TE506I - Digital Systems Design

Didier Meier

didier.meier@intervenants.efrei.net

2021-2022

Objectives of the module

- Design and analyze a logic circuit (sequential and combinatorial) and a state graph and then perform simulations and syntheses on a programmable component
- Distinguish the use cases of a programmable component from CPU/GPU/ASIC/MCU
- Configure an FPGA and know the different steps of an EDA tool
- Describe, simulate and synthesize a combinatorial and sequential system using the VHDL
- Apply the essential rules of logic circuit design
- Master the essential techniques of optimization of programmable components

Summary – Digital Design

- Part 1: from digital systems to components with programmable architectures (Lectures: approx. 4h)
  - Reminder of combinatorial, sequential, synchronous, asynchronous system definitions... to generic structures
  - From the generic architecture of a synchronous sequential system to the architecture of a programmable component: PLD and CPLD
  - Evolution from CPLD architectures to FPGA architectures

- Part 2: The use of a hardware description language for the simulation and synthesis of digital systems: VHDL (Lectures: approx. 4h)
  - First approach: VHDL synthesis of an oriented graph
  - Introduction to VHDL
  - Structure of the VHDL language
  - Signals, variables and processes in VHDL
  - Typing objects in VHDL

- Part 3: How to describe, synthesize and simulate combinatorial digital systems in VHDL (Approx. 2 Labs of 3h each)
  - Combinatorial logic
  - Sequential logic

Summary – Digital Design

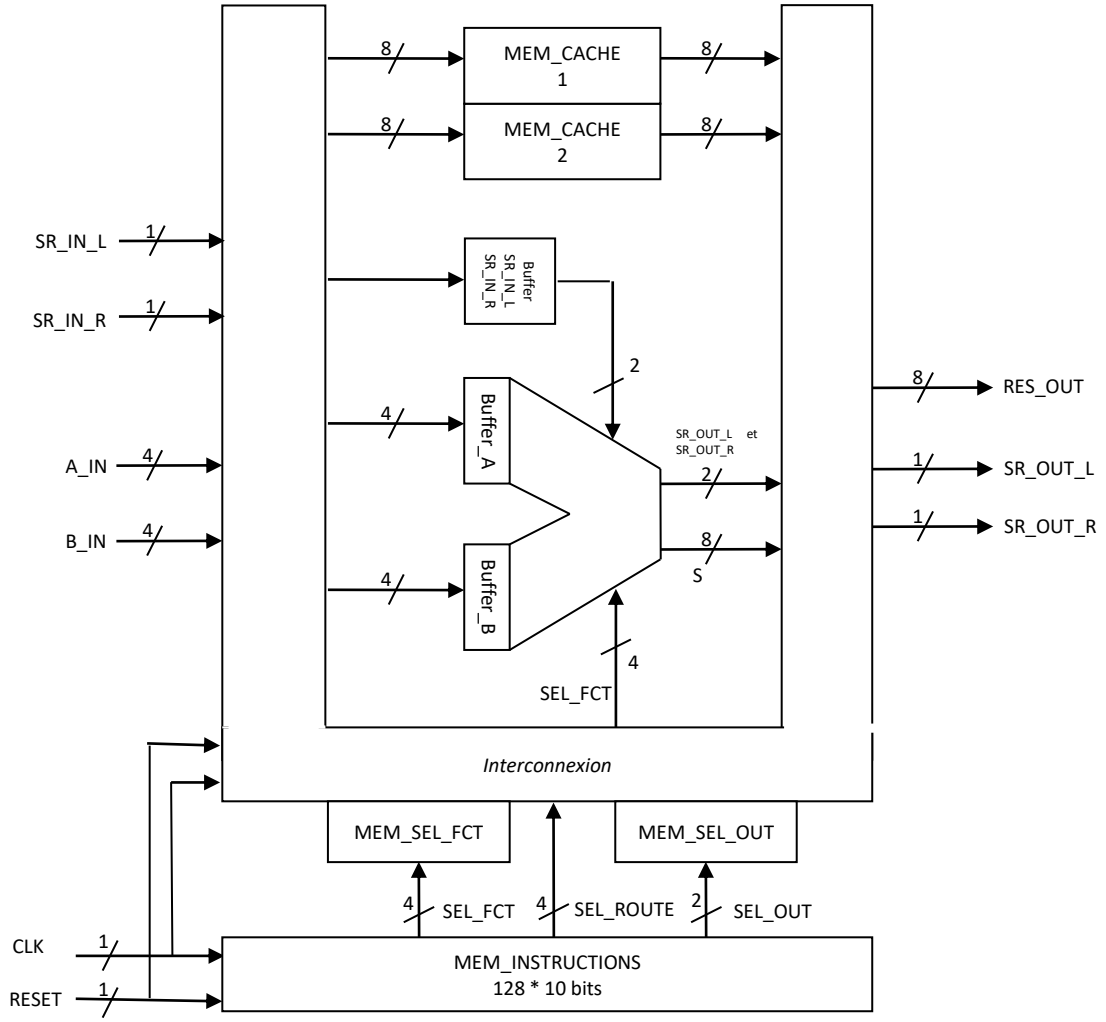
- Part 4: optimization of digital systems on programmable components; architectures and VHDL language (Lectures: approx. 2h)
  - Performance
  - Space & Resources
  - Energy
- Part 5: Programmable Architecture Optimization Exercises (Exercices : approx. 3h)
  - Adders
  - Multipliers
  - Sequential Systems (Fmax)
- Part 6: design, simulation and synthesis of a microcontroller using VHDL in a programmable architecture component (PRJ: approx. 9h)
  - Presentation of the subject and the target operation
  - Theoretical design of the targeted architecture
  - Synthesis in VHDL
  - Performing simulations (testbench) in VHDL

