Department of Electronics and Telecommunications
Bachelor degree in Electronic Engineering

Digital Systems Electronics

LAB 3: Data Path Elements.

Due date: April 09, 2024
- extended to: April 11, 2024
Delivery date: April 11, 2024

Group: 19

Contributions:

- Valtorta Alexander James
- Tedesco Angelo
- Urgu Sara

Contents

0 Introduction

0.1 Aim of this project

This purpose of this lab is to design arithmetic circuits that add, subtract, and multiply binary numbers. This are fundamental operations usually implemented in a Data-Path (DP), which generally are sequential circuits that numerically process some kind of digital input stream.

0.2 Content and structure of the document

The report is divided into four sections, each for every exercise of the laboratory assignment. Moreover, each section contains the key parts in order to design and implement each circuit:
- Design Entry, which describes the circuit to be implemented on the FPGA device, in particular it is indicated the overall architecture of the circuit, by reporting the name of the source file or files and the key role of each of them.
- Functional Simulation, which describes the testbench and the approach used to assess the completed design.
- Synthesis, which shows the results of Modelsim functional simulation and those of the timing simulation. Here are also reported the timing analyzer results

1 4-bit Sequential RCA

1.1 Design entry

To describe the 4-bit version of the adder that supports signed numbers in 2's-complement form it is possible to observe that given two numbers A and B, their sum can either be positive or negative. However, if the sum results positive but the MSB (*S(3)*) is equal to '1', it means that the number is actually negative (being in 2's-complement form) thus the carry out (*Co*) should be equal to '0'; similarly if the sum is negative but the MSB is equal to '0', it means that the number is actually positive thus the carry out should be equal to '1'. Using a truth table to represent this reasoning it is possible to obtain the logic function for the overflow condition (which is exploited for most of the following designs of the circuits).

$A(3)$	$B(3)$	$S(3)$	C_o	OVF	
0	0	1	0	1	$A+B>0$
1	1	0	1	1	$A+B<0$
x	x	x	x	0	

$$OVF = \overline{(A(3)B3)}S(3)\overline{(C_o)} + A(3)B(3)C_o\overline{(S(3))}$$

The implementation is carried out by designing each component (except for the registers and the D-flipflop, defined in *regn.vhd* and *flipflop.vhd* respectively, that were already provided in the assignment) in a dedicated VHDL file:

- The full adder (defined in *f_adder.vhd*) is implemented by means of multiplexer (defined in *mux_2_1_1bit*, whose implementation was discussed for previous laboratories) and xor gate. In particular the multiplexer is defined in the structural architecture as a component and *v1* as internal signal, then, in the descriptive part of the architecture the inputs and output of the multiplexer are mapped onto those of the full adder (previously declared in the entity), the internal signal is associated to either the *a* or the *b* input of the full adder (by means of the xor operation), same for the *sum* output of the full adder associated to the *c_i* input of the full adder or *v1*.

- The Ripple Carry Adder (RCA) is defined in *rca_4bit.vhd* by means of a structural architecture implementing four full adders. In the declarative part the full adder (defined in *f_adder.vhd*) is defined as a component and *c1,c2,c3* are defined as internal signals so that, in the descriptive part the inputs and outputs of the full adders can be properly mapped to those of the RCA (defined in the entity) and the internal signals.

- In order to display the hexadecimal values of the inputs A and B, and of the sum S, the *hex_decoder.vhd* file consider all the possible 4-bit inputs and associates to the 7-bit output the suitable pattern of ones and zeros in order to lit the segments and display the corresponding hexadecimal value (remembering that a segment is lit when driven to the logic value '0'), in case of errors the display will be blank.

- The *signed_adder_4bit.vhd* file defines the completed 4-bit adder for signed numbers, with indication of overflow, by mapping the inputs and outputs values of the RCA, the flipflop and the registers (all declared as components) to those defined in the entity (*A4, B4, CR_in, CLK, RSTn, OVF, SUM4*) and the internal signals (*r1, r2, Q1, Q2, Y*).

