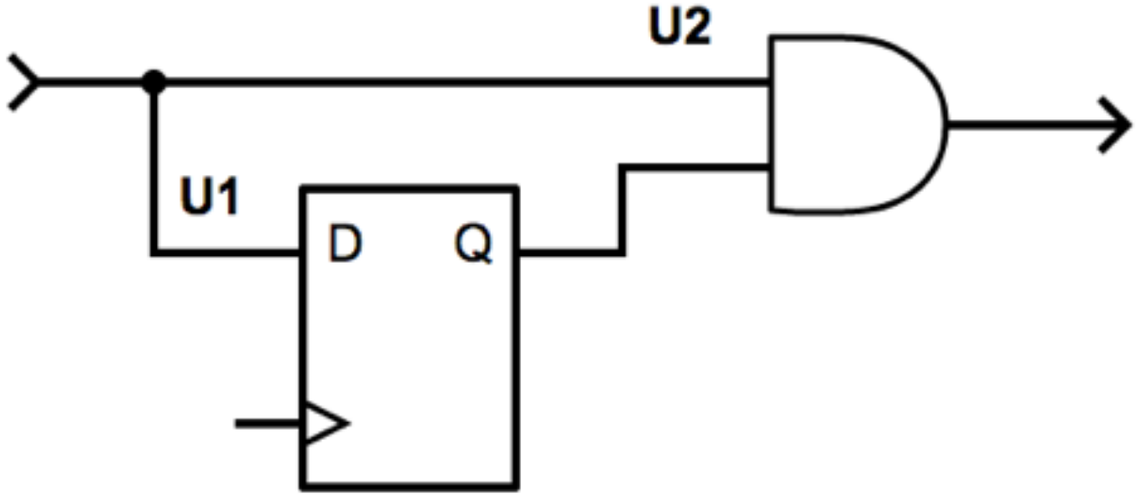


What is the behaviour of this circuit ?

Examples



Continuously inverts the output



Outputs 1 if the input is 1 for two consecutive clock cycles

Exercises

1. Using only logic gates, write the schematic of the 4-inputs multiplexer.

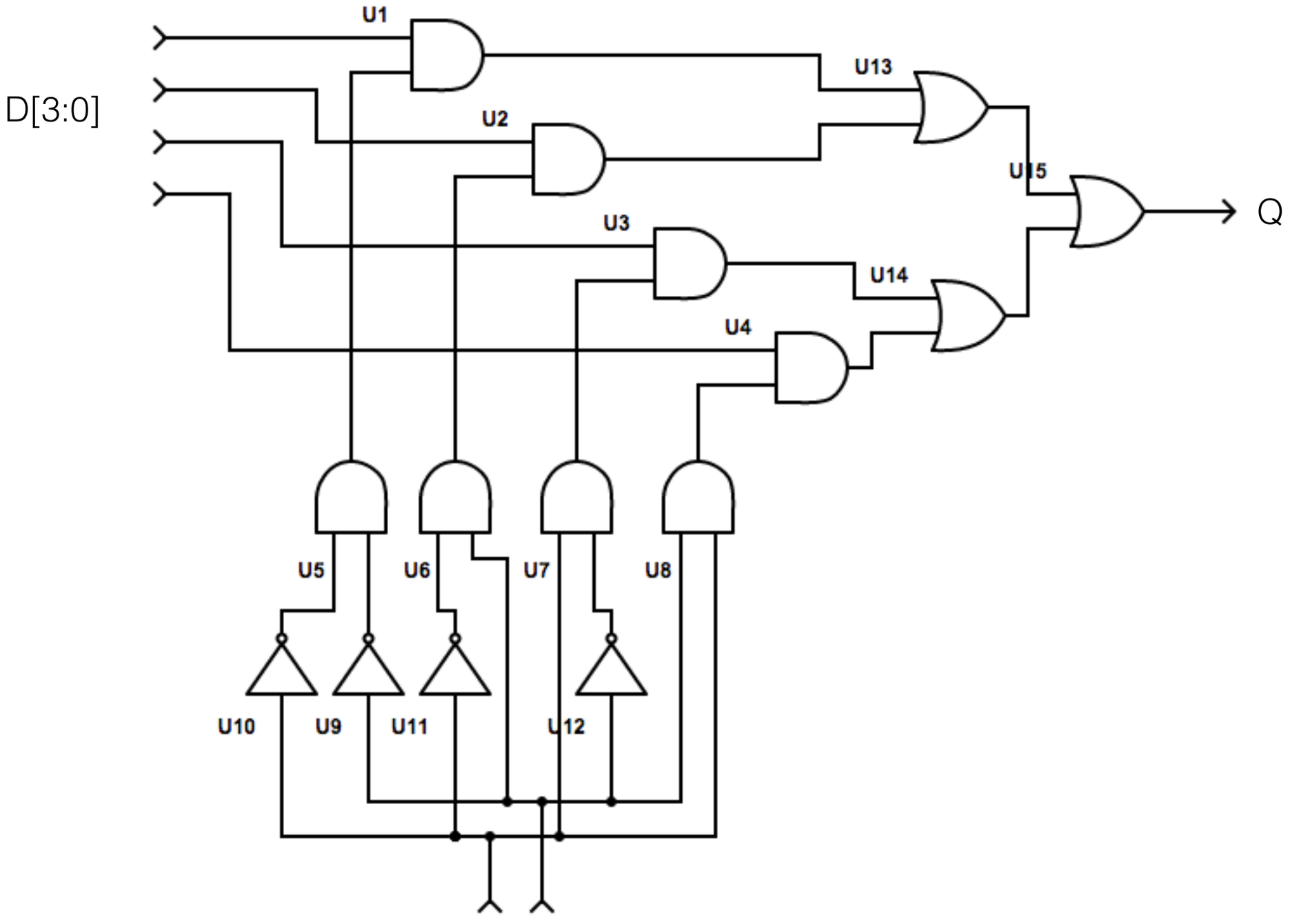
2. Using D flip-flops, write the schematic of a 3-bits shift register (the output is shifted by 3 bits).

3. Using D flip-flops, write the schematic of a 4-bits deserialiser.

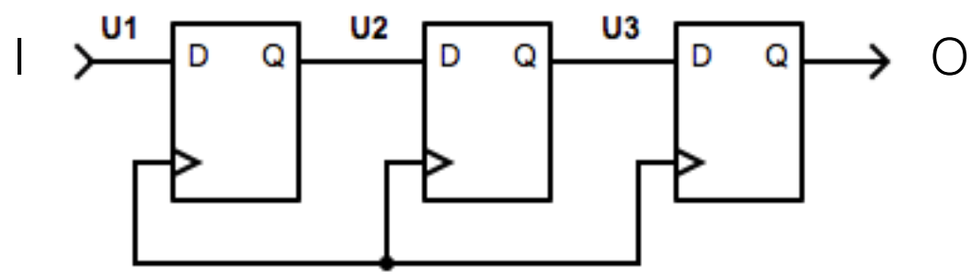
4. Write a counter by 3 (“000”, “011”, …) using only logic gates and D flip-flops.

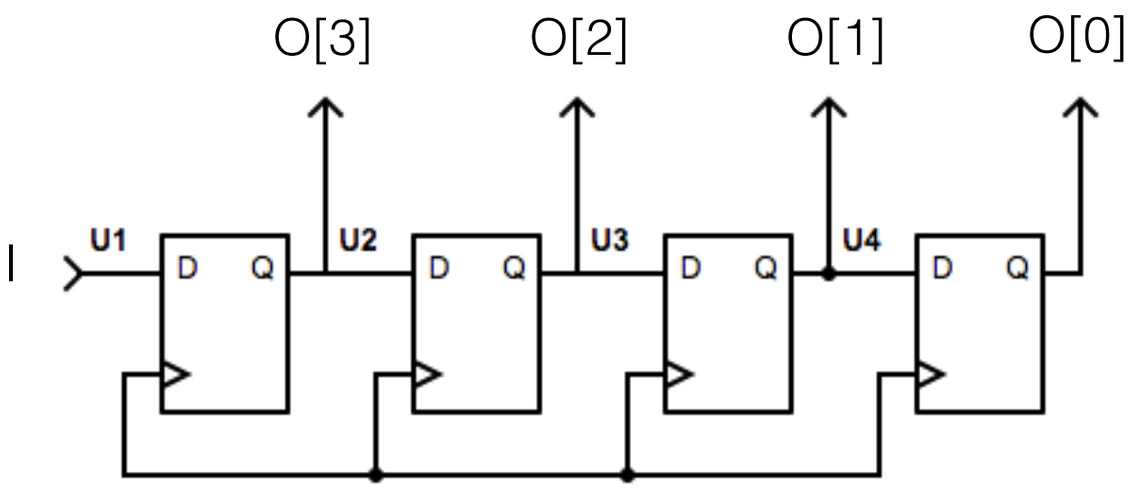
5. Write the logic for an adder of two number of two bits which yields a result on three bits.

Solution 1



Solutions 2 & 3





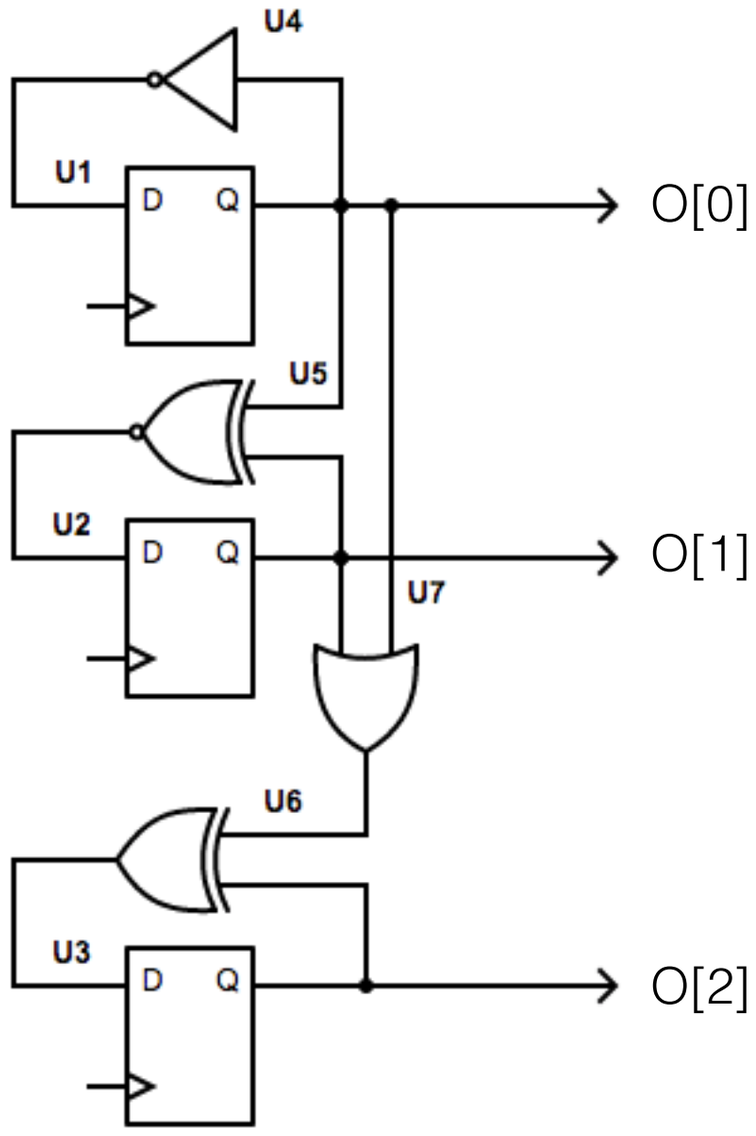
Solution 4

O[2]	O[1]	O[0]
0	0	0
0	1	1
1	1	0
0	0	1
1	0	0
1	1	1
0	1	0
1	0	1

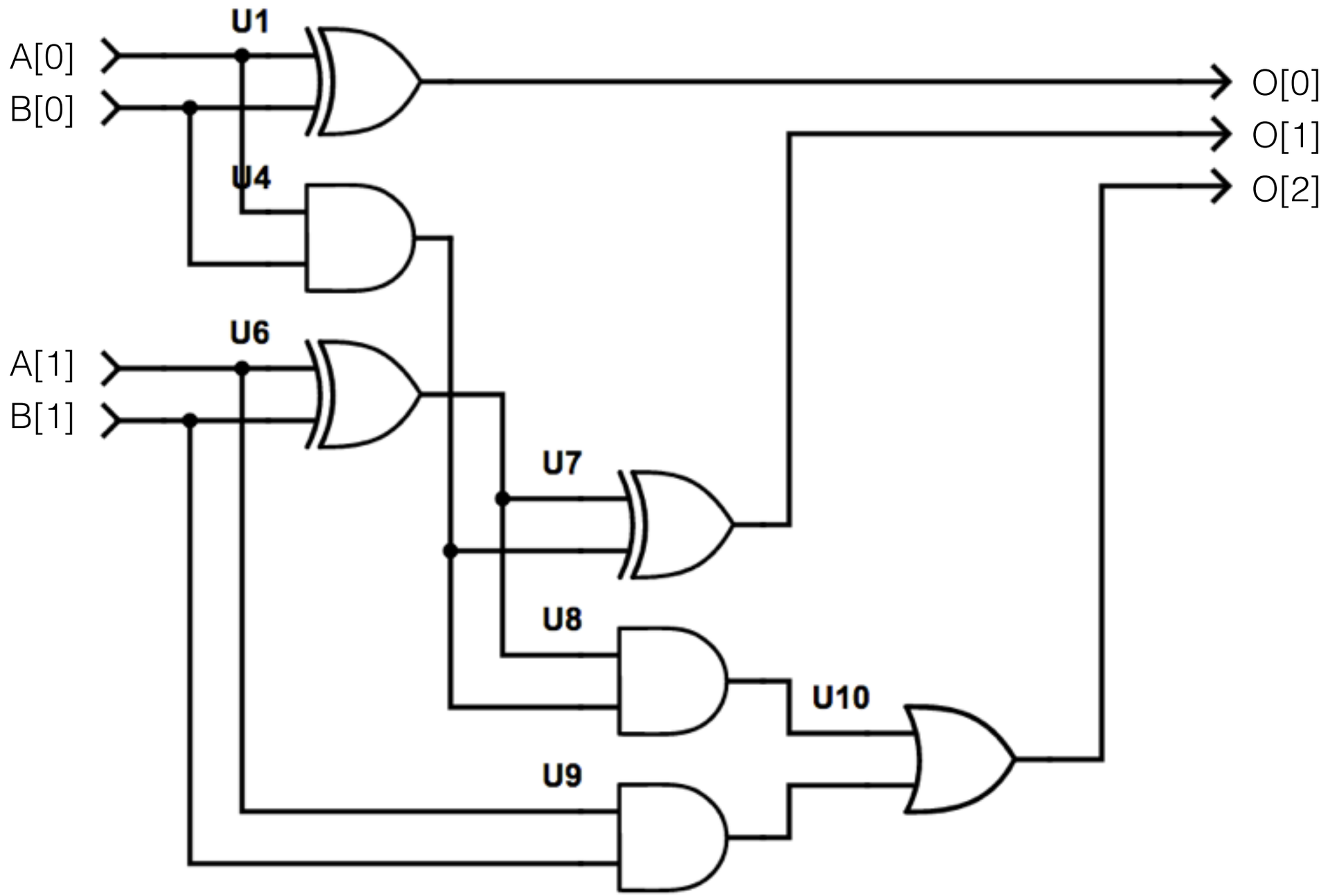
O[0]’ = NOT O[0]

O[1]’ = 0[0] NXOR 0[1]

O[2]’ = O[2] XOR (O[1] OR O[2])



Solution 5



Digital Signal Processing

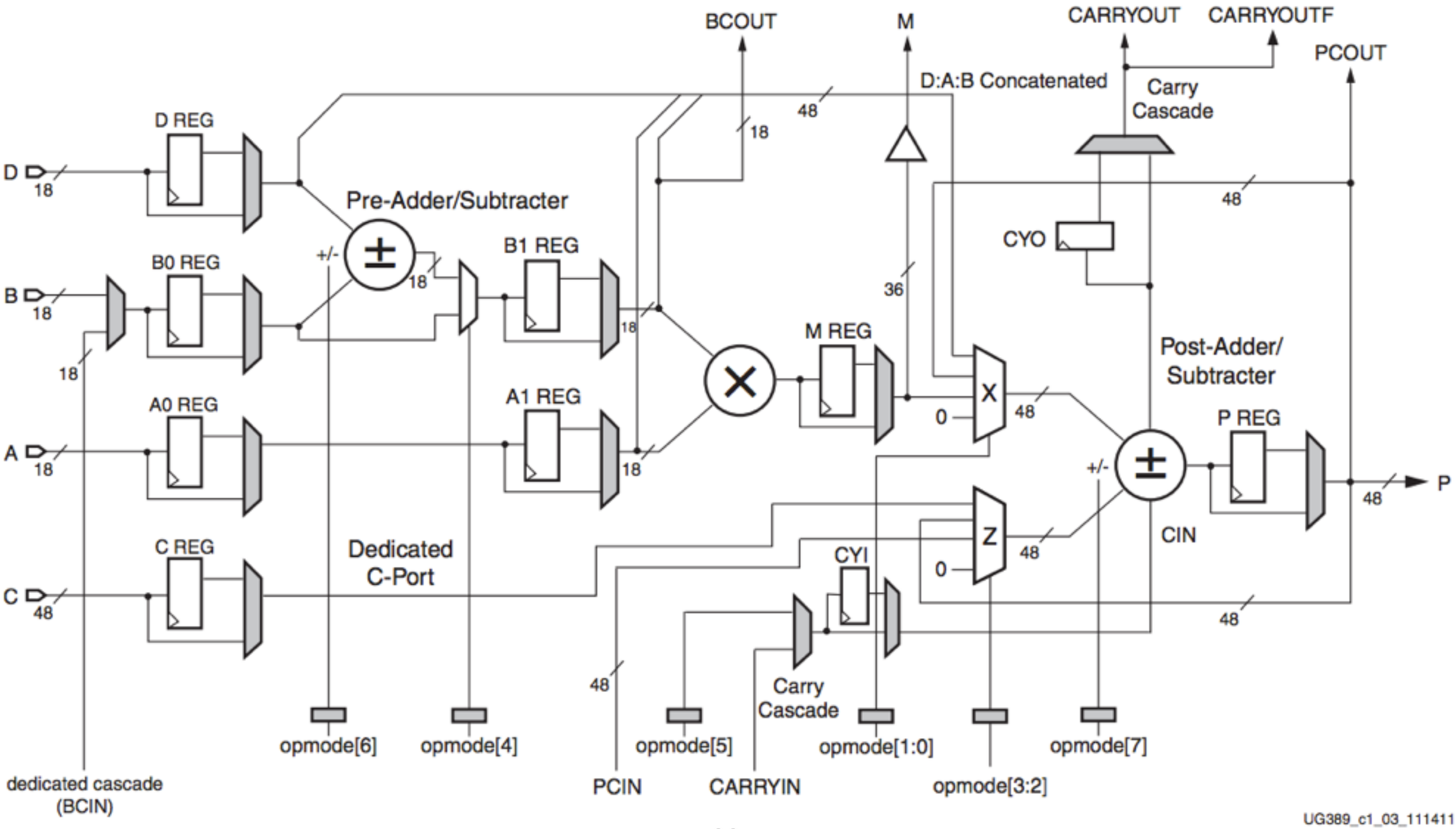
Digital Signal Processors

- DSPs perform fast mathematical operations on signals and can be used to implement a wide range of time-critical algorithms.