Part 6: Project

Design, simulation and synthesis of a microcontroller using VHDL in a programmable architecture component

Overview of the target architecture

- The objective of this project is to create a microcontroller core integrating:
  - A unit of arithmetic and logical processing
  - Internal computational memories
  - An instruction memory
- This microcontroller core will operate on 4 bits for inputs and on 8 bits for outputs and internal memories.
- From basic operations, it is possible to perform all logical functions and arithmetic operations using state machines (sequences of instructions).
  - The more "complex" functions are only a succession of simple functions with sometimes the memorization of intermediate results.
  - It is therefore necessary to add to the basic structure
    - ✓ An internal memory to store intermediate results
    - ✓ An internal memory to store the successive instructions to be carried out
- The last work will consist of making tree automata and implementing them in the instruction memory.
  - For this last part, you will be required to implement the tree state machines to control the UAL in order to perform the chosen functions



Functions carried out by the UAL

SEL_FCT[3]	SEL_FCT[2]	SEL_FCT[1]	SEL_FCT[0]	Meanings
0	0	0	0	nop (no operation) S = 0 | SR_OUT_L = 0 et SR_OUT_R = 0
0	0	0	1	S = A | SR_OUT_L = 0 et SR_OUT_R = 0
0	0	1	0	S = B | SR_OUT_L = 0 et SR_OUT_R = 0
0	0	1	1	S = not A | SR_OUT_L = 0 et SR_OUT_R = 0
0	1	0	0	S = not B | SR_OUT_L = 0 et SR_OUT_R = 0
0	1	0	1	S = A and B | SR_OUT_L = 0 et SR_OUT_R = 0
0	1	1	0	S = A or B | SR_OUT_L = 0 et SR_OUT_R = 0
0	1	1	1	S = A xor B | SR_OUT_L = 0 et SR_OUT_R = 0
1	0	0	0	S = Right shift A (avec SR_IN_L) | SR_IN_L for the input bit and SR_OUT_R for the output bit
1	0	0	1	S = Left shift A (avec SR_IN_R) | SR_IN_R for the input bit and SR_OUT_L for the output bit
1	0	1	0	S = Right shift B (avec SR_IN_L) | SR_IN_L for the input bit and SR_OUT_R for the output bit
1	0	1	1	S = Left shift B (avec SR_IN_R) | SR_IN_R for the input bit and SR_OUT_L for the output bit
1	1	0	0	S = A + B binary adder with SR_IN_R as carry in
1	1	0	1	S = A + B binary adder without carry in
1	1	1	0	S = A – B binary substraction
1	1	1	1	S = A * B binary multiplication