- Lastly in the *signed_adder_map.vhd* it takes place the mapping of the inputs and outputs of the whole circuit (defined in *signed_adder_4bit.vhd*) to those assigned to the DE1 board, in order to visualize the hexadecimal numbers in the displays. In particular are assigned switches *SW3-0* and *SW7-4* to input the numbers A and B, the key *KEY0* as an active-low asynchronous reset input, the *KEY1* as a manual clock input, the red *LEDR9* to show overflow of the adder, and the seven segment displays *HEX0, HEX1, HEX2* to show the hexadecimal values of the A, B and the sum.

1.2 Functional simulation

With the testbenches it is possible to check the functionalities of the completed designs.

- In the *tb_fadder.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *f_adder.vhd*), and the internal signals for the simulation (*in_t*, *s_t* and *c_o_t*.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones, the simulation is performed by considering the possible switching combinations and waiting a short amount of time in between.

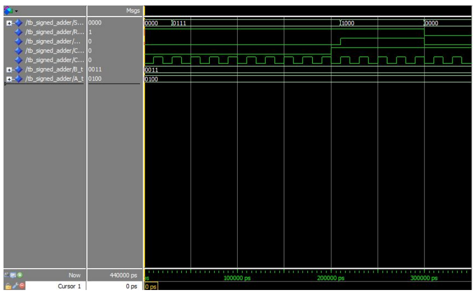
- In the *tb_rca4.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *rca_4bit.vhd*), and the internal signals for the simulation (*a_t*, *b_t*, *s_t*, *co_t* and *ci_t*.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones, the simulation is performed by assigning a 4-bit number to the internal signals *a_t* and *b_t*, then, in the process, associate to *ci_t* first the value '0' and then '1', waiting a short amount of time in between.

- In the *tb_signed_adder.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *signed_adder_4bit.vhd*), and the internal signals for the simulation (*A_t*, *B_t*, *SUM_t*, *CR_t*, *OVF_t*, *RSTn_t* and *CLK_t*, all initialized at "0000" or '0', depending if they are vectors or not.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones the simulation is performed through means of a two processes. In the first one the simulation of the sum and the behaviour of the reset are performed, where a value is assigned to the internal signals *A_t*

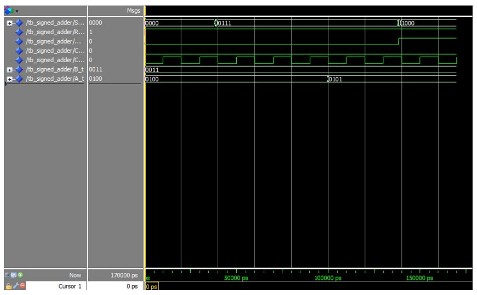
and *B_t* and the reset *RSTn_t* value is '1', then the value of *A_t* is changed twice, *CR_t* switches from '0' to '1' and finally the *RSTn_t* is '0' (waiting a short amount of time in between each commutation). In the second process it is considered the commutation of the clock *CLK_t*.

1.3 Synthesis

Once the circuit is successfully synthesized by Quartus it is possible to run a timing simulation, as well as the Modelsim functional simulation and see the results.