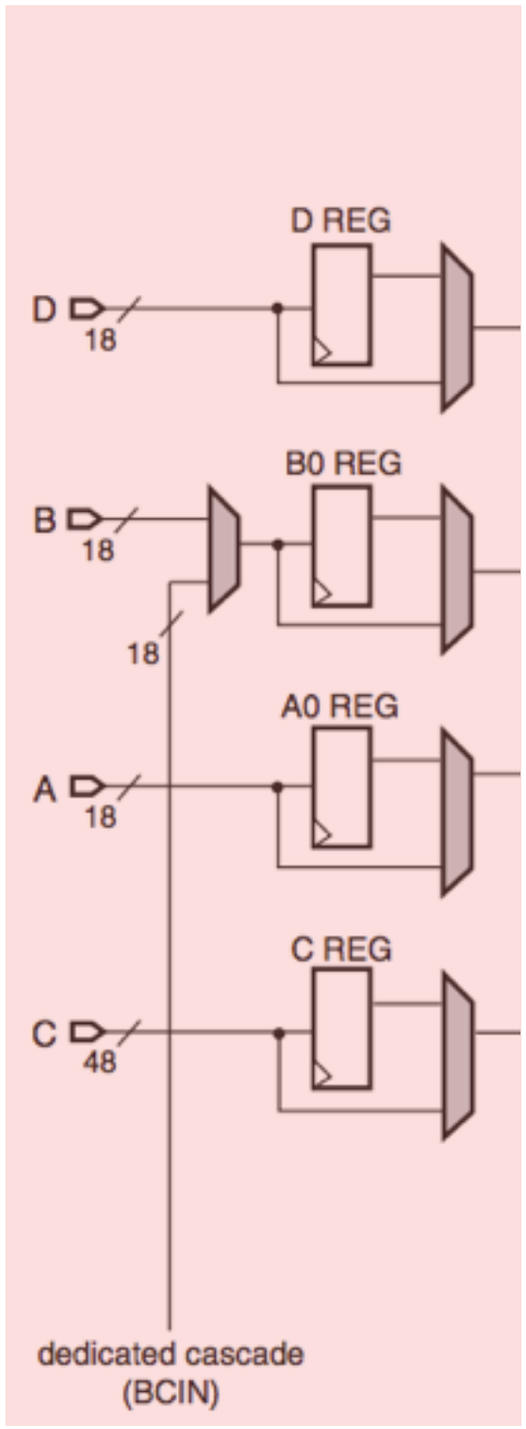
- We will analyse the schematic of the **DSP48A1** block which is used in the Xilinx Spartan6 FPGA.

- A full documentation can be found at the following address: http://www.xilinx.com/support/documentation/user_guides/ug389.pdf

DSP48A1 I


BCOUT
M
CARRYOUT
CARRYOUTF
PCOUT
D:A:B Concatenated
Carry Cascade
D REG
D
18
Pre-Adder/Subtracter
+/-
B1 REG
B0 REG
B
18
18
18
18
M REG
Post-Adder/ Subtracter
A1 REG
A0 REG
A
18
18
0
X
48
P REG
+/-
P
48
48
CIN
C REG
Dedicated C-Port
CYI
Z
0
C
48
48
48
Carry Cascade
opmode[6]
opmode[4]
opmode[5]
opmode[1:0]
opmode[7]
dedicated cascade (BCIN)
PCIN
CARRYIN
opmode[3:2]
UG389_c1_03_111411

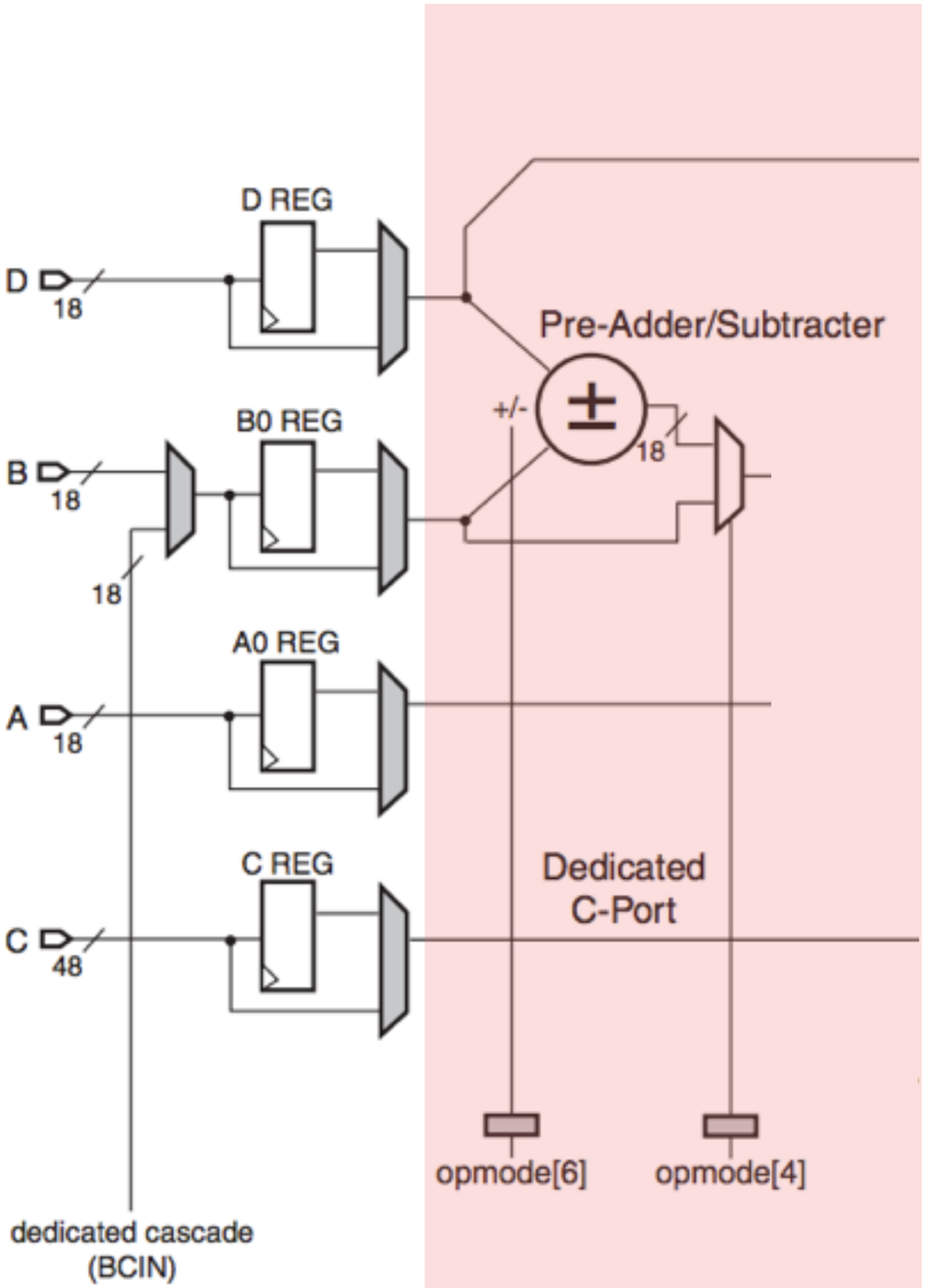
DSP48A1 II



Each input is equipped with a multiplexer that either selects a buffered/synchronous or an unbuffered/asynchronous version of the signal.

UG389_c1_03_111411

DSP48A1 III

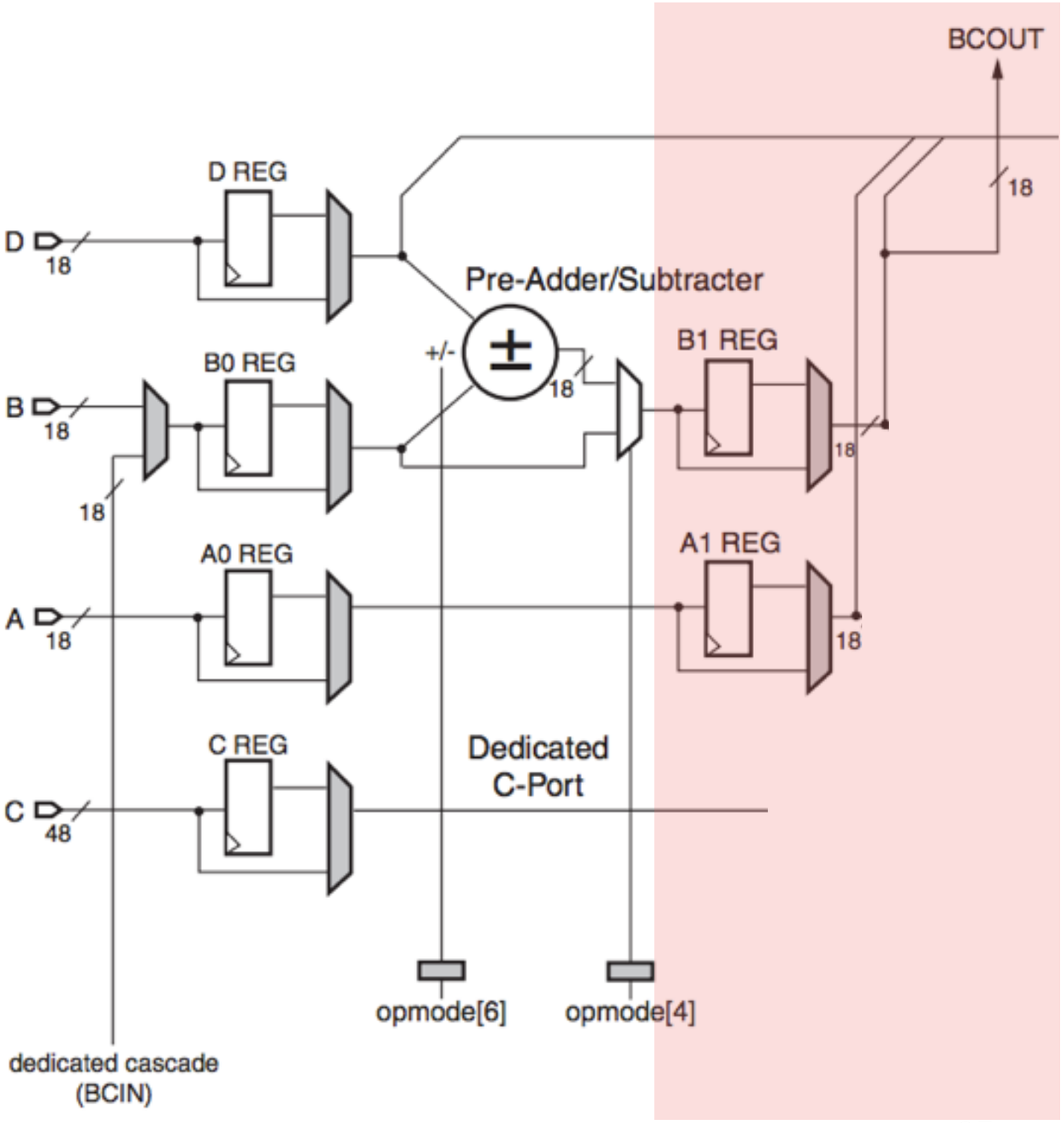


The *D* and *B* data buses then go through an Adder/Subtracter which behaviour is determined by the *opmode[6]* bit.

A multiplexer then performs a selection between the raw *B* signal or the *D±B* signal.

UG389_c1_03_111411

DSP48A1 IV

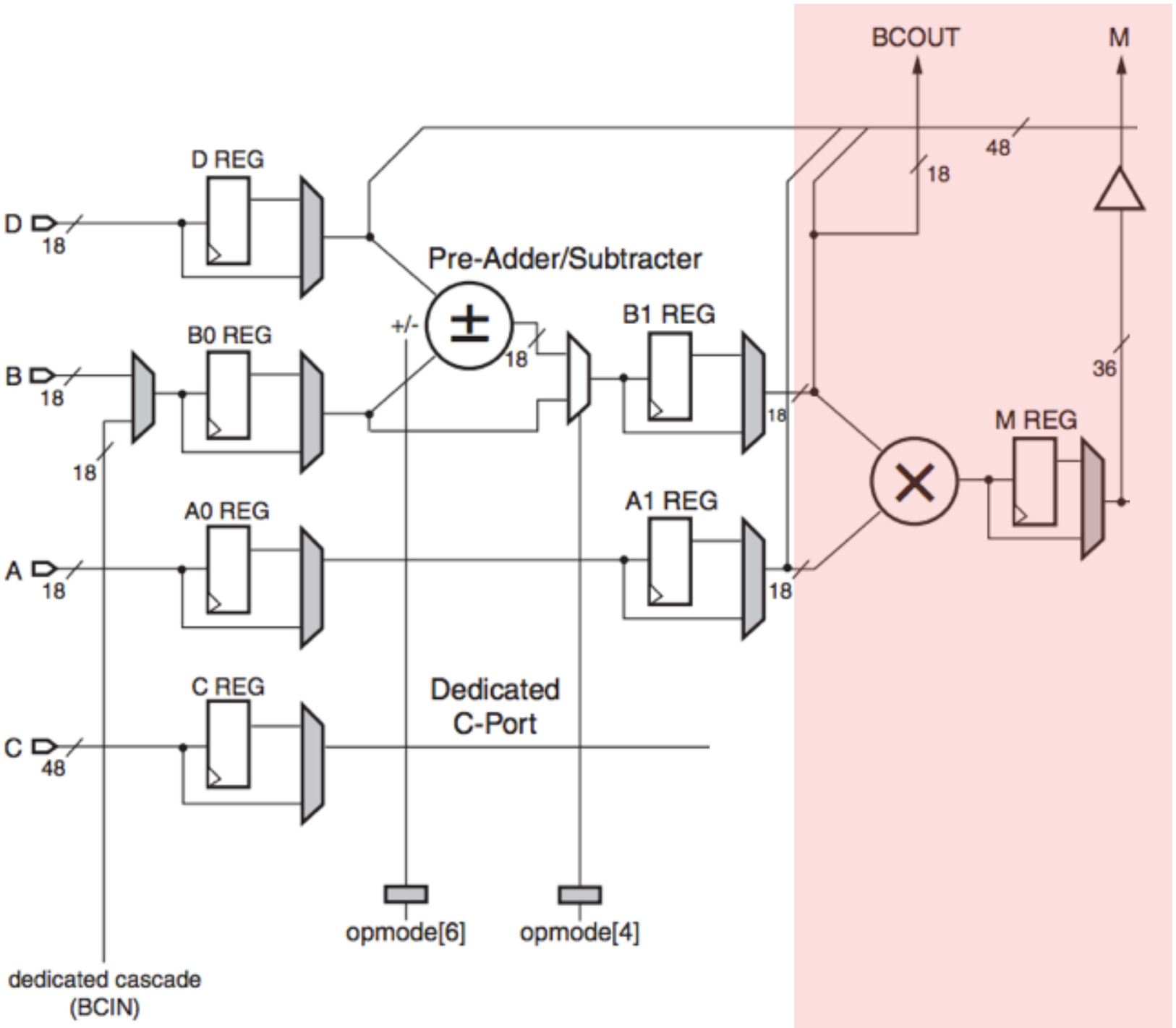


The outputs can once again be made synchronous or be kept asynchronous through two D latches and multiplexers.

Note the second register on the *A* signal which allows for synchronisation between *A* and B/*D±B*.