Memories

- Memories Buffer_A and Buffer_B
  - Les mémoires Buffer_A, Buffer_B permettent de stocker les données directement liées au cœur de l'UAL, c'est-à-dire à la sous-fonction arithmétique et logique.
  - Elles seront chargées (activées sur front montant de l'entrée clk) suivant les valeurs de l'entrée SEL_ROUTE.

- Memories MEM_SR_IN_L and MEM_SR_IN_R are used to store the values of input carries and shift bits
  - They are systematically stored on each rising front of the CLK clock.

- Memory MEM_SEL_FCT allows you to memorize the arithmetic or logical function to be performed
  - It is systematically loaded at each front clock mount.

- Memories MEM_CACHE_1 and MEM_CACHE_2
  - The MEM_CACHE_1 and MEM_CACHE_2 memories are used to store the data of the inputs A_IN and B_IN or the intermediate results of the output S, that is, at the output of the arithmetic and logical sub function.
    - ✓ It should be noted that SR_OUT_L and SR_OUT_R outputs cannot be memorized.
    - ✓ These two memories will be loaded (activated on the up front of the CLK input) according to the values of the SEL_ROUTE input.

- Memory MEM_SEL_OUT
  - The memory MEM_SEL_OUT is used to store the data of the SEL_OUT input.
  - It is systematically loaded at each rising front of the CLK clock.

- Memory MEM_INSTRUCTIONS
  - The memory MEM_INSTRUCTIONS allows to store the successive instructions to be carried out
  - Its pointer is systematically indented at each rising front of the CLK clock

- SEL_ROUTE allows you to define the data transfer that will be carried out during the next clock cycle (next front rising of the clock).
  - It should therefore not be memorized as MEM_SEL_FCT and MEM_SEL_OUT