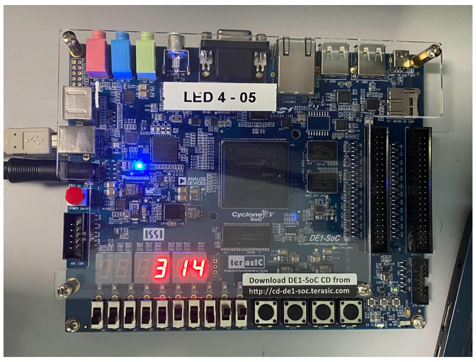


(a) Modelsim functional simulation

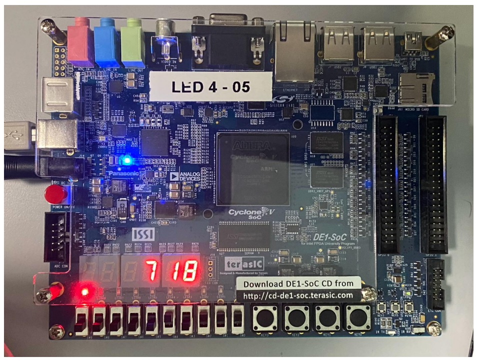


(b) Timing simulation

Figure 1: Simulations of the 4-bit signed adder

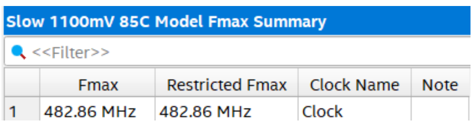


(a) Sum



(b) Sum with overflow

Figure 2: Results of sums

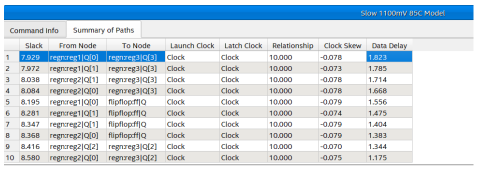


Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	482.86 MHz	482.86 MHz	Clock	

(a) Maximum frequency



Slow 1100mV 85C Model

Command Info | Summary of Paths

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	7.929	regn:reg1|Q[0]	regn:reg3|Q[3]	Clock	Clock	10.000	-0.078	1.823
2	7.972	regn:reg1|Q[1]	regn:reg3|Q[3]	Clock	Clock	10.000	-0.073	1.785
3	8.038	regn:reg2|Q[1]	regn:reg3|Q[3]	Clock	Clock	10.000	-0.078	1.714
4	8.084	regn:reg2|Q[0]	regn:reg3|Q[3]	Clock	Clock	10.000	-0.078	1.668
5	8.195	regn:reg1|Q[0]	flipflop:ff|Q	Clock	Clock	10.000	-0.079	1.556
6	8.281	regn:reg1|Q[1]	flipflop:ff|Q	Clock	Clock	10.000	-0.074	1.475
7	8.347	regn:reg2|Q[1]	flipflop:ff|Q	Clock	Clock	10.000	-0.079	1.404
8	8.368	regn:reg2|Q[0]	flipflop:ff|Q	Clock	Clock	10.000	-0.079	1.383
9	8.416	regn:reg2|Q[2]	regn:reg3|Q[2]	Clock	Clock	10.000	-0.070	1.344
10	8.580	regn:reg2|Q[0]	regn:reg3|Q[2]	Clock	Clock	10.000	-0.075	1.175

(b) Longest delay path

Figure 3: Results of Timing Analyzer

2 4-bit Sequential Adder/Subtractor

2.1 Design entry

In order to design a circuit that can perform both addition and subtraction of four-bit numbers, it is necessary to add an inverter to the circuit, so that when the subtraction is requested the sign of the second number (B) is inverted. That is because a subtraction is actually an addition of a first number and a second number with inverted sign: A-B=A+(-B).

The inverter is defined in *not_en.vhd* by means of a process in the behavioural architecture that assigns to the output O directly the input *I*, if the value of the enable signal *En* is '0', or the complement of the I if *En* is '1'.

Now the complete circuit (adder/subtractor) is defined in *signed_addsub_4bit.vhd*, which is simply an integration of the file *signed_adder_4bit.vhd* of the previous exercise. In particular the input *Sub* is added in the entity which allows to enable the addition if equal to '0' or the subtraction if equal to '1', there is also the addition of the inverter (defined in *not_en.vhd*) as a component and additional internal signals *Q3,Q3,Y,S* in the architecture, so that also the inputs and output of the inverter can be mapped along with the others' (previously described in the exercise 1).

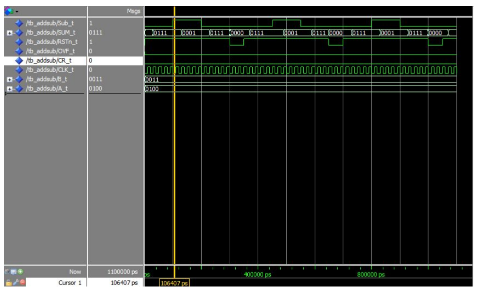
Finally for the mapping of the signals of the circuit to those assigned to the DE1 board it is only a matter of modify accordingly in the *signed_adder_map.vhd* file, defined for the previous exercise: the component defined in the declarative part of structural architecture is changed with the adder/subtractor (defined previously in *add_sub_4bit.vhd*), instead of the adder, while in the descriptive part it is only needed to add the assignment of the input related to the subtraction *Sub* to the switch *SW8*, like requested in the assignment provided.