UG389_c1_03_111411

DSP48A1 V



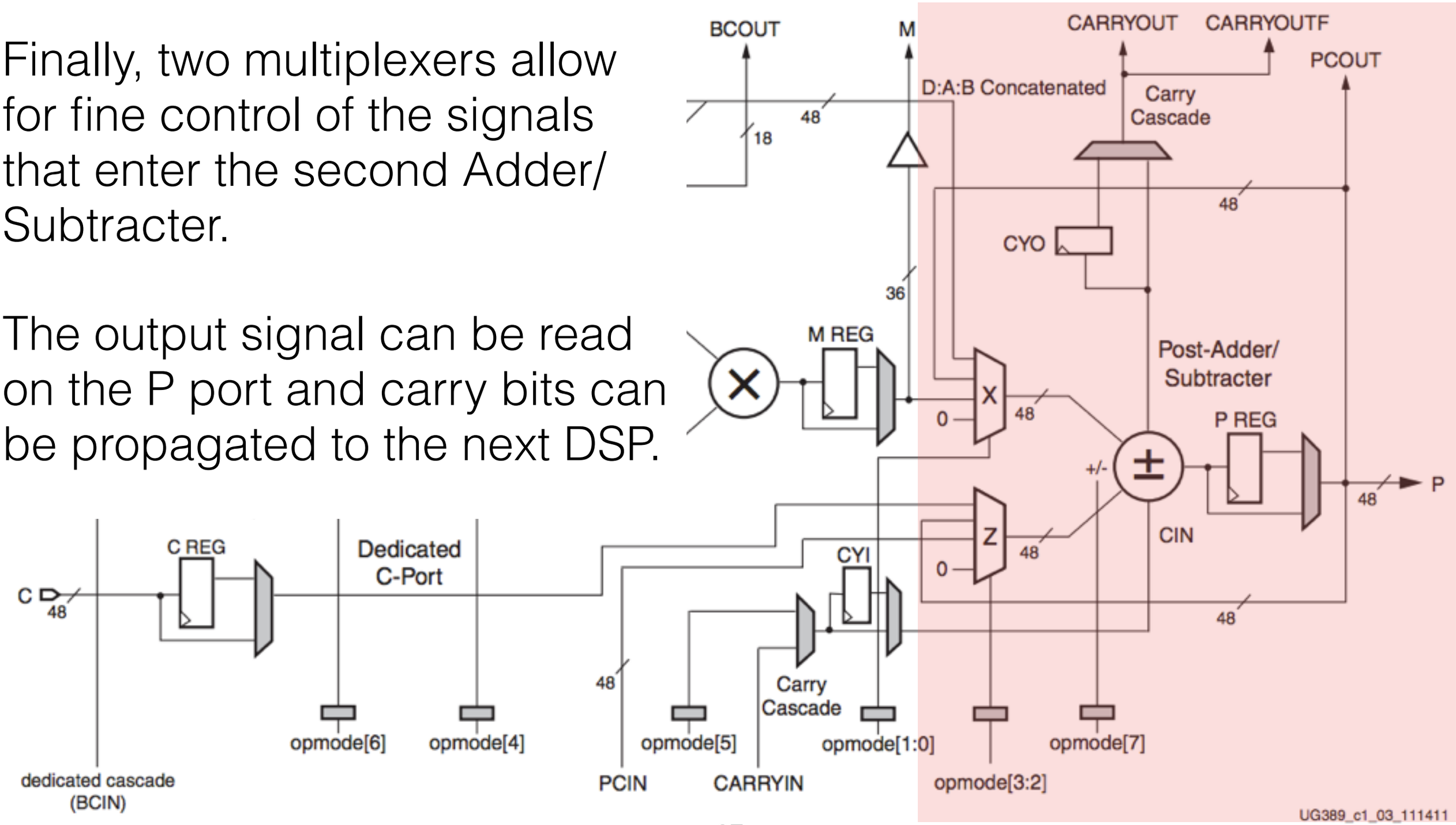
The B/*D*±*B* signal is then multiplied by the *A* signal and once again registered.

UG389_c1_03_111411

DSP48A1 VI

Finally, two multiplexers allow for fine control of the signals that enter the second Adder/ Subtracter.

The output signal can be read on the P port and carry bits can be propagated to the next DSP.



Why is this relevant?

- In any programming language (C, Python, etc) you would write something similar to

  P = (D + B) * A - C

  which the CPU would execute after compilation. This is not the case when designing for FPGAs.

- You need to think about what the hardware does in order to code efficiently!

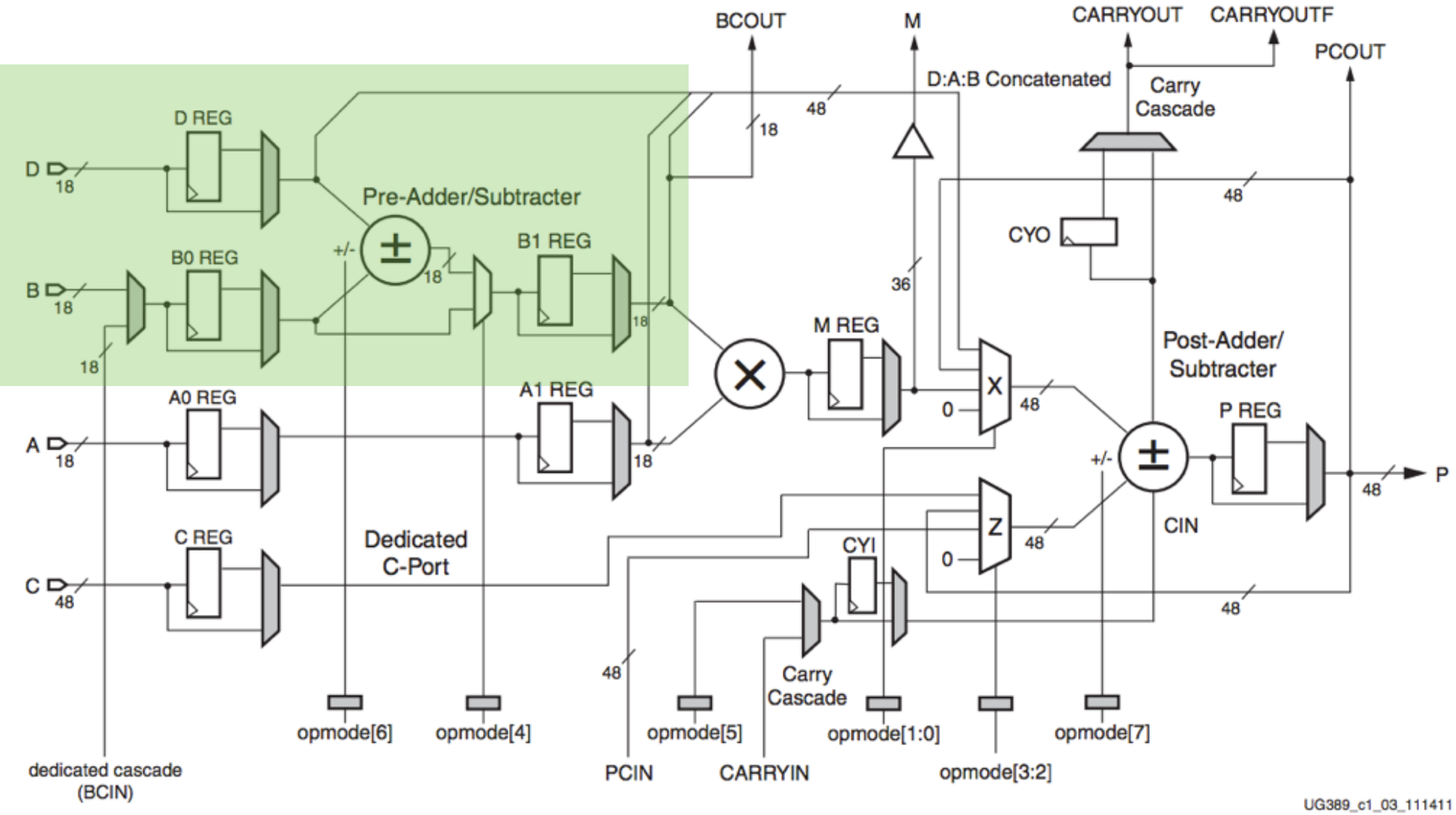
- The above code could be understood by the VHDL “compiler”, but it would be bad practice to use it. You wouldn’t be able to control when the signals are valid or not.

Exercises

By parametrising the DSP48A1, perform the following operations:

1. R = A + B
2. R = A * B
3. R = A + B + C + D
4. R = A * B - C * D

Solution 1



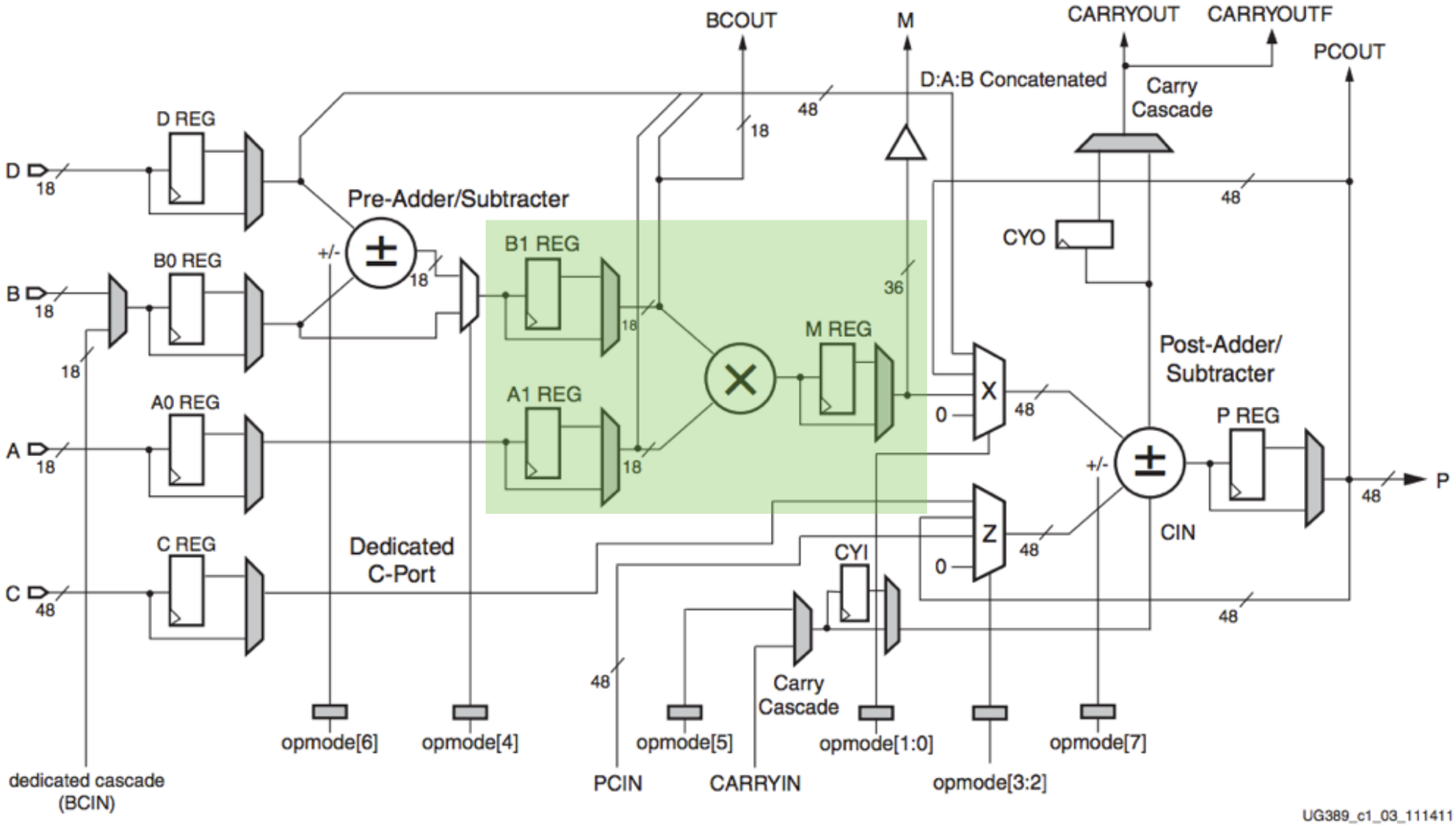
Use D and B as inputs, extract signal on BCOUT.

Use registers D, B0, and B1.

opmode[6] in Adder mode

opmode[4] selects the Adder output

Solution 2

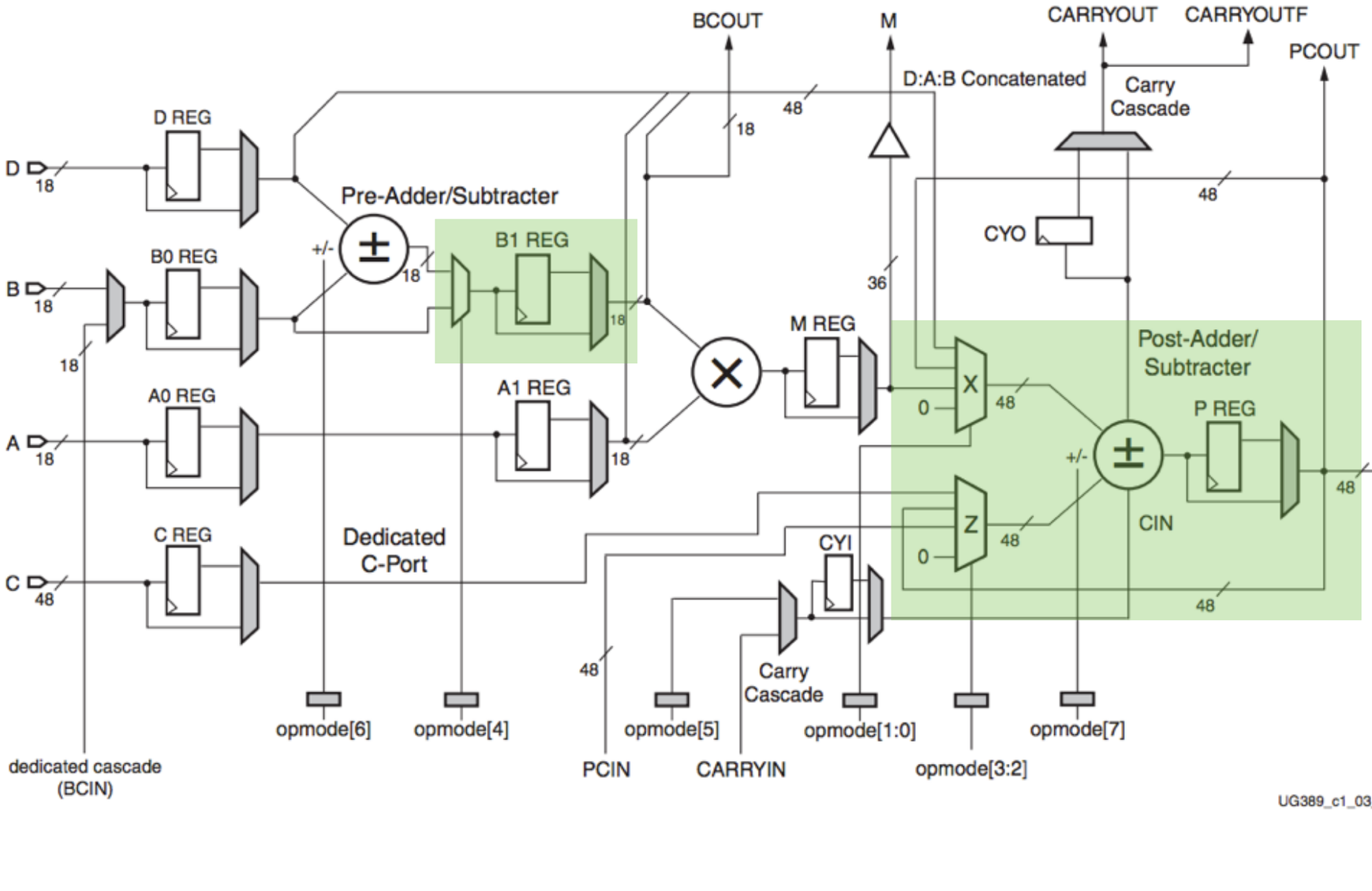


Use B and A as inputs, extract signal on M.

Use registers B1, A1, and M.

opmode[4] selects B as an output

Solution 3



Use B as input, extract signal on P.

Use registers B1 and P.