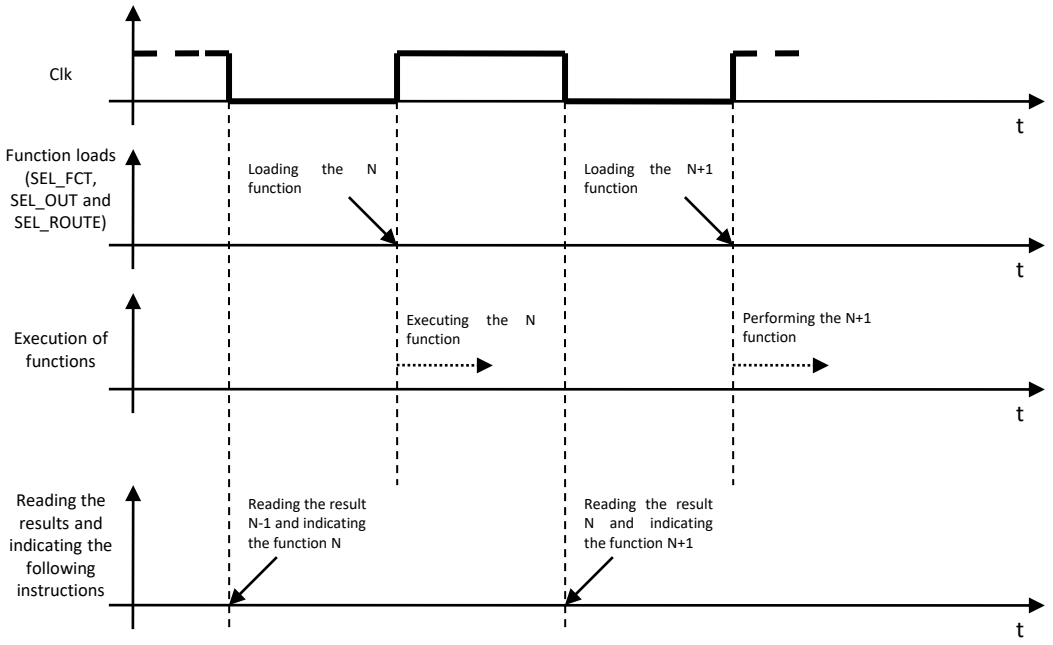
Interconnections and data routing

SEL_ROUTE[3]	SEL_ROUTE[2]	SEL_ROUTE[1]	SEL_ROUTE[0]	Meanings
0	0	0	0	Input A_IN storage in Buffer_A
0	0	0	1	MEM_CACHE_1 storage in Buffer_A (4 LSB bits)
0	0	1	0	MEM_CACHE_1 storage in Buffer_A (4 MSB bits)
0	0	1	1	MEM_CACHE_2 storage in Buffer_A (4 LSB bits)
0	1	0	0	MEM_CACHE_2 storage in Buffer_A (4 MSB bits)
0	1	0	1	S storage in Buffer_A (4 LSB bits)
0	1	1	0	S storage in Buffer_A (4 MSB bits)
0	1	1	1	Input B_IN storage in Buffer_B
1	0	0	0	MEM_CACHE_1 storage in Buffer_B (4 LSB bits)
1	0	0	1	MEM_CACHE_1 storage in Buffer_B (4 MSB bits)
1	0	1	0	MEM_CACHE_2 storage in Buffer_B (4 LSB bits)
1	0	1	1	MEM_CACHE_2 storage in Buffer_B (4 MSB bits)
1	1	0	0	S storage in Buffer_B (4 LSB bits)
1	1	0	1	S storage in Buffer_B (4 MSB bits)
1	1	1	0	S storage in MEM_CACHE_1
1	1	1	1	S storage in MEM_CACHE_2

SEL_OUT[1]	SEL_OUT[0]	Meanings
0	0	No output : RES_OUT = 0
0	1	RES_OUT = MEM_CACHE_1
1	0	RES_OUT = MEM_CACHE_2
1	1	RES_OUT = S

Time synchronization

- The diagram opposite shows the temporal operation that the assembly will have to respect
  - In case of external control (without instruction memory) for testing
  - Internally with instruction memory
- External components using this UAL change the input values on the falling edge of the CLK clock.
  - At the next rising edge of clock, the UAL executes the instruction.
  - The results are read at the next falling edge (while indicating the new function to be performed).
- The RESET input is used to asynchronously initialize all memories to 0



Instructions for testing and validation

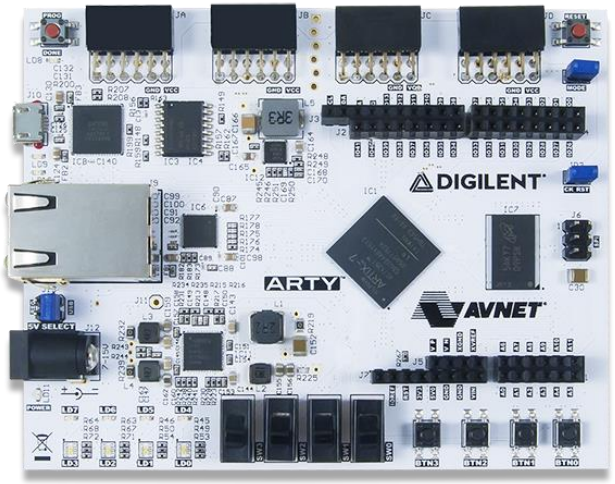
- You will have to implement 3 specific functions in the instruction memory:
  1) RES_OUT_1 = (A mult. B) *(RES_OUT_1 on 8 bits)*
  2) RES_OUT_2 = (A add. B) xnor A *(xnor on the 4 LSB – RES_OUT_2 on 4 bits)*
  3) RES_OUT_3 = (A_0 and B_1) or (A_1 and B_0) *(RES_OUT_3 on the LSB)*

- In a 1st step, we consider the inputs A, B, SR_IN_L and SR_IN_R stable throughout the duration of the calculation:
  - Which, in reality, will not always be the case…

