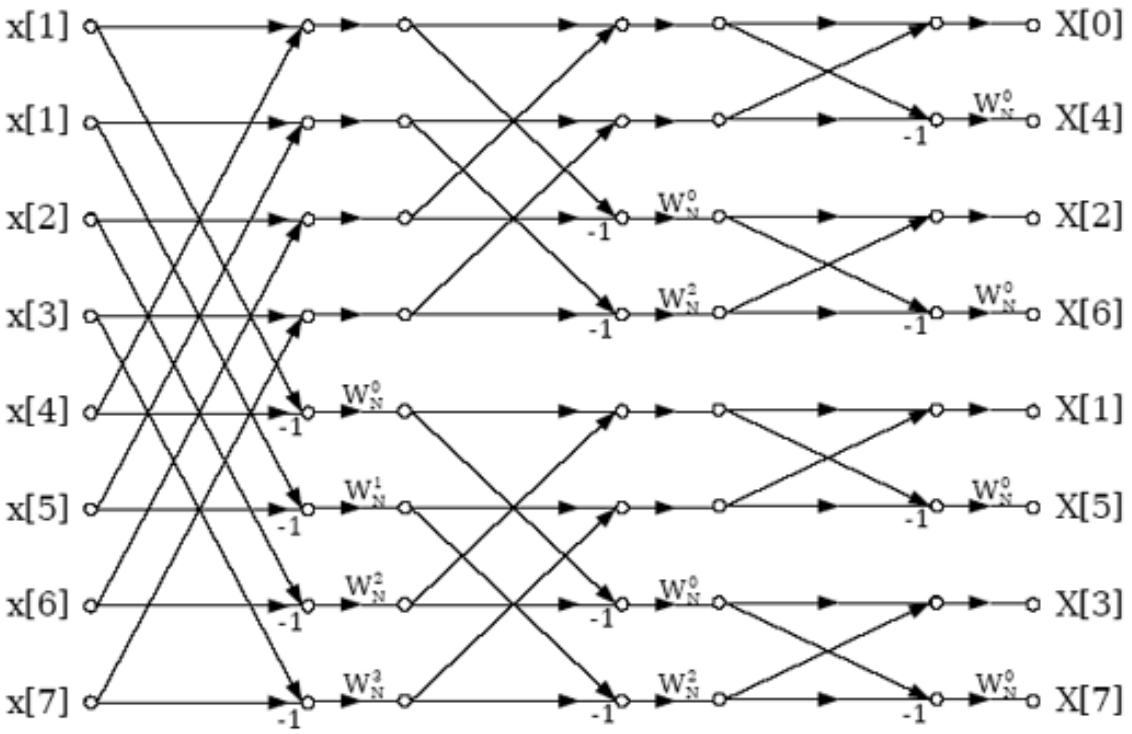
Unit 1 lab : **Multi-processor Architectures**

Exercise 1

- Implementation of a 512-points FFT with a single core solution
  - 512-point FFT according to the algorithm below
  - Generation in Matlab of test vectors and golden patterns
    - Matlab folder
  - C++ source file → complex data
    - std::complex<float>



Exercise 1

- Phases of the exercise (check the sources folder provided):
  - Study and analysis of the matlab model for the FFT
    - *Understand all the computations according to the algorithm*
    - *Understand the input data (test vector) and golden patterns generated*
  - Study of the C model
    - *Understand all the operations in the algorithm's model, in correspondence with previous Figure and matlab model*
  - Introduction to Vivado design environment; creation of a project to implement the FFT in Vivado/SDK
  - Develop the main.cc file with the implementation of a FFT, associated to the vector tests previously generated, and validated with the golden patterns
    - *Integration and verification of the C model in Vivado for a single core solution*
  - Measure the performance of the implementation
    - *Use of the function XTime_GetTime to obtain the run time of the FFT*

Exercise 2

- Accelerating the 512-points FFT by using the Neon GPU
- Use of the Ne10 library
- List of functions from the Ne10 library to be used:
  - ne10_init()
  - ne10_fft_alloc_c2c_float32_neon()
  - Initialization of pointers src and dst
    - *psrc_ddr = (ne10_fft_cpx_float32_t *) (cfg->buffer + (sizeof (ne10_fft_cpx_float32_t) * (fftSize)))*
    - *pdst_ddr = (ne10_fft_cpx_float32_t *) (psrc_ddr + (sizeof (ne10_fft_cpx_float32_t) * (fftSize)));*
  - Initialization of the src buffer
    - *psrc_ddr[i] = (ne10_fft_cpx_float32_t) (complex_data);*
  - ne10_fft_c2c_1d_float32_neon()

- Check the sources folder. The FFT can be validated in SDK by using the same input data vectors and golden patterns as before in Exercise 1

Exercise 2

- Procedure to make Neon library visible in a SDK project:
  - Add the library to the gcc linker:
    - *-l Ne10*
    - *-L ../Ne10-standalone/build/modules/*
  - Add the library included in Path and Symbols (C/C++ General):
    - *../Ne10-standalone/inc/*
  - If you have any problem with compilation, change the sentence of the Miscellaneous:
    - *-mcpu=cortex-a9 -mfpu=neon -mfloat-abi=hard*
  - You have to make sure that in Directories, you can view the directory: /Ne10-standalone/inc
  - You may not have enough heap memory…
    - *Change the properties of the linker script*
- Measure the run time as in Exercise 1 for comparison

Exercise 3

- Synchronization of a dual-core solution with shared memory
- Use of mutex described in the following files to manage memory access:
  - mutex.S mutex_blk.h
  - Functions:
    - *lock_mutex*
    - *unlock_mutex*
- Suitable division of the DDR memory in the linker script in the SDK project between the two cores
  - Range for core 0: origin $1000000 len $1000000
  - Range for core 1 : origin $2000000 len $1000000
  - Range shared by both cores: origin $3000000 len $1000000
- Confirm that both SDK projects for core 0 and 1 share the same compiler options and common variables definition
- Note that shared memory cannot be cacheable, according to Xilinx specifications. This may be needed to be configured by the corresponding function

Exercise 3

- Modifications in the linker script (lscript.ld):

```
MEMORY
{
   ps7_ddr_0 : ORIGIN = 0x1000000, LENGTH = 0x1000000
   ps7_ram_0 : ORIGIN = 0x0, LENGTH = 0x30000
   ps7_ram_1 : ORIGIN = 0xFFFF0000, LENGTH = 0xFE00
   ps7_mem_mutex  : ORIGIN = 0x03000000, LENGTH = 0x01000000
}
```

```
.mem_mutex : {
   __mem_mutex_start = .;
   *(.mem_mutex)
   __mem_mutex_end = .;
} > ps7_mem_mutex

_SDA_BASE_ = __sdata_start + ((__sbss_end - __sdata_start) / 2 );

_SDA2_BASE_ = __sdata2_start + ((__sbss2_end - __sdata2_start) / 2 );
```

- Application to a certain computation
  - Multiplication of 100 data by a constant in a single core
  - Multiplication of 100 data by a constant in two cores (50 each)
  - Comparison of timing performances

- Other application?

Unit 1 lab : **Multi-processor Architectures**