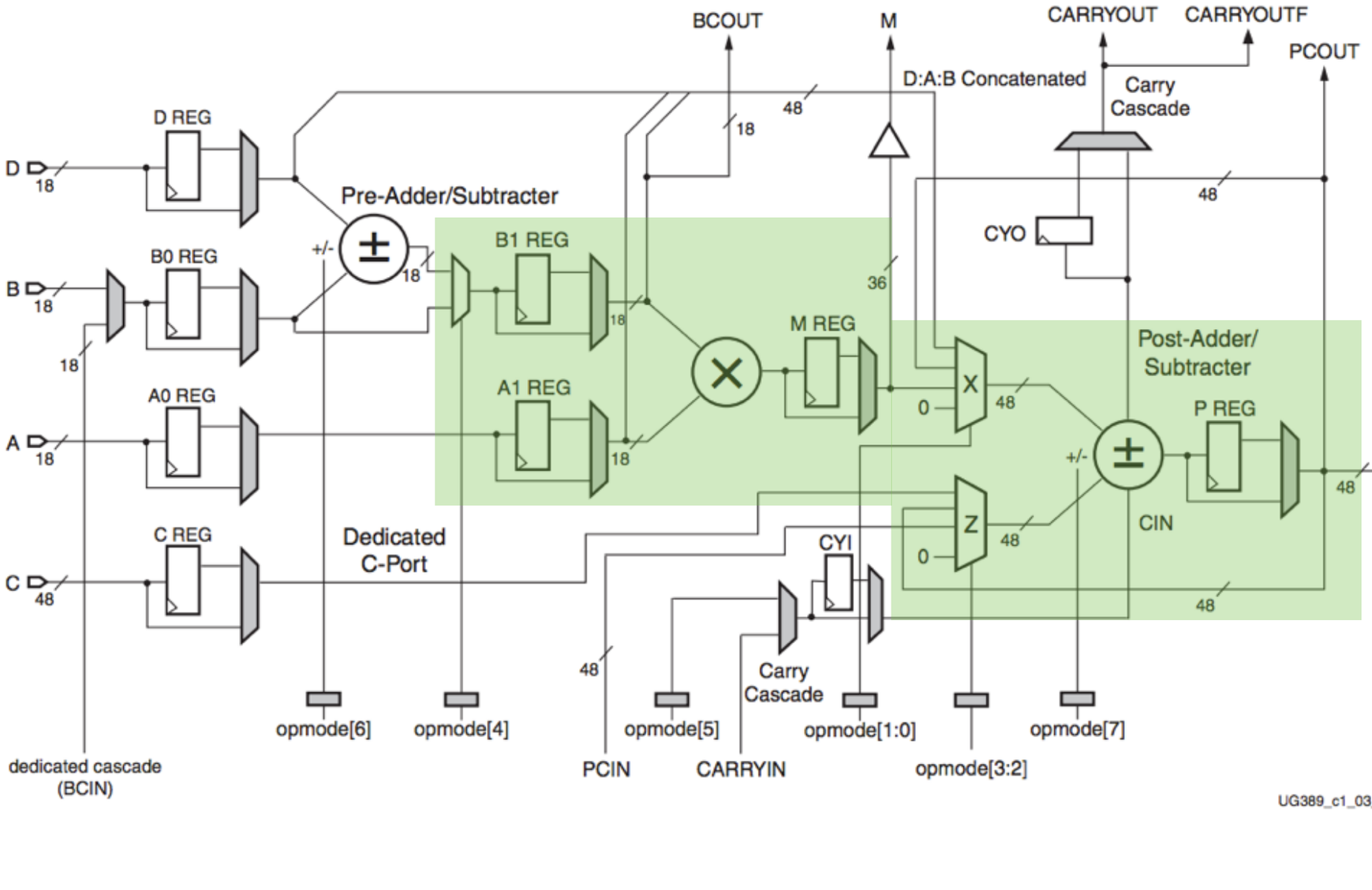
opmode[4] selects B as an output

opmode[1:0] selects B as output of multiplexer X

opmode[3:2] selects P as output of multiplexer Z

opmode[7] in Adder mode

Solution 4



Use B and A as inputs, extract signal on P.

Use registers B1, A1, M and P.

opmode[4] selects B as an output

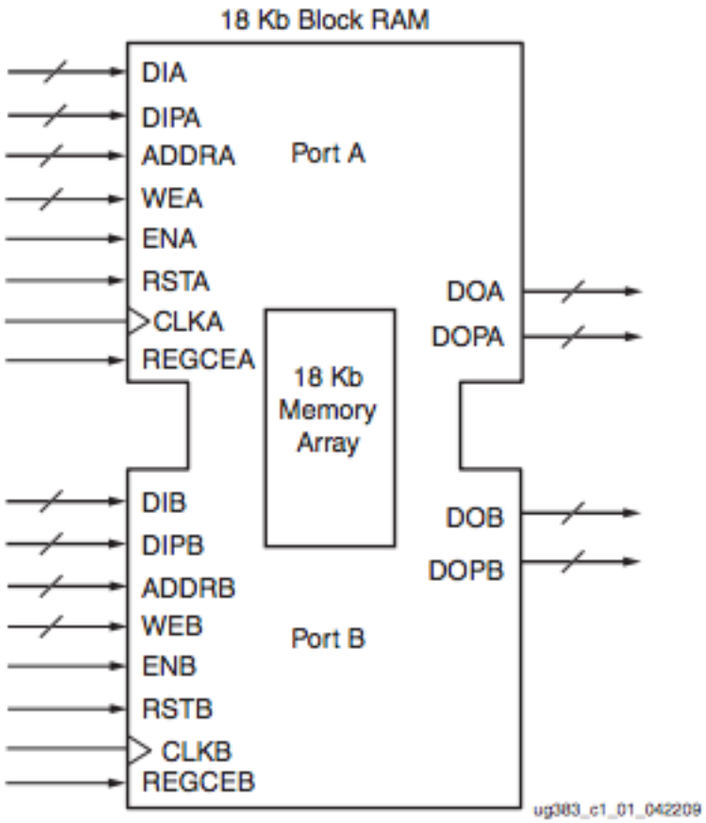
opmode[1:0] selects M as output of multiplexer X

opmode[3:2] selects P as output of multiplexer Z

opmode[7] in Subtracter mode

Block RAM

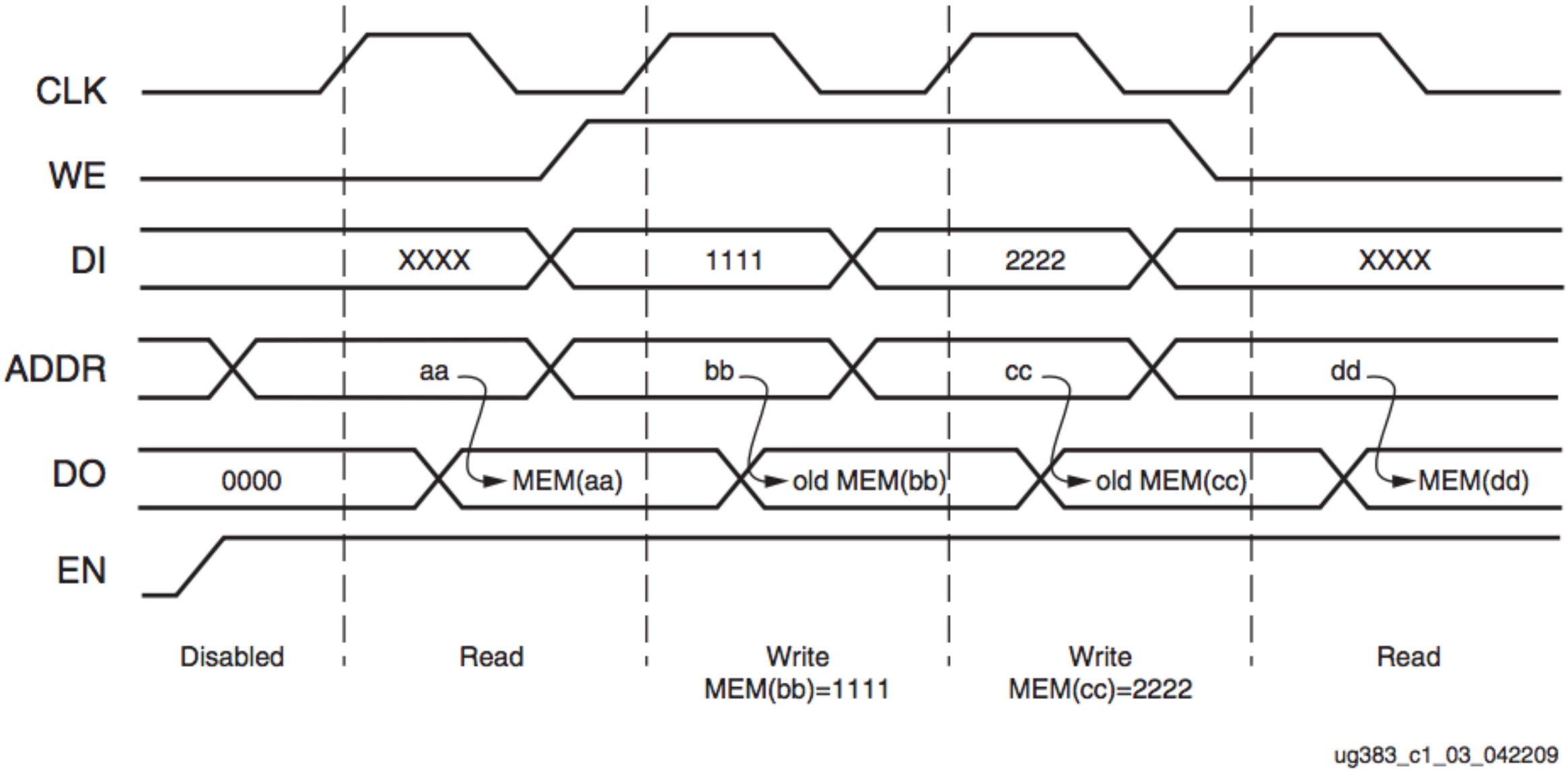
Random-Access Memory



- RAM is a read/write memory in which each entry is accessed through addressing.

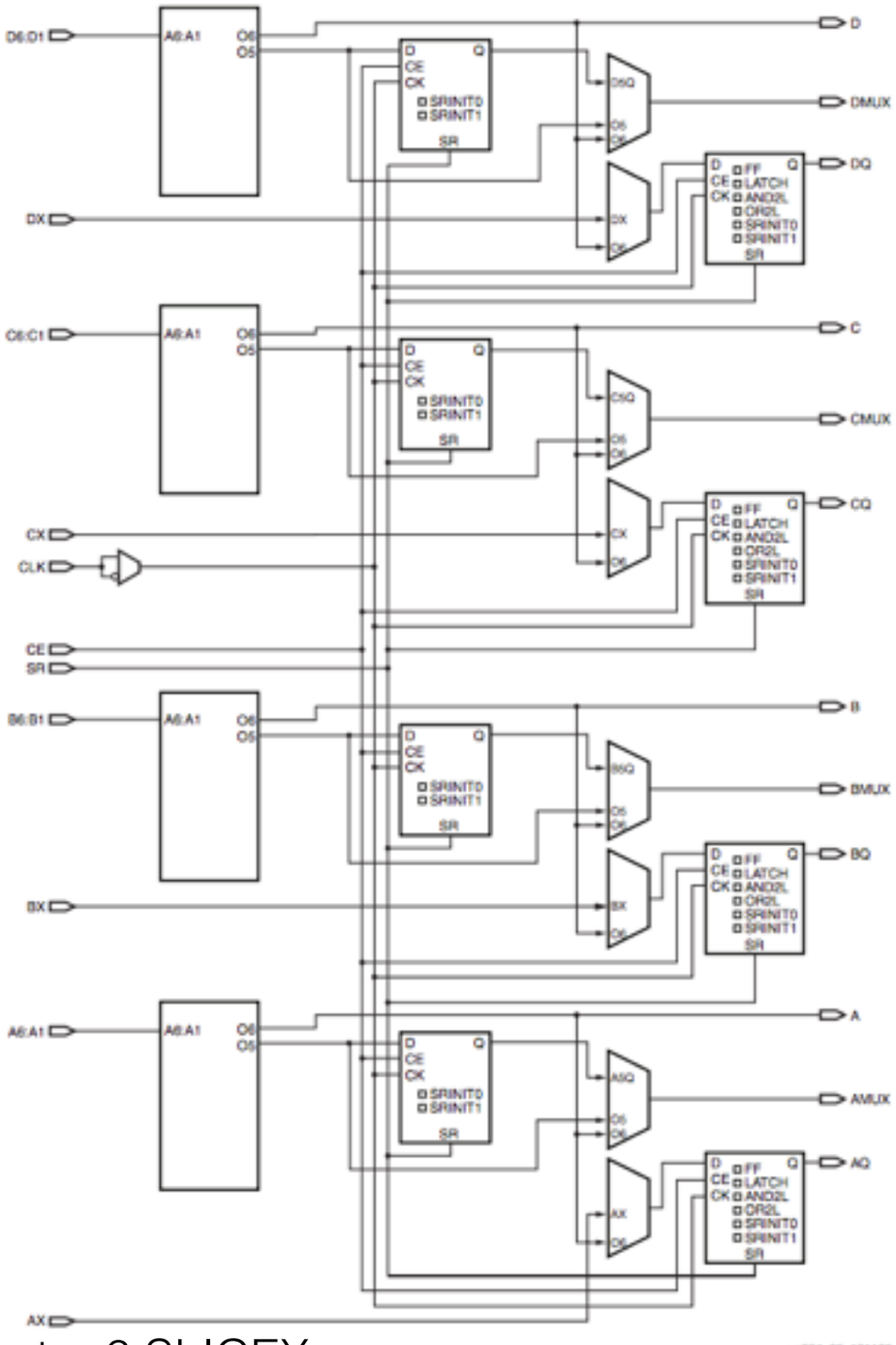
- The Spartan6 BlockRAM ressources are described in the following document: http://www.xilinx.com/support/documentation/user_guides/ug383.pdf

Data Flow


CLK
WE
DI
XXXX
1111
2222
XXXX
ADDR
aa
bb
cc
dd
DO
0000
MEM(aa)
old MEM(bb)
old MEM(cc)
MEM(dd)
EN
Disabled
Read
Write
MEM(bb)=1111
Write
MEM(cc)=2222
Read
ug383_c1_03_042209

Configurable Logic Blocks

Slices

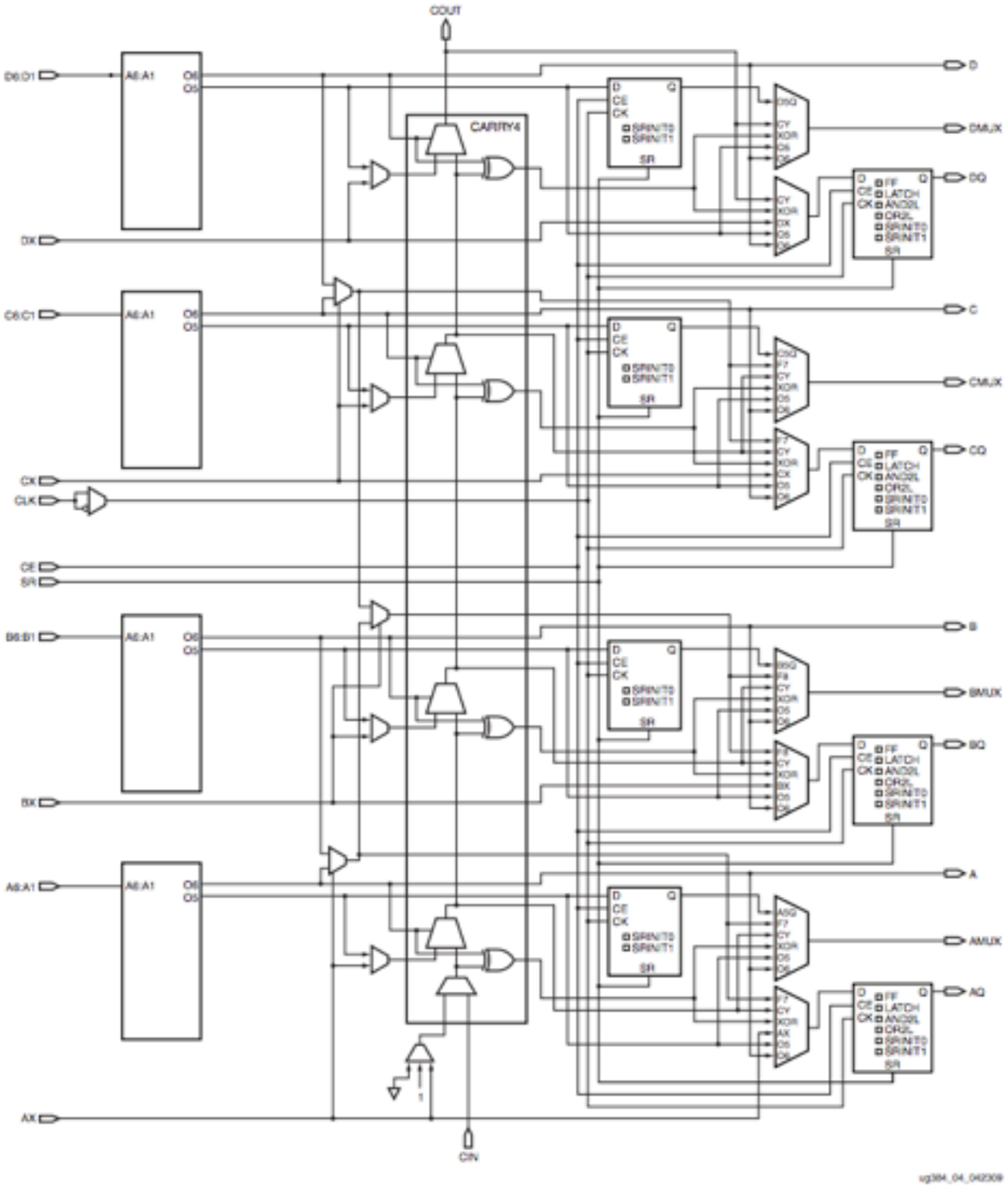


- Slices are collections of LUTs, latches, multiplexers, logic gates, … that are tightly interconnected.