2.2 Functional simulation

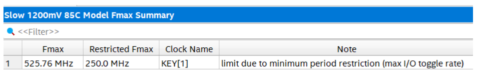
With the testbench *tb_addsub.vhd* it is possible to check the functionalities of the completed design, in particular, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *signed_addsub_4bit.vhd*), and the internal signals for the simulation (*A_t*, *B_t*, *SUM_t*, *Sub_t*, *CR_t*, *OVF_t*, *RSTn_t* and *CLK_t*, all initialized at "0000" or '0', depending if they are vectors or not.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones the simulation is performed through means of a two processes. In the first one the simulation of the subtraction is performed by assigning a 4-bit numerical value to the internal signals *A_t* and *B_t*,'1' to the reset value *RSTn_t* and the same to *Sub_t* internal signal, then the value of *Sub_t* and that of *RSTn_t* are swapped to '0' to simulate the sum (waiting a short amount of time in between each commutation). In the second process it is considered the commutation of the clock *CLK_t*.

2.3 Synthesis

Once the circuit is successfully synthesized by Quartus it is possible to run a timing simulation, as well as the Modelsim functional simulation, and than observe the results.
Moreover, by examining the results reported by the Timing Analyzer it is possible to check that the longest path in the circuit in terms of delay results is 1.902 ns (obtained by subtracting the period of 1 ns to the slack of -0.902), thus the the maximum operating frequency fmax of the circuit is 525.76 MHz.



(a) Modelsim functional simulation

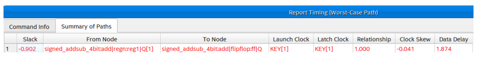


Slow 1200mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	525.76 MHz	250.0 MHz	KEY[1]	limit due to minimum period restriction (max I/O toggle rate)

(a) Maximum frequency



Report Timing (Worst-Case Path)

Command Info | Summary of Paths

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-0.902	signed_addsub_4bit:add|regn:reg1|Q[1]	signed_addsub_4bit:add|flipflop:ff|Q	KEY[1]	KEY[1]	1.000	-0.041	1.874

(b) Longest delay path

Figure 4: Results of Timing Analyzer

3 16-bit RCA, Carry-Bypass Adder and Carry-Select Adder

3.1 16-bit RCA

3.1.1 Design entry

In order to implement a circuit that can perform an addition of 16-bit numbers it is possible to modify the one from exercise one.

- The flipflop and the register are provided in the assignment and defined in *flipflop.vhd* and *regn.vhd* respectively. Only the register is slightly modified: the generic N is now 16 not 4

- The full adder, defined in *full_adder.vhd*, can be approached with a dataflow structure.

- The 16-bit RCA, defined in *rca16bit.vhd*, uses the full adder (defined in *full_adder.vhd*) as a component and the 16-bit vector c as an internal signal, then in the declarative part once the input Cin, defined in the entity, is assigned to the first carry *c(0)*, it is possible to map the inputs and output for each full adder, by means of a generate statement, and finally assign to the output $Cout$ the carry-out propagated *C(16)*.

- The completed circuit is defined in *complete_16bit.vhd*, where in the declarative part of the architecture the flipflop, the register and the 16-bit RCA are defined as components and *Q1, Q2, Sv, cout* and Dv as internal signals, while in the descriptive part the mapping of inputs and outputs of the components to those declared in the entity (*A, B, CR_in, Clock, RSTn, OVF* and S) takes place, as well as the association for the condition of overflow.

3.1.2 Functional simulation

With the testbenches it is possible to check the functionalities of the completed designs.