- The output RES_OUT will remain at 0 as long as the final result of the calculation is not available (no intermediate results on RES_OUT)
  - Once the result is calculated, it will be held on the output RES_OUT until the next start of the calculation
    - ✓ For tests on the FPGA board, a specific signal will also be increased from 0 (calculation in progress) to 1 (calculation completed) to indicate that the result RES_OUT is available

Definition of the lab environment

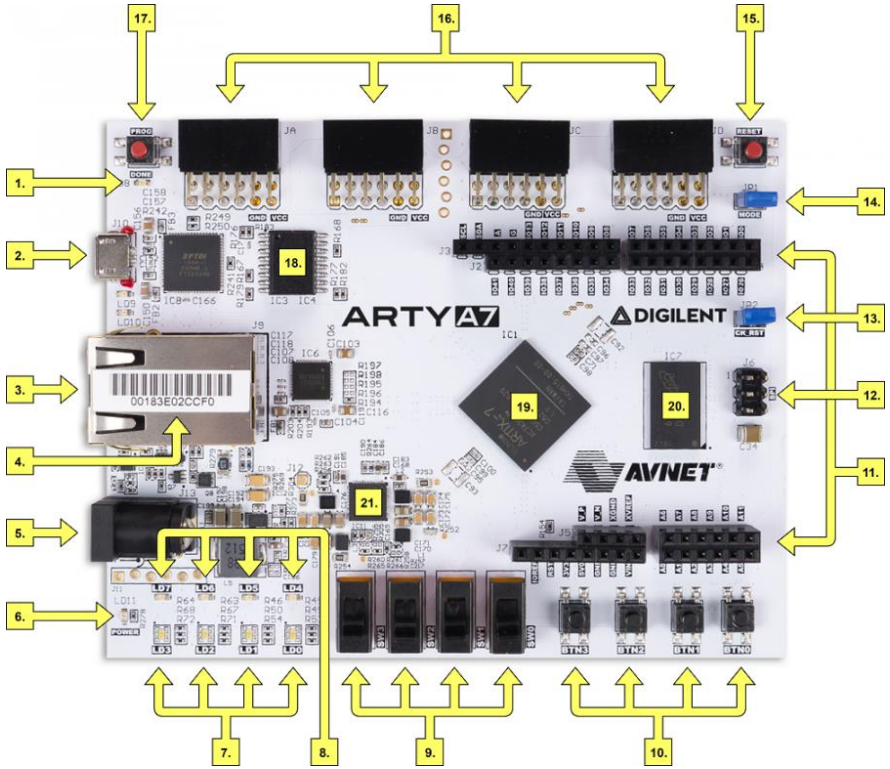
- Development and simulation environment:
  - EDA Playground
  - Xilinx Vivado 2018.2 (or later version)



- Development Board:
  - ARTY from Digilent
    - ✓ Xilinx Artix-35T FPGA
  - The development board is not necessary for the realization of the project
    - ✓ Nevertheless, it will be used to achieve the implementations during the last project session