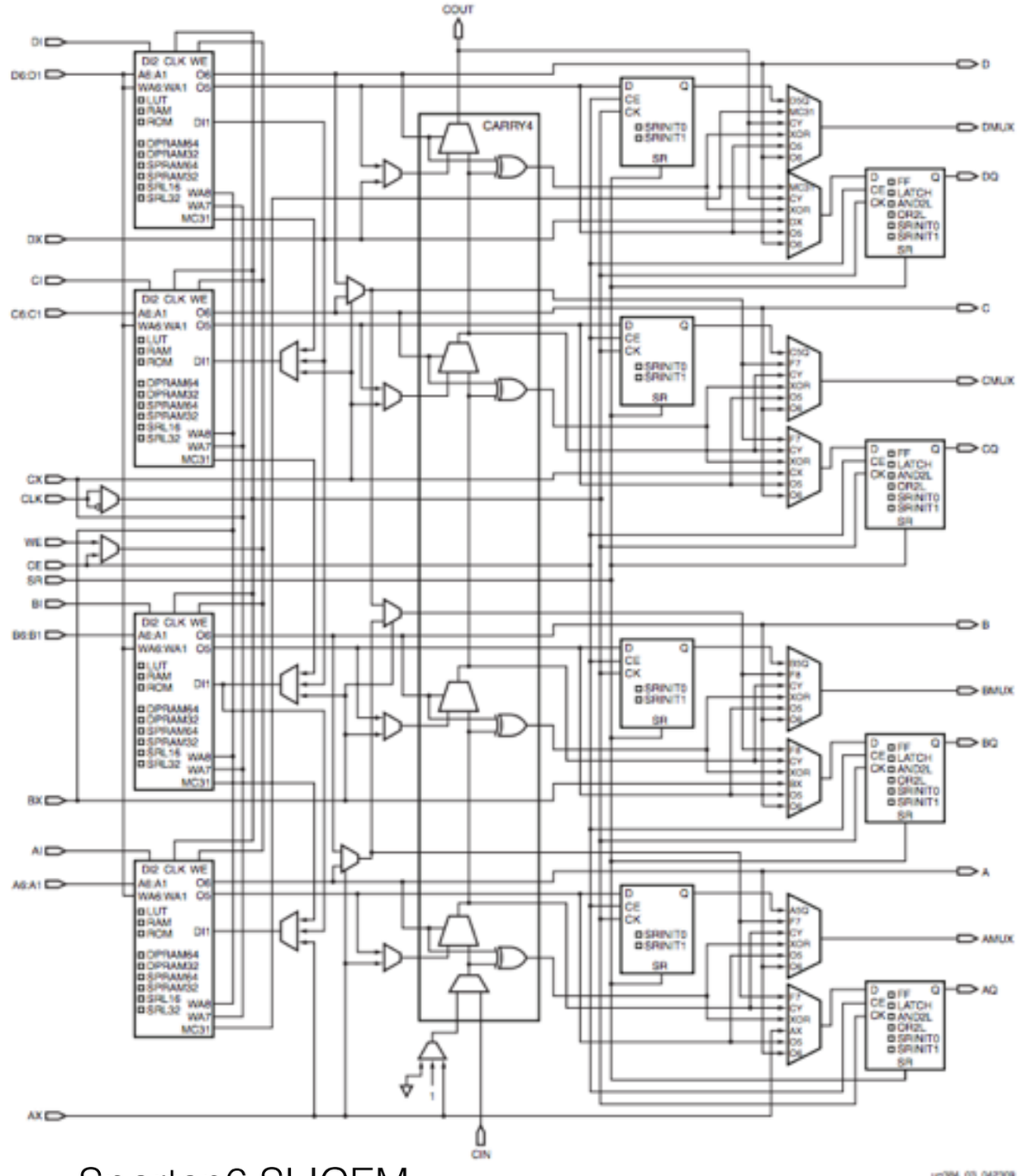
- The example on the left holds 4 LUTS, 8 DFF, and 8 multiplexers.

- Documentation on the slices can be found here: http://www.xilinx.com/support/documentation/user_guides/ug384.pdf

Spartan6 Slices



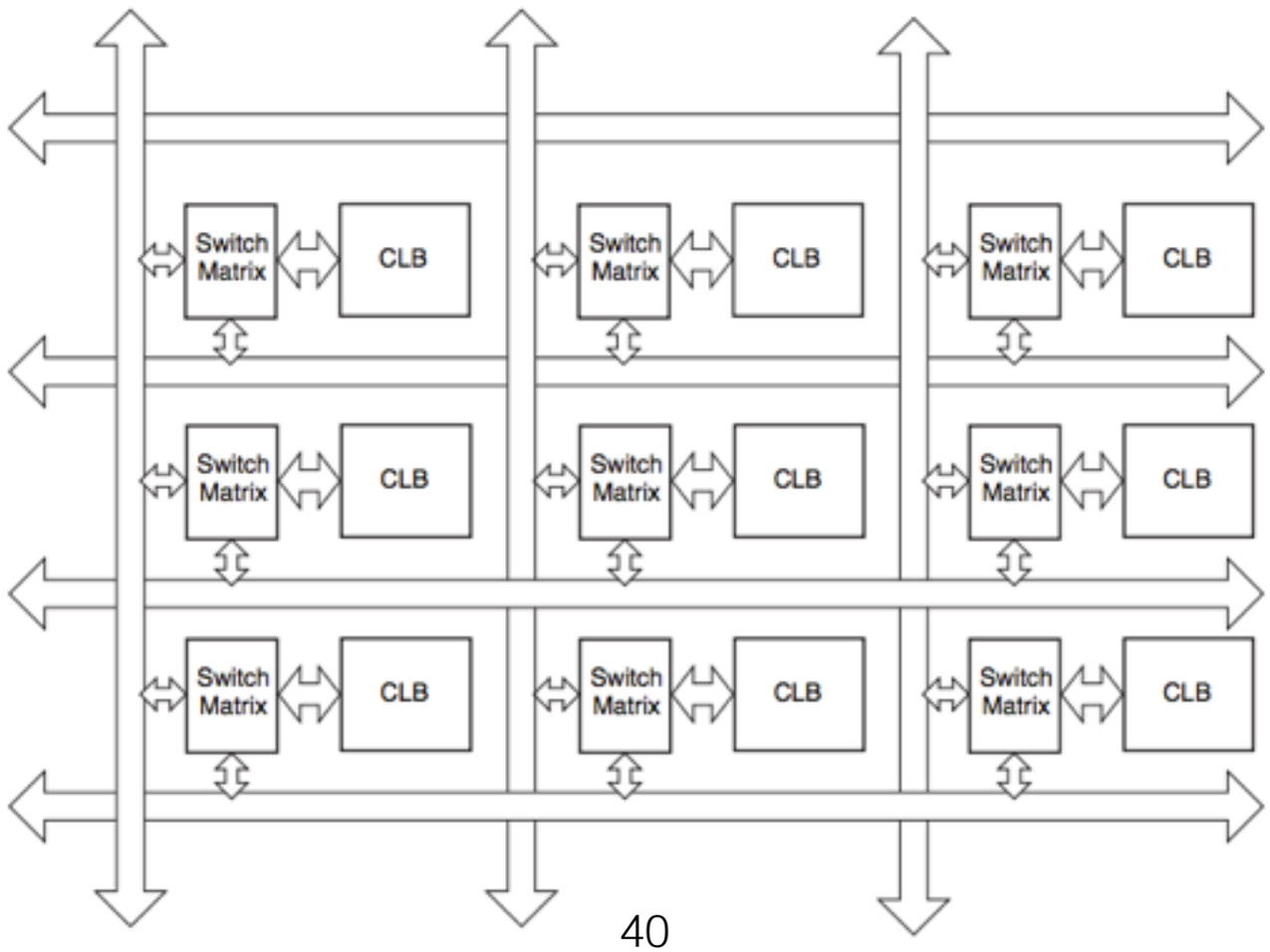
Spartan6 SLICEL



Spartan6 SLICEM

Configurable Logic Blocks

CLBs contain one or more slices and are the building blocks of the FPGA. They are the components that are replicated in the FPGA in order to form an array. They are connected to the switch matrix which defines the interconnections between the blocks.



Input / Output Pins

Pins

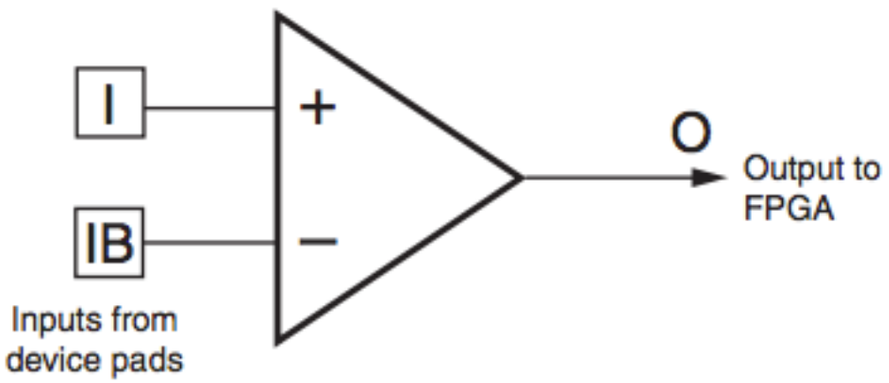
- In every design, you will have signals entering/leaving your FPGA. To do so, you need to connect internal signals to Input/Output (IO) pins, which are routed to other components on the PCB.

- IO pins are not simply wires which enter/leave the FPGA, they can be buffers, serialisers/deserialisers, differential drivers, … All those possibilities are implemented at the hardware level. They are components that are connected directly to the pins.

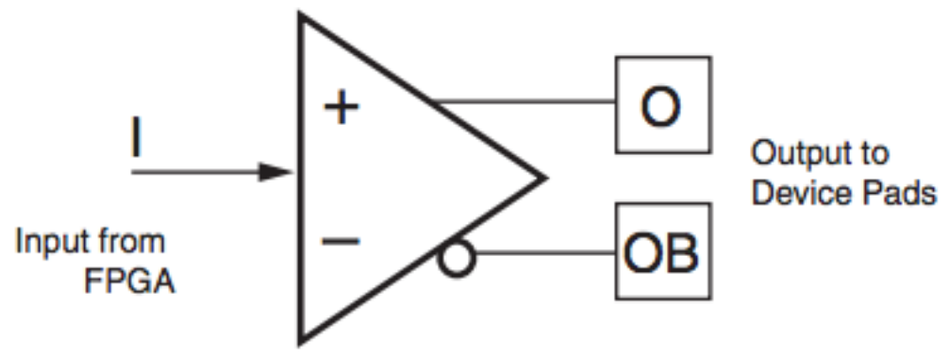
- Documentation about the Spartan6 IO pins is available here: http://www.xilinx.com/support/documentation/user_guides/ug381.pdf

Differential Signalling

IO pins support differential signalling and can convert differential signals to single-ended signals at the IO level. Two pins are used to form one signal inside the FPGA (or vice-versa).



ug381_c1_13_111909

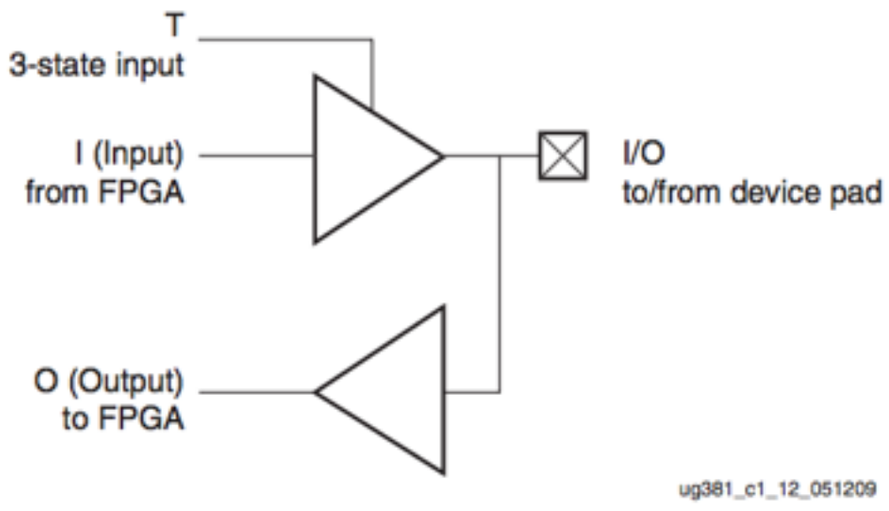


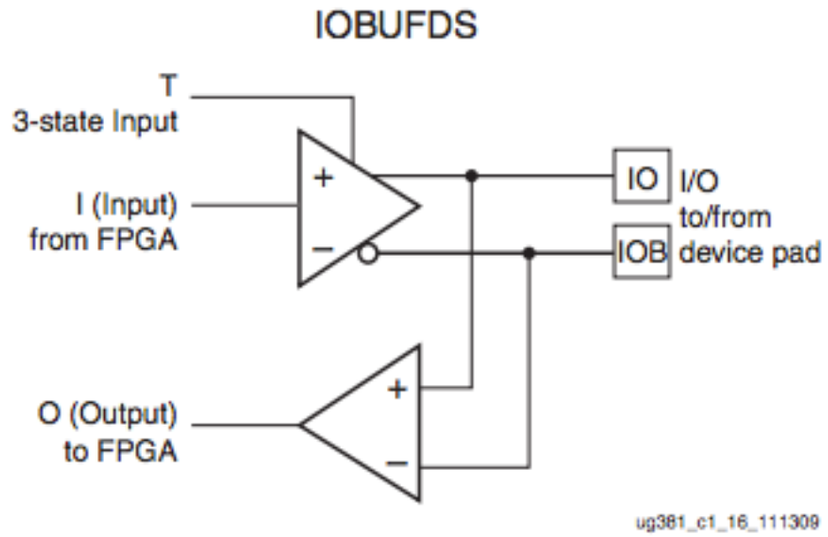
ug381_c1_14_051209

Tri-State IOs

An other possibility is to use a pin as both an input and an output signal (I2C for example). In this case, the FPGA must know when to drive the signal (put voltage on the line) and when to listen (get voltage).

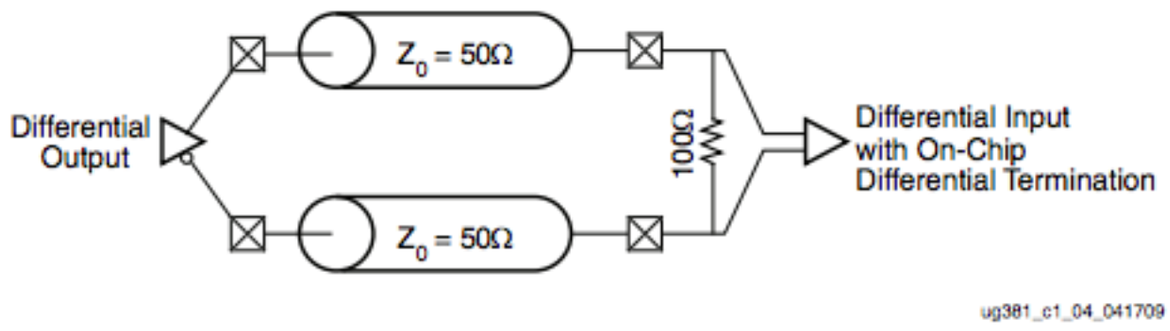
To do so, a tri-state buffer is used to switch between input and output mode.





Differential Termination Pull-Ups/Down

The FPGA also offers the possibility to add differential terminations to differentials pairs or pull-up/down resistors to single-ended signals.



Clocking

Clocking Ressources

- The full documentation on clocking ressources in the Spartan6 devices can be found here: http://www.xilinx.com/support/documentation/user_guides/ug382.pdf

- We will focus on the most common operations that can be performed on clocks.

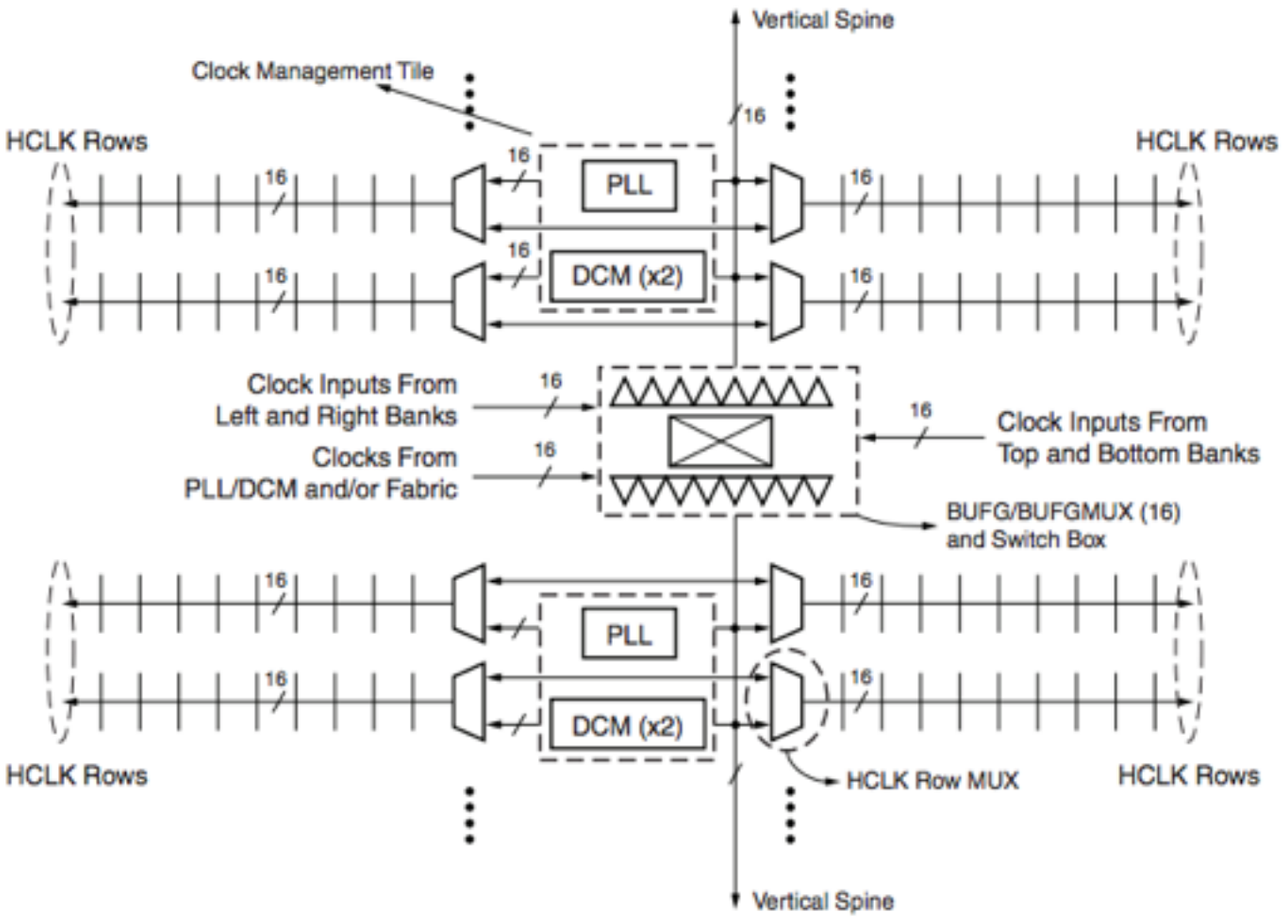
Clock Signals

- Clock signals offer the possibility to drive a design at a given frequency. This is needed when using communication protocols or any other task that need some sort of synchronicity.