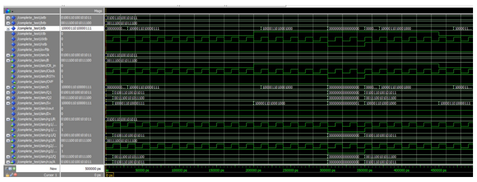
- With the *tb_rca16bit.vhd* it is possible to simulate the behaviour of the RCA: after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *rca16bit.vhd*), and the internal signals for the simulation (*atb*, *btb*, *stb*, *ctb* and *cotb*.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones,

the simulation is performed by means of a process that assigns a 16-bit number to the internal signals *atb* and *btb*, the value '0' to *ctb* and then waits a short amount of time.

- In the *complete_test.vhd* file, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *completed_16bit.vhd*), and the internal signals for the simulation (*atb, btb, stb, ctb, cktb, rstb* and *ovftb*, all initialized at "0000000000000000" or '0', depending if they are vectors or not.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones the simulation is performed through means of two processes. In the first one the simulation of the sum and the behaviour of the reset are performed: once a 16-bit value is assigned to the internal signals *atb* and *btb*, the reset *RSTn_t* value is associated to '1' and the value of *CR_t* to '0', then *CR_t* switches to '1' and lastly the *RSTn_t* to '0' (between each commutation a short amount of time is needed). In the second process it is considered the commutation of the clock *CLK_t*.

3.1.3 Synthesis

Once the circuit is successfully synthesized by Quartus it is possible to run a timing simulation, as well as the Modelsim functional simulation.


(a) Modelsim functional simulation

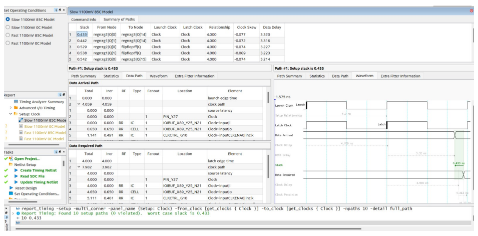

(b) Results of Timing Analyzer

Figure 5: Results of the 16-bit RCA

From the results of the Timing analyzer it can be shown that the longest delay path (relative to a 4 ns period) is 3.567 ns, thus the maximum frequency is 280.35 MHz.

3.2 Carry-Bypass Adder

3.2.1 Design entry

To implement a Carry-bypass adder following the design provided in the assignment some components are needed:

- The flipflop and the register are provided in the assignment and defined in *flipflop.vhd* and *regn.vhd* respectively (with the generic N assigned to 16 for the register).

- The multiplexer and the adder are defined in *mux.vhd* and *adder.vhd*, both with a dataflow approach.

- The 4-bit adder, defined in *adder_4bit.vhd*, uses as component the adder (defined in *adder.vhd*) and as internal signal *c*, so that, through means of a generate statement, for each of the four adders, the inputs and outputs of the component can be mapped to those defined in the entity *A, B, Cin, S* and *Cout*.

- The carry-bypass, also referred to as skip-adder is defined in *skip_adder.vhd*, where in the declarative part of the behavioural architecture the 4-bit adder and the multiplexer are defined as components (from *adder_4bit.vhd* and *mux.vhd* respectively), while *P, pm* and *countm* as internal signals. In the descriptive part it is possible to calculate the propagating bit by firstly making use of a process that loops over the 4 bits of the vector P and associate to it either input (between An or Bn, declared in the entity) of the corresponding bit of the vector and then associate to pm the values resulting from the and-combination of each bit of the vector P, so that pm can be used as selection bit for the multiplexer; in fact it follows the mapping of inputs and outputs of the components to those defined in the entity (*An, Bn, Cin, S* and *Cout*) and the internal signals *countm* and *pm.*

- To implement a 16-bit carry-bypass adder (file *skip16bit.vhd*) four carry-bypass adders are needed: it is only a matter of mapping properly the inputs and outputs of the component (which is the carry-bypass defined in *skip_adder.vhd*) to those defined in the entity (*A, B, Cin, S, Cout*) and the to the internal signal *c.*

- Lastly, the completed circuit is defined in *complete_skip.vhd*, where the 16-bit skip-adder, the flipflop and the register are used as components. In the descriptive part of the behavioural architecture the mapping of the components' inputs and outputs to those declared in the entity (*A, B, CR_in, Clock, RSTn, OVF, S*) and the internal signals (*Q1, Q2, Sv, cout, Dv*) occurs, as well as the association for the condition of overflow.