Integration on the ARTY development board

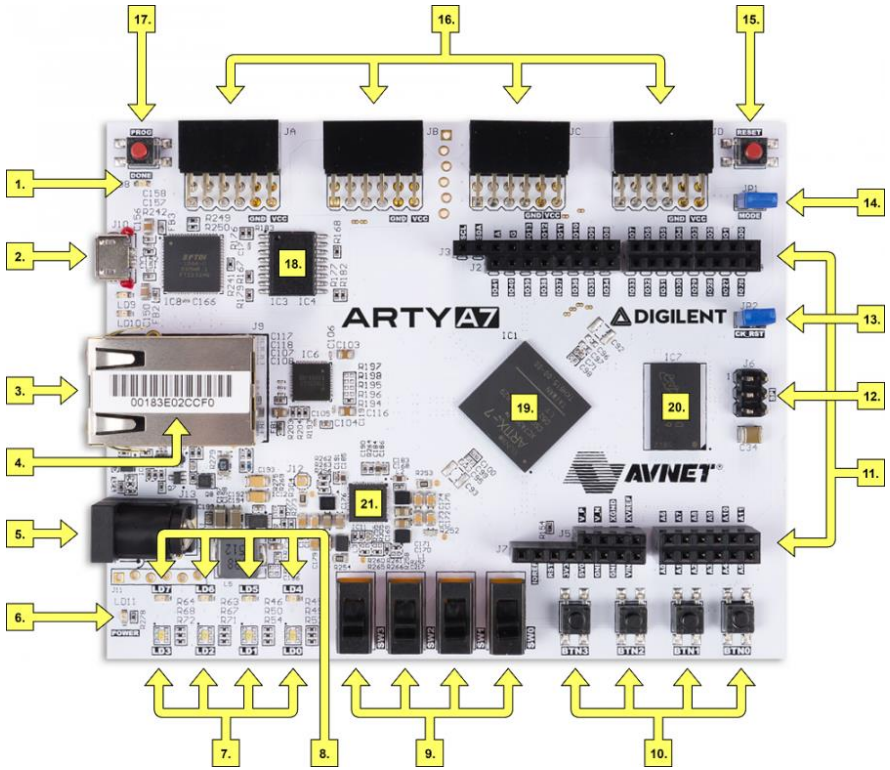
- The development board



Callout	Description	Callout	Description	Callout	Description
1	FPGA programming DONE LED	8	User RGB LEDs	15	chipKIT processor reset
2	Shared USB JTAG / UART port	9	User slide switches	16	Pmod connectors
3	Ethernet connector	10	User push buttons	17	FPGA programming reset button
4	MAC address sticker	11	Arduino/chipKIT shield connectors	18	SPI flash memory
5	Power jack for optional external supply	12	Arduino/chipKIT shield SPI connector	19	Artix FPGA
6	Power good LED	13	chipKIT processor reset jumper	20	Micron DDR3 memory
7	User LEDs	14	FPGA programming mode	21	Dialog Semiconductor DA9062 power supply

Integration on the ARTY development board

- Performing tests on the development board:
  - Clock at 100 MHZ
  - A = B
    - ✓ Swt(3-0) : User slide switches
  - SR_IN_L et SR_IN_R = 0 (not used for testing)
  - Global Reset
    - ✓ btn(0) : User push buttons
  - Start calculations:
    - ✓ btn(1) : RES_OUT_1
    - ✓ btn(2) : RES_OUT_2
    - ✓ btn(3) : RES_OUT_3
  - Calculation results:
    - ✓ 8 leds (colour = rouge)
    - ✓ Available result : 8th led with green
    - ✓ SR_OUT_L et SR_OUT_R (not connected)



Description of the global entity and constraint file

- For implementation on FPGA, you will need to follow the description of the global entity so that the input and output ports can match with the development board:
  - The entity description is available for download on Moodle
- In order to perform fpga board testing, you will need to use the constraint file specific to the development board:
  - The constraints file is available for download on Moodle

Expected Deliverables and Evaluation

- Definition of deliverables (reports and archives) and expected results (validations)
  - At the end of the period dedicated to the realization of the project, your results will be evaluated by combining:
    - ✓ A technical validation (integration of the assembly in a programmable component and realization of an automatic sequence of tests).
    - ✓ Theoretical validation in the development environment (using your project archives and technical report) via different simulations.
    - ✓ Evaluation of your technical report of the project.
  - It is therefore essential to provide the various elements requested.
    - ✓ No delay will be tolerated.
    - ✓ Any missing element during the delivery of the archives may affect the evaluation of your project.
      - So be careful to scrupulously respect the requested elements.

- Deliveries of documents and archives
  - At the end of the period allotted to the realization of the project, you will be asked to provide:
    - ✓ 1 ZIP archive in « students_names" format .zip containing
      - All of your project file(s)
      - 1 file in "noms_des_élèves" format.txt containing a summary of each of your entities in VHDL
      - 1 detailed report of the mini-project in electronic format (.doc or .pdf).

- The archive must be uploaded on Moodle before the end of the project deadline
  - No delay will be tolerated
  - The transfer of these files by email is not accepted

End of Part 6

End of the Digital Systems Design Module