- As previously shown, components and paths will add delay to the signals. This is a major problem for clocks.

- Therefore, the FPGA is equipped with a dedicated high-speed, low-squew, clock network.

Clock Network



- Clocks live in two dedicated networks: global & local network.
- The global network spans all over the FPGA and allows the clocks to be transferred from one domain to another.
- The local networks provide clocks to specific sectors of the FPGA.
- Clock buffers and multiplexers are used to select which signal enters which part of the device.
- The clock network can be seen as an water irrigation system that covers the entire FPGA.

Buffers and Multiplexers

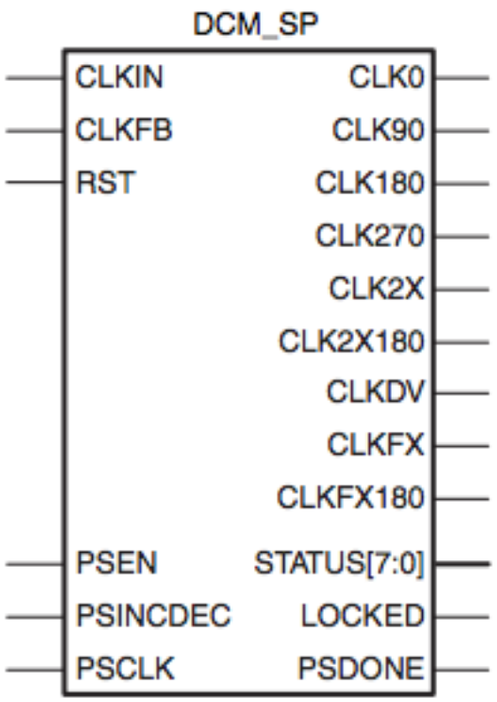
- The FPGA is equipped with global buffers (BUFG) and local buffers (BUFH).

- Global buffers are used to bring signals into the global clock network.

- Local buffers are used to bring clocks from the global to the local network.

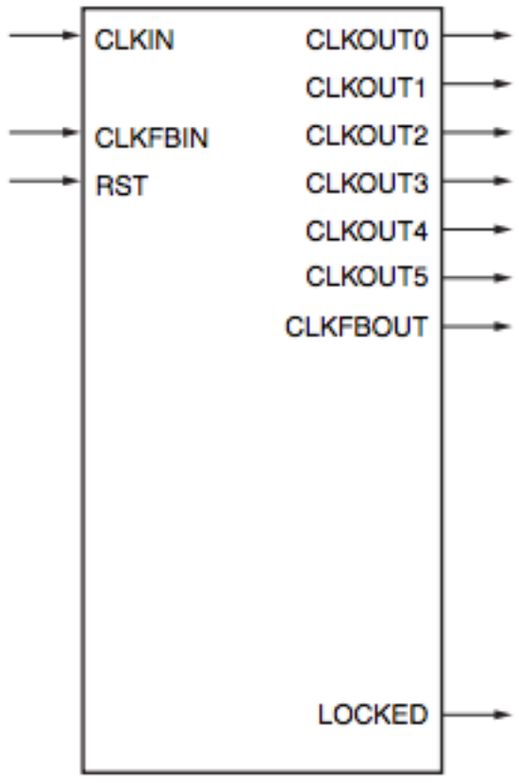
- Clock multiplexers are also present in the FPGA and allow to switch between clocks dynamically or select which clocks will enter a defined domain.

Digital Clock Management



- DCMs offer the possibility to generate, deskew, phase-shift, … a given clock signal. From a given input clock, they will:
  - shift the clock by 0°, 90°, 180°, or 270°
  - double the frequency (0° or 180° shift)
  - divide or multiply the frequency by a given factor (0° or 180° shift)
- The clocks generated by the DCM are not placed on any network. They have to be routed “manually”.

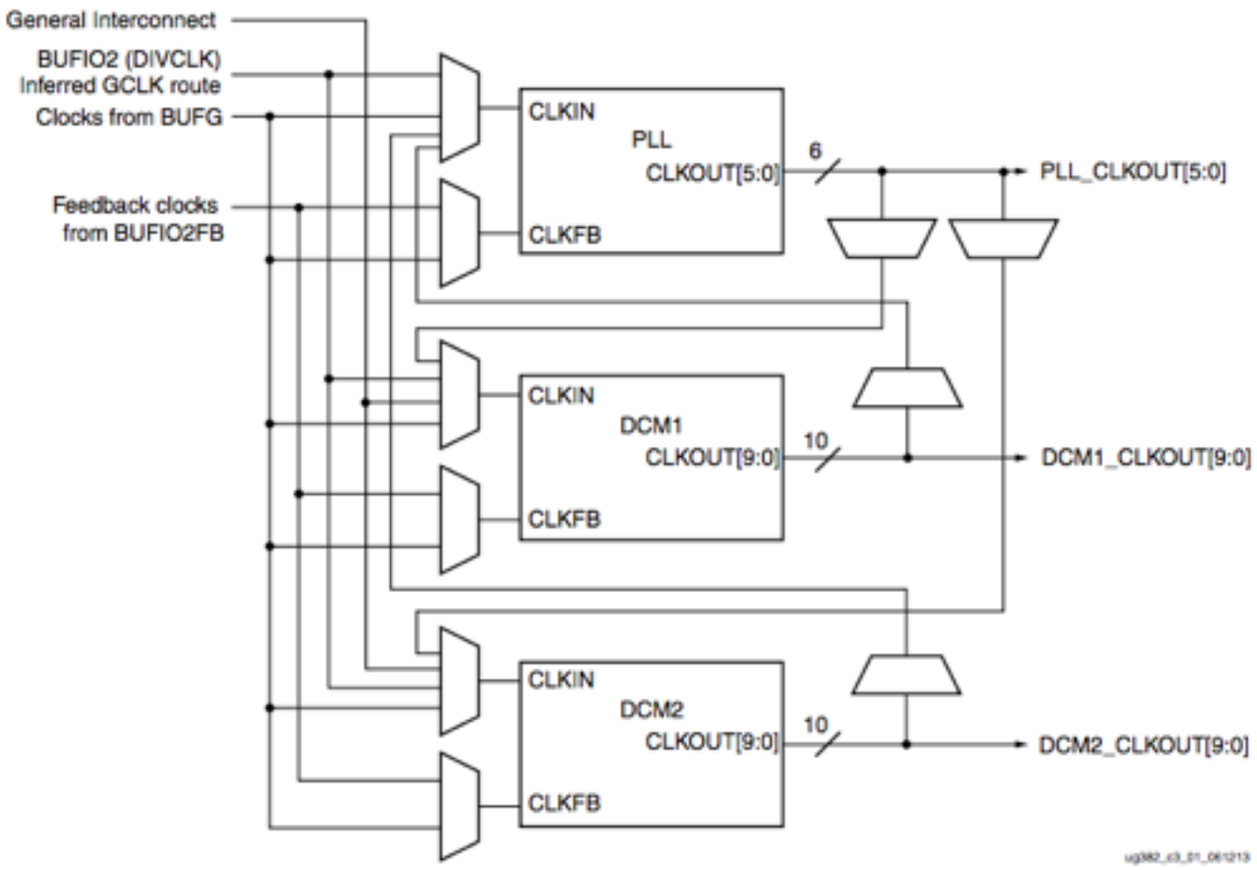
Phase-Locked Loop



- PLLs are components that generate multiple clock signals in phase with the input clock but with different frequencies.

- The clocks generated by the PLL are not placed on any network. They have to be routed “manually”.

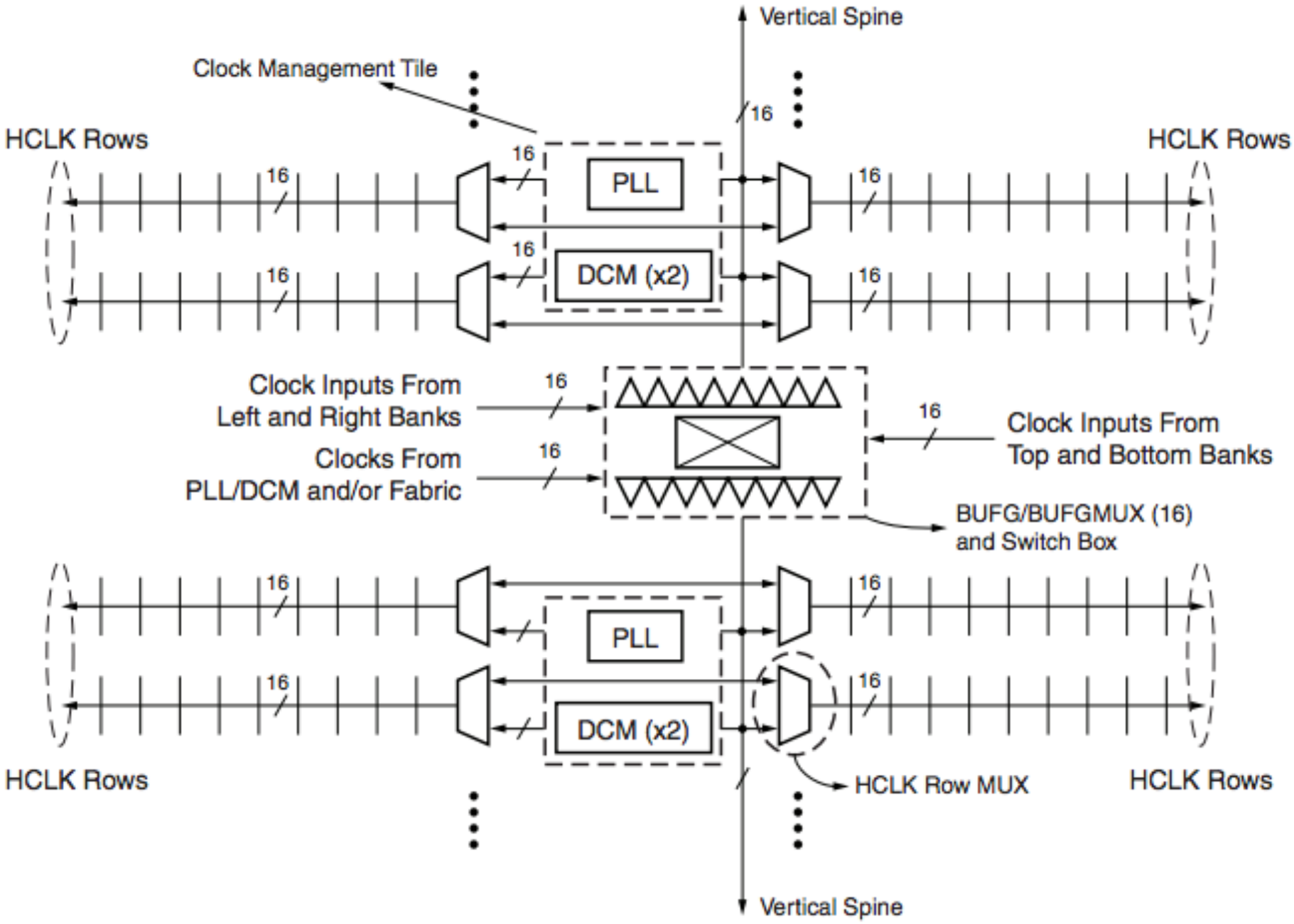
Clock Management Tile



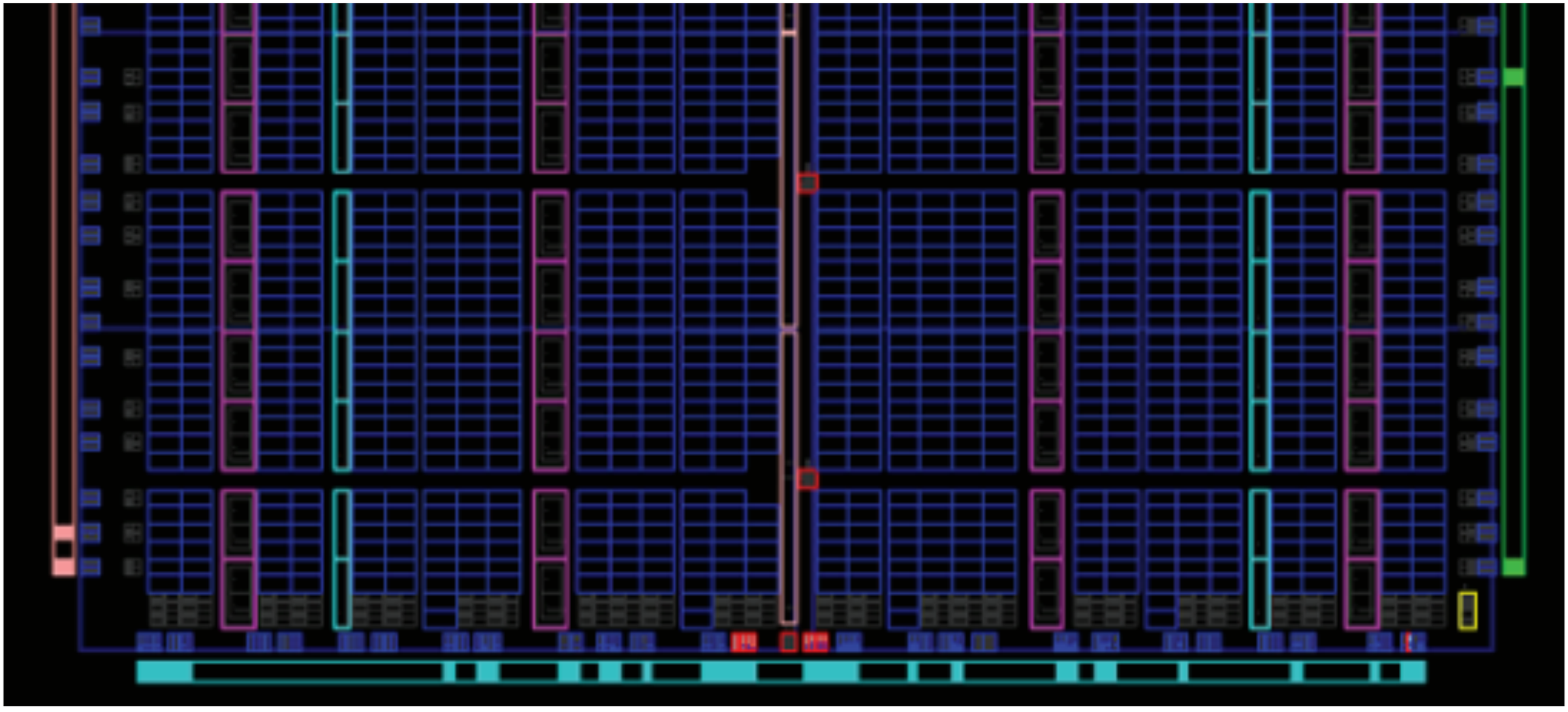
In the Spartan6, clocking ressources are regrouped in CMTs. Each CMTs contains 2 DCMs and 1 PLL.

In a CMT, clocks can be routed between PLLs and DCMs.