3.2.2 Functional simulation

With the testbenches it is possible to check the functionalities of the completed designs.

- With the *tb_skip_adder.vhd* it is possible to simulate the behaviour of the carry-bypass: after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *skip_adder.vhd*), and the internal signals for the simulation (*atb*, *btb*, *stb*, *ctb* and *coutb.* Then, in the descriptive part, once the internal signals are properly mapped to the components' ones, the simulation is performed by means of a process that assigns a 4-bit number to the internal signals *atb* and *btb*, the value '1' to *ctb* and then waits a short amount of time, before repeating again for two more times, each time switching the value of *atb, btb* and *ctb.*

- The simulation for the 16-bit carry-bypass is defined in *tb_skip16bit.vhd*, which is the same as the one for the carry-bypass (defined previously in *tb_skip_adder.vhd*), except that the values assigned to the internal signals *atb* and *btb* are 16-bit numbers.

- In the *complete_test.vhd* the completed design is simulated, exactly like for the 16-bit RCA, except for the component defined (*complete_skip* instead of *complete_16bit*)

3.2.3 Synthesis

Once the circuit is successfully synthesized by Quartus it is possible to run a timing simulation, as well as the Modelsim functional simulation.
From the results of the Timing analyzer it can be shown that the longest delay path (relative to a 4 ns period) is 3.962 ns, thus the maximum frequency is 252.397 MHz. Those related to the period of 10 ns would be 4.030 ns for the longest delay path and 247.64 MHz.

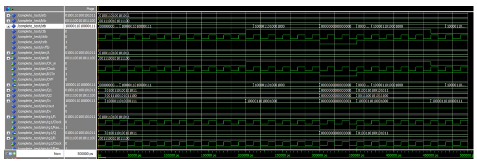
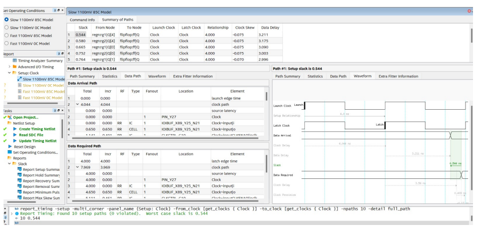
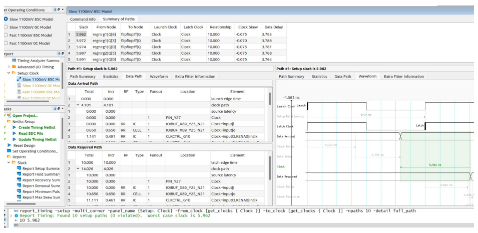


Figure 6: Modelsim functional simulation



(a) T = 4ns



(b) T = 10ns

Figure 7: Results of Timing Analyzer for different periods

3.3 Carry-Select Adder

3.3.1 Design entry

To implement a Carry-select adder following the design provided in the assignment some components are needed:

- The flipflop and the register are provided in the assignment and defined in *flipflop.vhd* and *regn.vhd* respectively (with the generic N changed to 16).

- The multiplexer and the full adder are defined in *mux.vhd* and *full_adder.vhd*, both with a dataflow approach.

- The 4-bit adder, defined in *rca4bit.vhd*, uses as component the adder (defined in *full_adder.vhd*) and as internal signal c, so that, through means of a generate statement, for each of the four full adders, the inputs and outputs of the component can be mapped to those defined in the entity *A, B, Cin, S* and *Cout*.

- The 4-bit carry-select is defined in *select4bit.vhd*, where in the declarative part of the behavioural architecture the 4-bit RCA and the multiplexer are defined as components (from *rca4bit.vhd* and *mux.vhd* respectively), while *sum1, sum2, Cout1* and *Count2* as internal signals. In the descriptive part occurs the mapping of inputs and outputs of the components to those defined in the entity (*A, B, Cin, S, Cout*) and the internal signals, in particular the carry propagation for the two RCAs is assigned (it is equal to '1' for one RCA and equal to '0' for the other, since they need to work in parallel thus considering each possibility); it also takes place the association of the sum output *S*, depending on the value of the carry propagation.