Clocking Scheme I



Clocking Scheme II

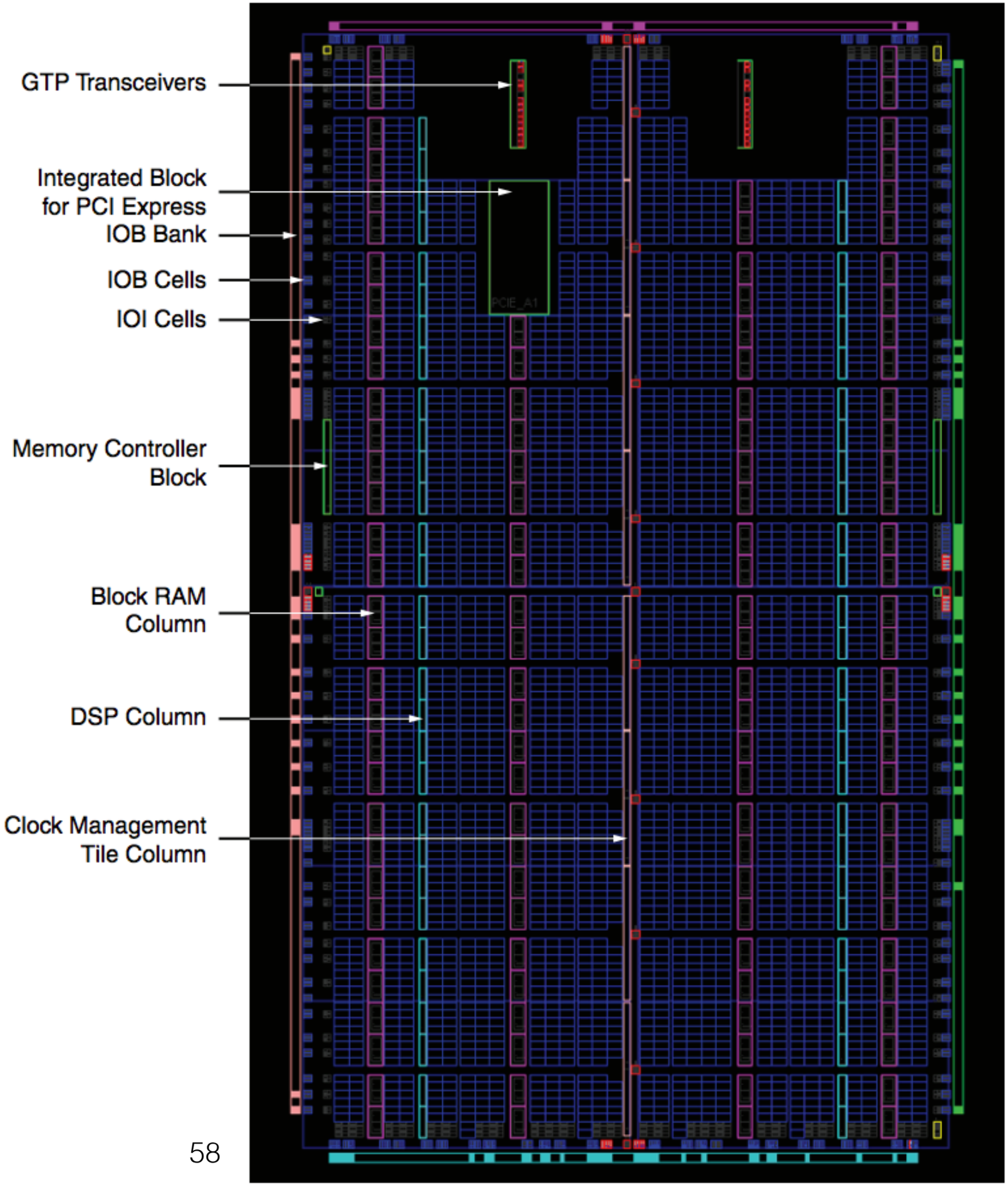


FPGA

Summary

- LUTs, DFFs, multiplexers, … are grouped to form Slices.
- Slices are grouped to form CLBs.
- An FPGA contains many CLBs which are interconnected through a network which can be programmed to form certain paths between CLBs.
- The IO pins of the FPGA are equipped with buffers, tri-state buffers, … and are also connected to this network.
- Furthermore, the FPGA also contains DSPs and BRAM.

FPGA


GTP Transceivers
Integrated Block for PCI Express
IOB Bank
IOB Cells
IOI Cells
Memory Controller Block
Block RAM Column
DSP Column
Clock Management Tile Column
PCIE_A1

Programming an FPGA

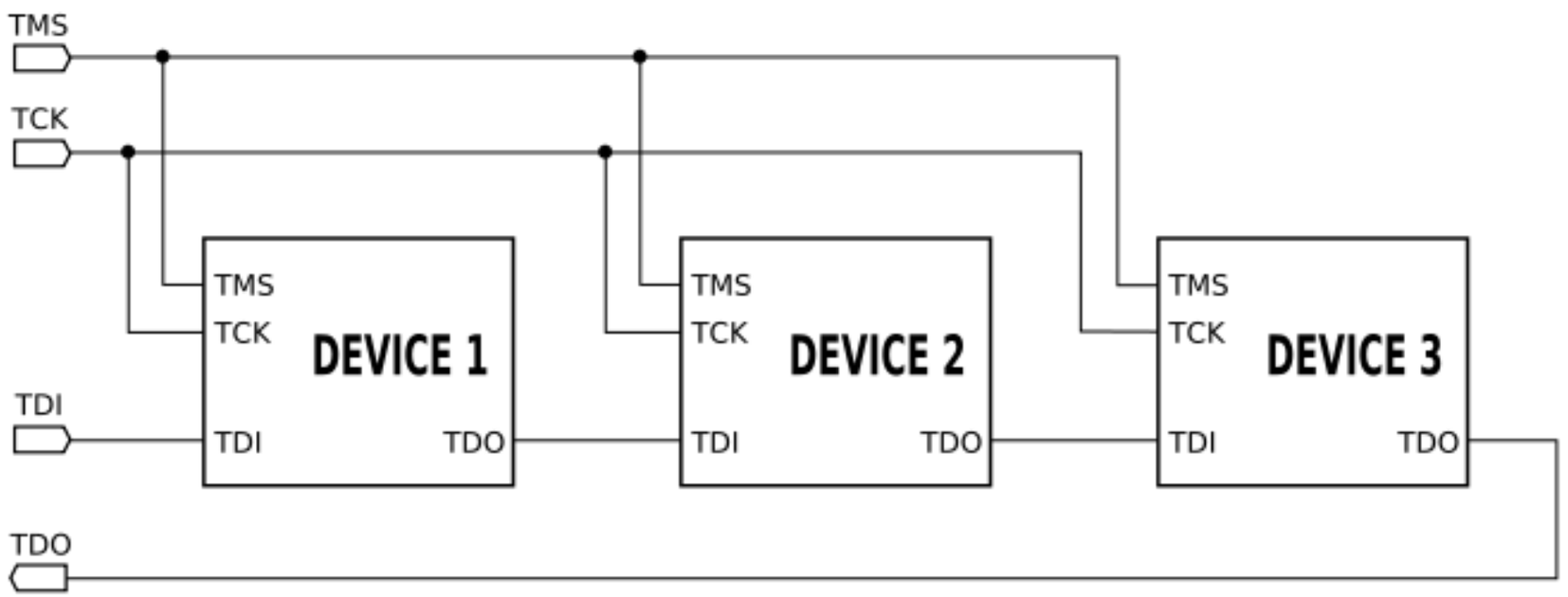
- Programming an FPGA consists in telling each component in each slice what its function is.
    - How will the multiplexers behave? What values are stored in the LUTs? Are the shift registers active?

- It also defines which connections are made in the switch matrix between the CLB.

JTAG

- What allows us to program an FPGA is called JTAG, a serial protocol that shifts data in and out of the devices it connects to.

- To program an FPGA, the design file is shifted inside the SRAM memory of the FPGA which tells each component how to act.



“Permanent” configuration

- As the FPGA uses an SRAM to describe its behaviour, the data is lost whenever the power is lost.

- In order to avoid manual reconfiguration of the FPGA each time we turn it on, it is also possible to store the design files in a non-volatile memory outside the FPGA called the Flash memory.

- On power up, the FPGA will try to get data out of the Flash memory if it is present in order to configure itself.

FPGA Design

- Designing for FPGAs is like playing with LEGOs: you have basic building blocks that you assemble in order to form a complex architecture.

- You do not program for an FPGA, but you design with an FPGA.

Exercise

- Using the building blocks of the FPGA, solve the following problem: the FPGA is fed a clock signal and a data signal (changing at the same frequency as the clock but the phase is not defined). How can you avoid sampling the data signal at the moment it changes (invalid data)?

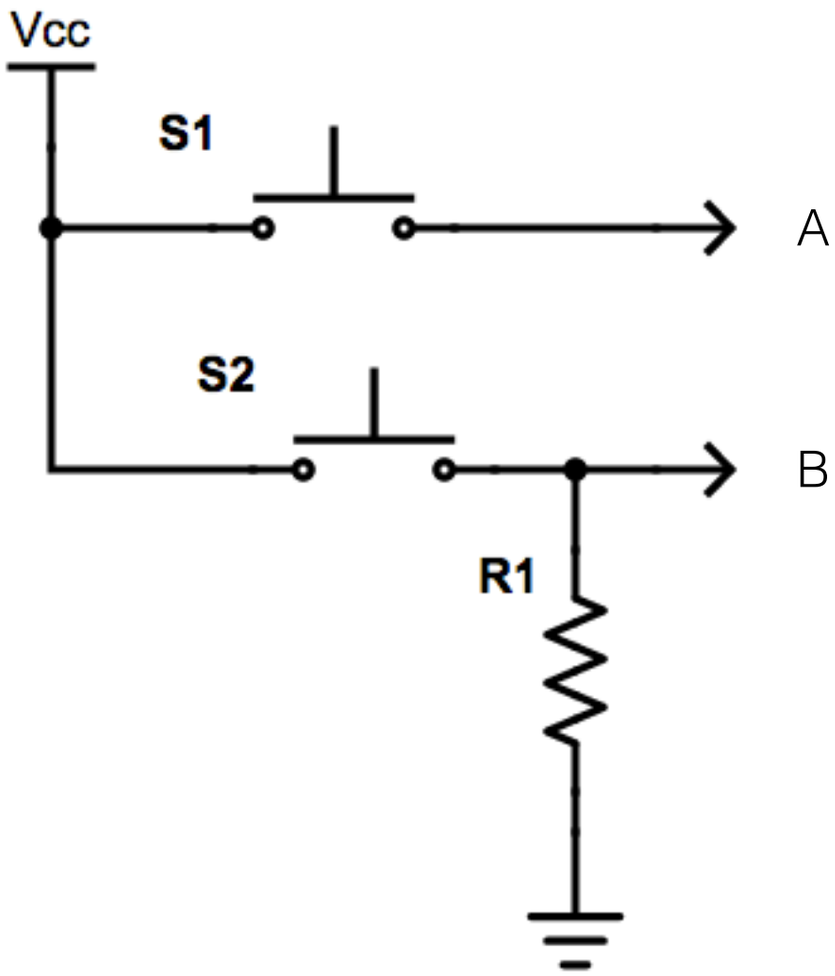
FPGA in the real world

What to do with an FPGA

- Now that we know what composes an FPGA, we can integrate it in a real world electronic design.

- But how do we connect an FPGA to the outside world?

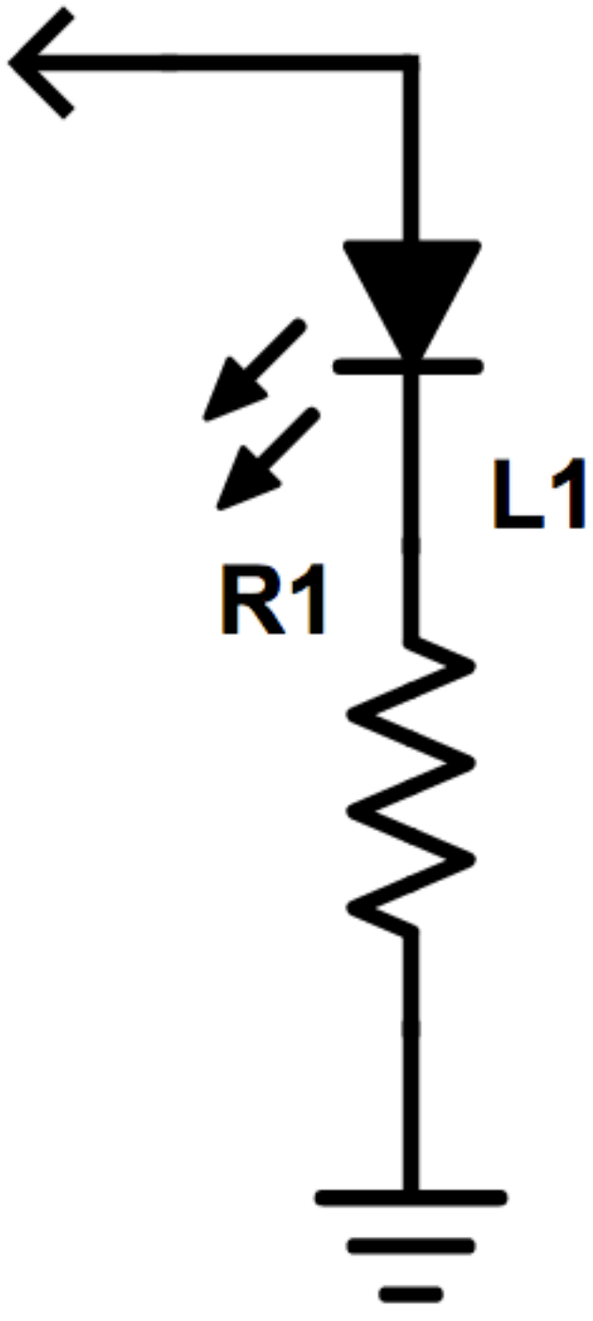
Buttons



- In the top example, the output A of the circuit is unstable: what is its value when the button is NOT pressed?

- In order to not leave signals floating, a pull-down resistor is placed between the button and the FPGA to ground the signal.

LEDs



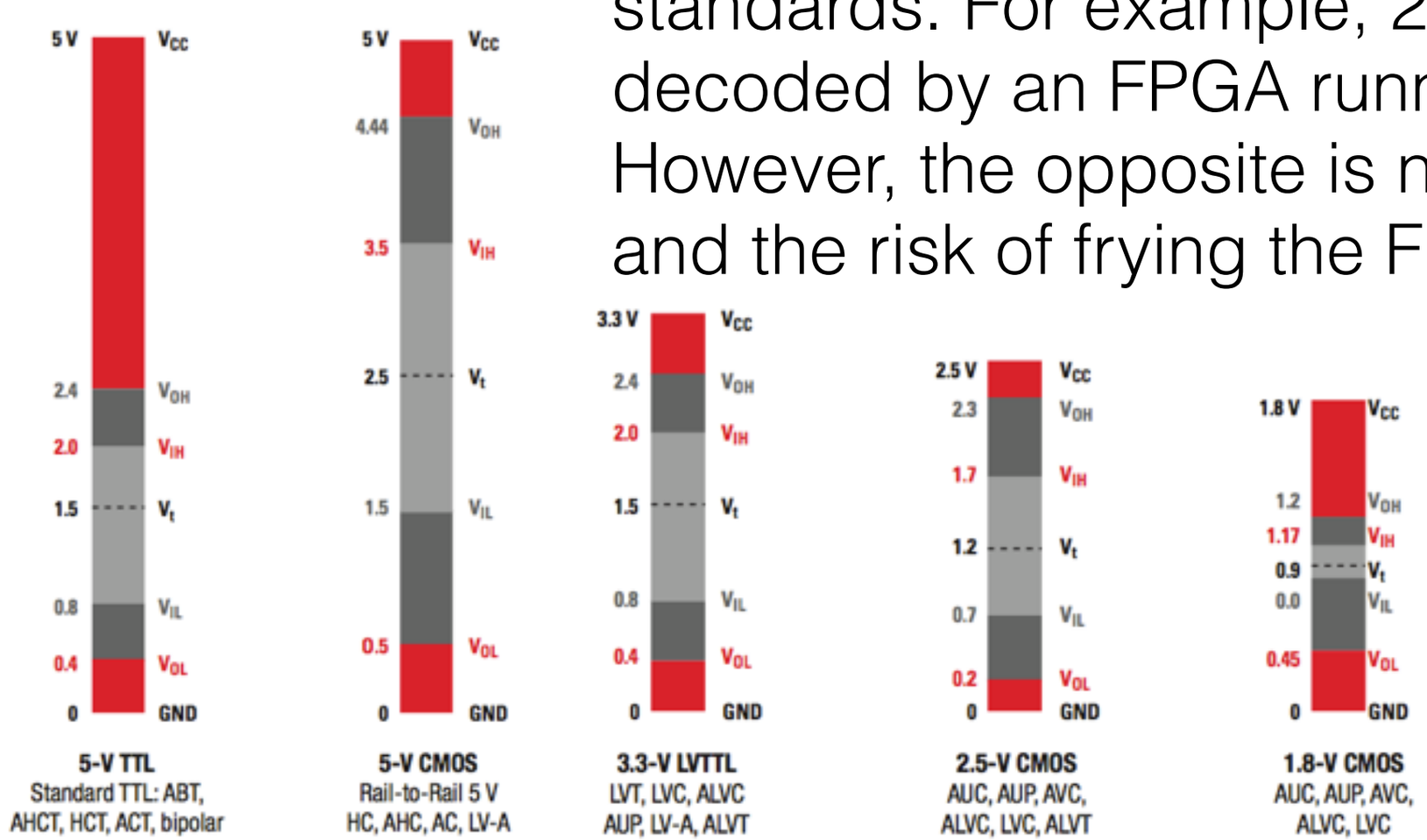
- The FPGA output pins set the PCB tracks to a given voltage and can deliver a small amount of current to the circuit.

- Input pins accept small amounts of current but will fry if a high current is forced through them.

- FPGAs work using voltage driven logic and not current driven logic (like NIM).

Logic levels

- An FPGA functions using a given supply voltages (3.3V, 2.5V, …) but it can understand a variety of logic levels standards. For example, 2.5V logic can be decoded by an FPGA running at 3.3V. However, the opposite is not always true and the risk of frying the FPGA arises.



Interface to other components

- The interface to other components becomes simple when the ICs use the same logic levels as the FPGA.

- The physical connections between chips is a set of copper PCB tracks.

- What requires more work is to decode/encode the signals on those tracks in order to make the ICs talk to each other.

Communication protocol

8.5 Programming

8.5.1 DAC8811 Input Shift Register

The DAC8811 has a 3-wire serial interface ($\overline{CS}$, SCLK, and DIN) compatible with SPI, QSPI, and Microwire interface standards, as well as most DSPs. See Figure 1 for an example of a typical write sequence.

The input shift register is 16 bits wide, as shown in Figure 25. The write sequence begins by bringing the $\overline{CS}$ line low. Data from the DIN line are clocked into the 16-bit shift register on each falling edge of CLK. The serial clock frequency can be as high as 50 MHz, making the DAC8811 compatible with high-speed DSPs. On the 16th falling edge of the serial clock, the last data bit is clocked in and the programmed function is executed.

At this point, the $\overline{CS}$ line may be kept low or brought high. In either case, it must be brought high for a minimum of 20 ns before the next write sequence so that a falling edge of $\overline{CS}$ can initiate the next write sequence.