- To implement a 16-bit carry-select adder (file *select16bit.vhd*) four 4-bit carry-select adders are needed: it is only a matter of mapping properly the inputs and outputs of the component (which is the carry-select defined in *select4bit.vhd*) to those defined in the entity (*A, B, Cin, S, Cout*) and the to the internal signal *c*.

- Lastly, the completed circuit is defined in *complete_select.vhd*, which is the same as the completed circuit for the carry-bypass, except for the component of the carry-select (*select16bit.vhd*) instead of the carry-bypass.

3.3.2 Functional simulation

With the testbenches it is possible to check the functionalities of the completed designs.

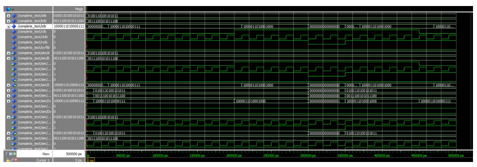
- With the *tb_select4bit.vhd* it is possible to simulate the behaviour of the carry-select, exactly like the simulation for the carry-bypass. After the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *skip_adder.vhd*), and the internal signals for the simulation (*atb*, *btb*, *stb*, *ctb* and *coutb*.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones, the simulation is performed by means of a process that assigns a 4-bit number to the internal signals *atb* and *btb*, the value '1' to *ctb* and then waits a short amount of time, before repeating again for two more times, each switching the value of *atb, btb* and *ctb*.

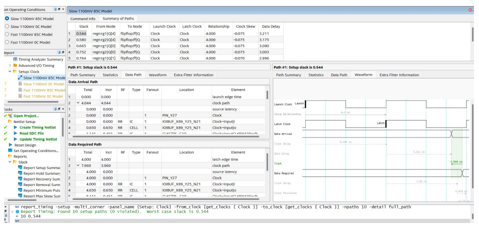
- The simulation for the 16-bit carry-bypass is defined in *tb_select16bit.tb*, which works like the one for the 4-bit carry-select (defined in *tb_select4bit.vhd*), except that the values assigned to the internal signals *atb* and *btb* are 16-bit numbers.

- In the *tb_complete_select.vhd* the completed design is simulated, exactly like for the 16-bit RCA, and the carry-baypass except for the component defined (*complete_select*.

3.3.3 Synthesis

Once the circuit is successfully synthesized by Quartus it is possible to run a timing simulation, as well as the Modelsim functional simulation.


(a) Modelsim functional simulation



Slow 1100mV 85C Model

Summary of Paths

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	0.544	regnzg1|Q[4]	flipflopdf|Q	Clock	Clock	4.000	-0.075	3.211
2	0.580	regnzg2|Q[4]	flipflopdf|Q	Clock	Clock	4.000	-0.075	3.175
3	0.665	regnzg1|Q[0]	flipflopdf|Q	Clock	Clock	4.000	-0.075	3.090
4	0.752	regnzg2|Q[0]	flipflopdf|Q	Clock	Clock	4.000	-0.075	3.003
5	0.764	regnzg1|Q[1]	flipflopdf|Q	Clock	Clock	4.000	-0.070	2.996

Path #1: Setup slack is 0.544

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	0.000	0.000					launch edge time
2	4.044	4.044					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_Y27	Clock
3	0.000	0.000	RR	IC	1	IOIBUF_X89_Y25_N21	Clock~input|i
4	0.650	0.650	RR	CELL	1	IOIBUF_X89_Y25_N21	Clock~input|o

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	4.000	4.000					latch edge time
2	7.969	3.969					clock path
1	4.000	0.000					source latency
2	4.000	0.000			1	PIN_Y27	Clock
3	4.000	0.000	RR	IC	1	IOIBUF_X89_Y25_N21	Clock~input|i
4	4.650	0.650	RR	CELL	1	IOIBUF_X89_Y25_N21	Clock~input|o

```
report_timing -setup -multi_corner -panel_name {Setup: Clock} -from_clock [get_clocks { Clock }] -to_clock [get_clocks { Clock }] -npaths 10 -detail full_path
> Report Timing: Found 10 setup paths (0 violated).  Worst case slack is 0.544
← 10 0.544
```

(b) Results of Timing Analyzer

Figure 8: Results of the carry-select

From the results of the Timing analyzer it can be shown that the longest delay path (relative to a 4 ns period) is 3.456 ns, thus the maximum frequency is 289.35 MHz.

3.4 Comparison of the timing results

Theoretically we would expect that the Carry-Select and the Carry-Bypass were faster than the RCA, that is because their delay, differently from the RCA, is proportional to the root of the number of bits of the adder; in particular the Carry-Select should be faster than the Carry-Bypass. However, from the simulation it can be argued that the Carry-Bypass results to be the slowest.

4 Multiplier

4.1 Design entry

In order to implement the multiplier reported in the figure of the assignment it is possible to reuse the components defined in *f_adder.vhd, mux_2_1_1bit.vhd* and *hex_decoder.vhd* to specify the full adders, the multiplexers and the decoder respectively. For the RCA, since it is not required to manage signed number, it is possible to re-propose the *rca_4bit.vhd* file and change the declaration of the signed inputs and output, to *STD_LOGIC_VECTOR* (the file is renamed as *rca_4bit_mult.vhd*).

The multiplier is then outlined in *multiplier.vhd* where the RCA is defined as a component (from *rca_4bit_mult.vhd*) and *co_temp1, co_temp2, co_temp3, a_temp1, b_temp1, s_temp1, a_temp2, b_temp2, s_temp2, a_temp3, b_temp3, s_temp3* as internal signal in the declarative part of the structural architecture. In the descriptive part first it takes place the mapping of the inputs and outputs of the RCA to the internal signals, which are then related to the suitable conditions for the RCAs in order to finally associate the appropriate internal signals and inputs are associated to the output of the multiplier.

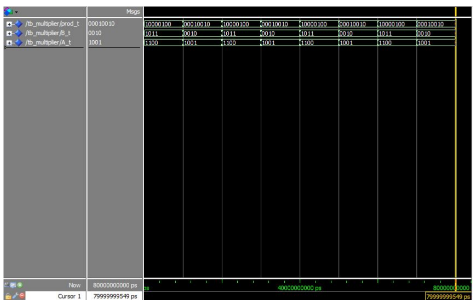
Finally in *multiplier_map.vhd* the multiplier an the decoder are defined as components and *Adec, Bdec, Pdec0, Pdec1, Pdec* as internal signals in order to map the inputs and outputs of the multiplier and the decoder to those associated to the DE1 board (previously defined in the entity) and the internal signals; in order to be able to use switches SW3-0 to represent the number B and switches SW7-4 to represent A, to display the hexadecimal values of A and B on the 7-segment displays HEX0 and HEX1, respectively and the result of the multiplication the the 7-segment displays HEX1 and HEX3.

4.2 Functional simulation

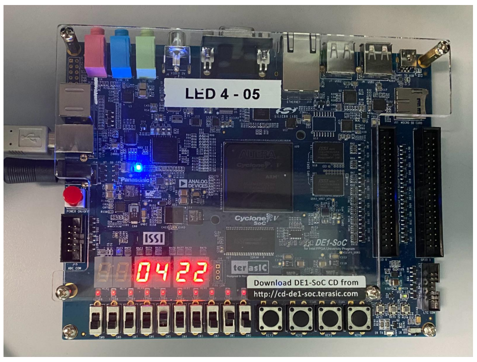
With the testbench, defined in *tb_multiplier* it is possible to check the functionality of the multiplier. In particular, after the typical empty entity, in the declarative part of the behavioural architecture is defined the component (specified in *multiplier.vhd*), and the internal signals for the simulation (*A_t*, *B_t* and *prod_t*.
Then, in the descriptive part, once the internal signals are properly mapped to the components' ones, the simulation is performed by assigning a number in binary form to each *A_t* and *B_t* twice (in the first case the numbers from the assignment were chosen) and waiting a short amount of time in between.

4.3 Synthesis

Once the circuit is successfully synthesized by Quartus it is possible to run a timing simulation, as well as the Modelsim functional simulation, and then observe the results.



(a) Modelsim functional simulation



(b) Results of a multiplication

Figure 9: Multiplier's results