Figure 24. Data Input Register

DB15															DB0
D15	D15	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

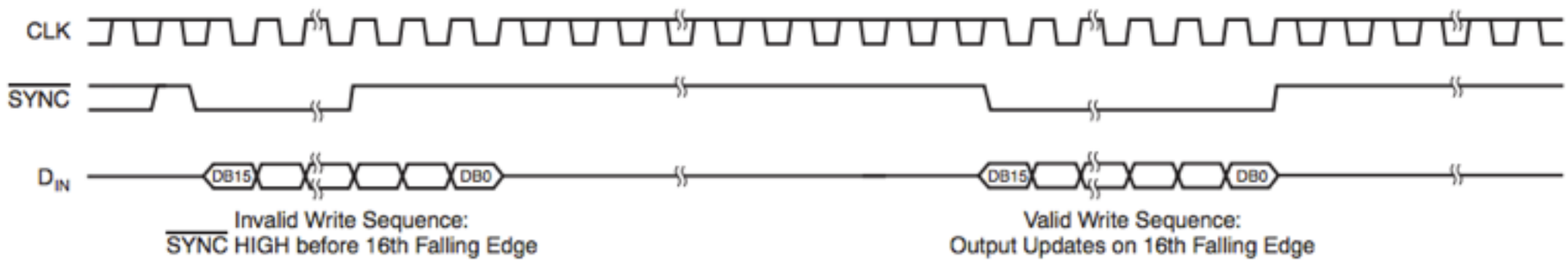


Figure 25. $\overline{SYNC}$ Interrupt Facility

What's next

- After this overview of the components present inside an FPGA, we will learn how to use them.

- We will first have a look at the development tools available and then go step-by-step through the process of implementing a design on an FPGA.

Ressources

- Spartan6 CLB: http://www.xilinx.com/support/documentation/user_guides/ug384.pdf
- Spartan6 DSP48A1: http://www.xilinx.com/support/documentation/user_guides/ug389.pdf
- Spartan6 Clocking: http://www.xilinx.com/support/documentation/user_guides/ug382.pdf
- Spartan6 BlockRAM: http://www.xilinx.com/support/documentation/user_guides/ug383.pdf
- Spartan6 SelectIO: http://www.xilinx.com/support/documentation/user_guides/ug381.pdf

Schematics

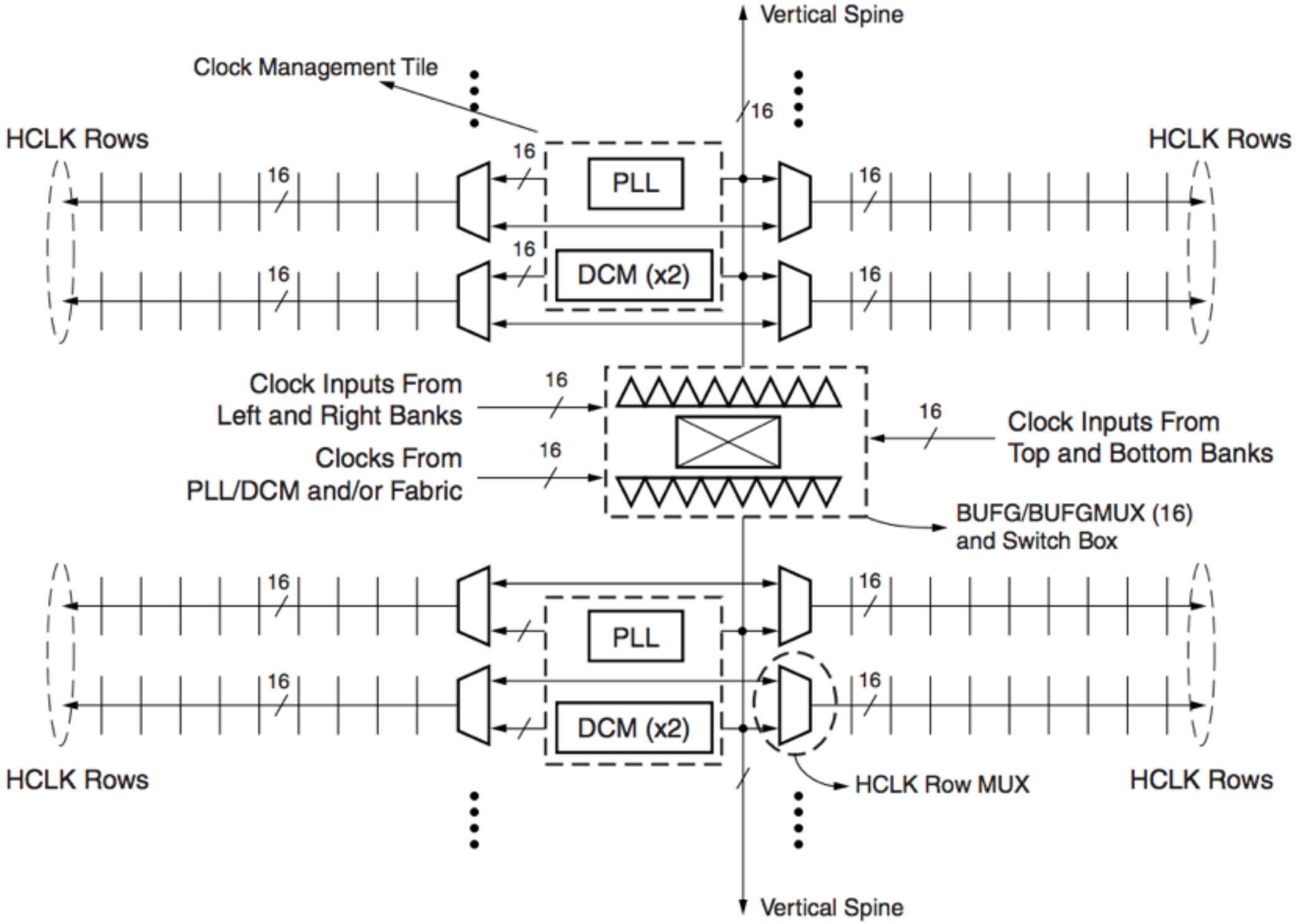


Figure 1-1: **Spartan-6 FPGA Global Clock Structure**

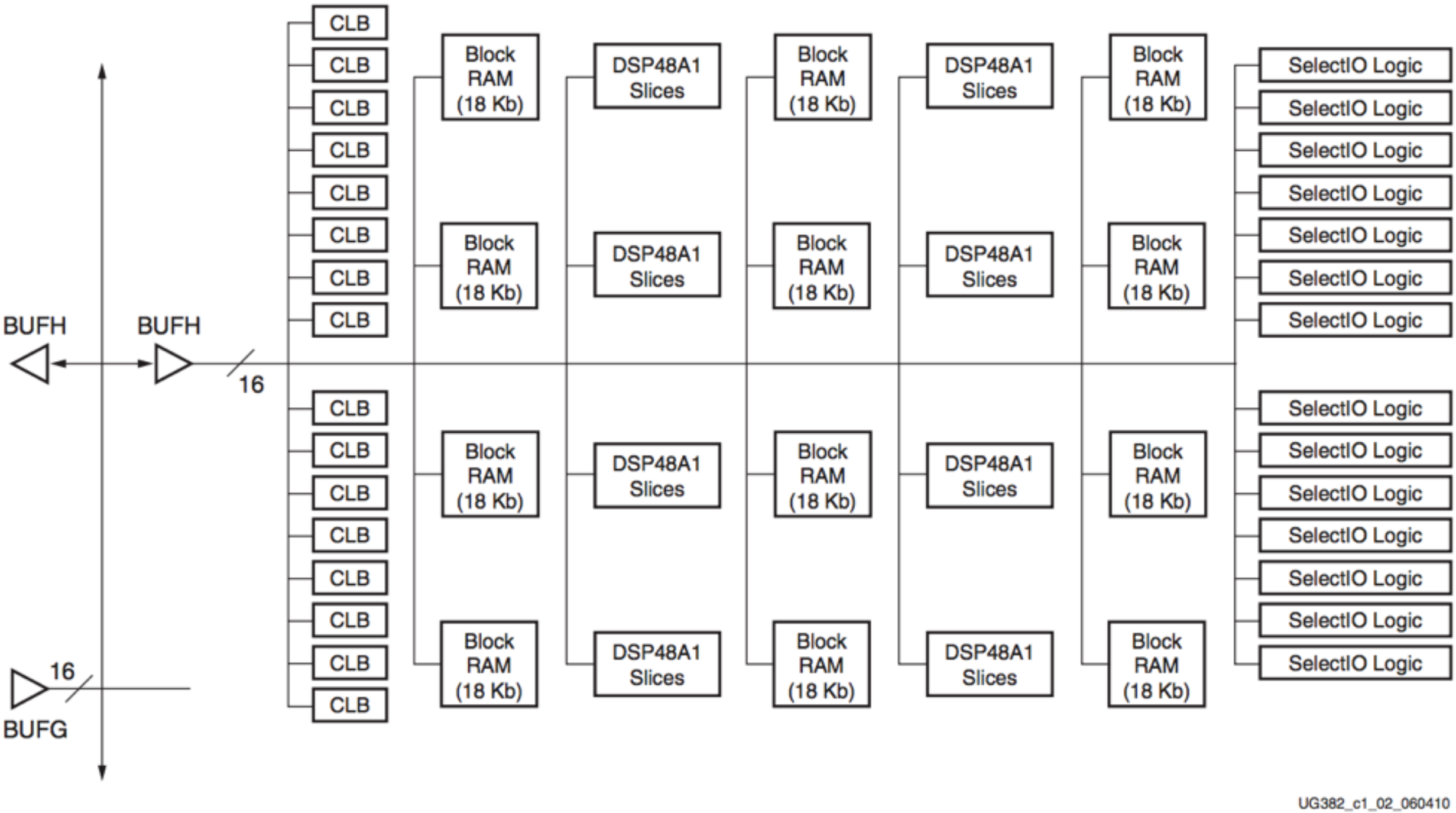


Figure 1-2: **BUFH Routing**

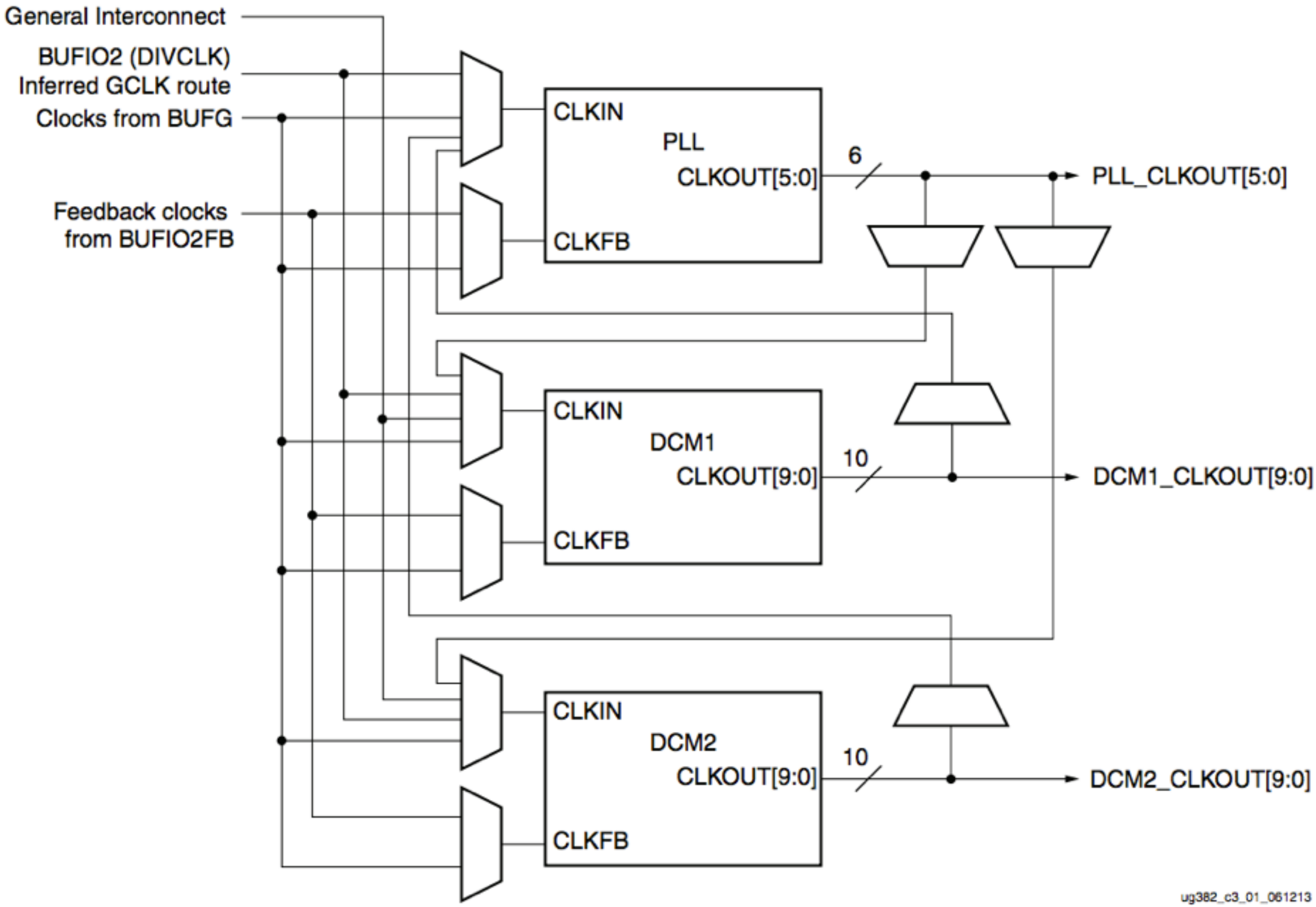


Figure 3-1: **Block Diagram of the Spartan-6 FPGA CMT**

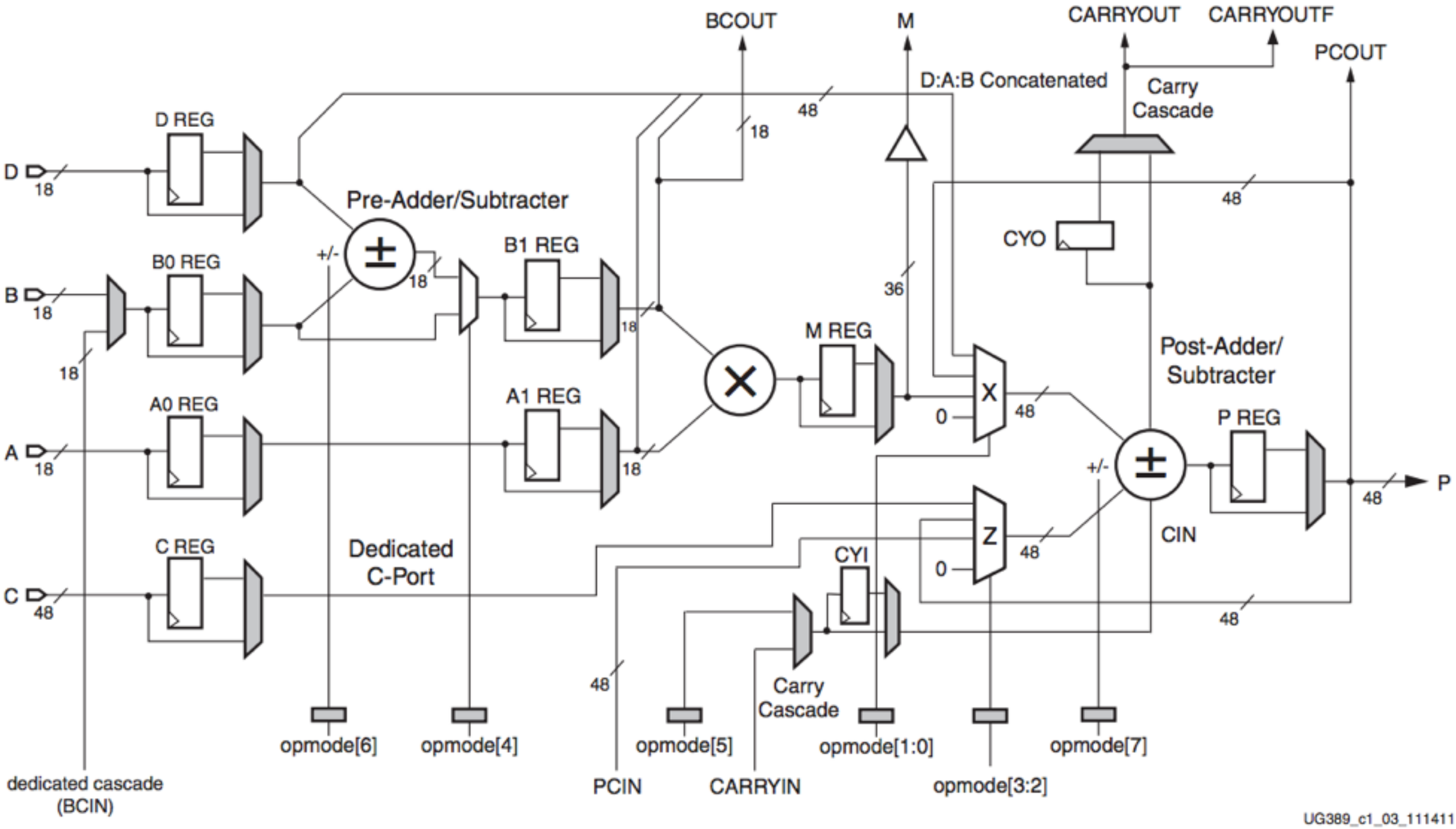


Figure 1-3: **DSP48A1 Slice**