[Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics (DS181)]
[7 Series FPGAs Data Sheet: Overview (DS180)]
[Nexys4 DDR™ FPGA Board Reference Manual 1 Overview]
[7 Series Product Tables and Product Selection Guide]

Table 4: **Artix-7 FPGA Feature Summary by Device**

Device	Logic Cells	Configurable Logic Blocks (CLBs)		DSP48E1 Slices[2]	Block RAM Blocks[3]			CMTs[4]	PCIe[5]	GTPs	XADC Blocks	Total I/O Banks[6]	Max User I/O[7]
		Slices[1]	Max Distributed RAM (Kb)		18 Kb	36 Kb	Max (Kb)						
XC7A12T	12,800	2,000	171	40	40	20	720	3	1	2	1	3	150
XC7A15T	16,640	2,600	200	45	50	25	900	5	1	4	1	5	250
XC7A25T	23,360	3,650	313	80	90	45	1,620	3	1	4	1	3	150
XC7A35T	33,280	5,200	400	90	100	50	1,800	5	1	4	1	5	250
XC7A50T	52,160	8,150	600	120	150	75	2,700	5	1	4	1	5	250
XC7A75T	75,520	11,800	892	180	210	105	3,780	6	1	8	1	6	300
XC7A100T	101,440	15,850	1,188	240	270	135	4,860	6	1	8	1	6	300
XC7A200T	215,360	33,650	2,888	740	730	365	13,140	10	1	16	1	10	500

Table 1: **7 Series Families Comparison**

Max. Capability	Spartan-7	Artix-7	Kintex-7	Virtex-7
Logic Cells	102K	215K	478K	1,955K
Block RAM[1]	4.2 Mb	13 Mb	34 Mb	68 Mb
DSP Slices	160	740	1,920	3,600
DSP Performance[2]	176 GMAC/s	929 GMAC/s	2,845 GMAC/s	5,335 GMAC/s
MicroBlaze CPU[3]	260 DMIPs	303 DMIPs	438 DMIPs	441 DMIPs
Transceivers	–	16	32	96
Transceiver Speed	–	6.6 Gb/s	12.5 Gb/s	28.05 Gb/s
Serial Bandwidth	–	211 Gb/s	800 Gb/s	2,784 Gb/s
PCIe Interface	–	x4 Gen2	x8 Gen2	x8 Gen3
Memory Interface	800 Mb/s	1,066 Mb/s	1,866 Mb/s	1,866 Mb/s
I/O Pins	400	500	500	1,200
I/O Voltage	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V	1.2V–3.3V
Package Options	Low-Cost, Wire-Bond	Low-Cost, Wire-Bond, Bare-Die Flip-Chip	Bare-Die Flip-Chip and High-Performance Flip-Chip	Highest Performance Flip-Chip

1. Look up the data sheet for the device we are using in the laboratory and then answer the following questions:
   1.1. 15,850/2 = 7925 CLBs

   1.2. 300 (datasheet) or 210 (Digikey Product Listing)

   1.3. 1.2V, 1.35V, 1.5V, 1.8V, 2.5V, and 3.3V. This is configured by selecting an I/O Standard when you go to the I/O Planning stage.
   However, 1.35V is mentioned in the datasheet, but it only shows up there, not in the I/O planning or any of the other tables.

   1.4. FGG484 or FGG676 which are compatible with FBG484 and FBG676 respectively. (Or CSG324 or FTG256)

   1.5. A speed grade is a relative metric of performance for an FPGA. The higher the number the faster the device is. (Artix®-7 FPGAs are available in -3, -2, -1, -1LI, and -2L speed grades, with -3 having the highest performance) ([https://forums.xilinx.com/t5/CPLDs-Archived/Speed-Grade/td-p/3052#:~:text

=There%20is%20no%20consistent%20definition,in%20%3C%3D%20out).)

2. With reference to the CLB:
   2.1. Combinatorial logic is typically implemented as look up tables.

   2.2. The main purpose of the flip-flops/latches is to synchronise all signals.

   2.3. A carry chain is a line through each LUT that propagates the carry bit through it. This is used to speed up arithmetic and counters.

   2.4. Slice M has LUTs that can be used for memory applications as well as logic ones whereas Slice L has LUTs that can only be used for logic applications.

   2.5. Connected by a programmable interconnect fabric

3. With regard to FPGA design:
   3.1. The Why and How of Pipelining in FPGAs - Technical Articles
   Pipelining is a technique that is used to make everything work in parallel. To make the processes parallel, you stick a D flip flop between each stage to clock each operation, input and output. This synchronises each operation however adds a latency in the output as data only propagates on the clock edge. The beneficial result is that after all the data has propagated through for the first time, the next wave of data is one clock cycle away from the output and each subsequent clock cycle a new output is produced.

   3.2. Setup time is the minimum amount of time before a clock edge that a data signal must be stable for it to be latched correctly. Hold time is the minimum amount of time after a clock edge that it must be stable. The hold time is required because the data is not latched instantaneously after a clock edge.

   3.3. Metastability is a state that the output of a clocked device can be in. This state is when the input of a clocked device, in this case a flip-flop, doesn't meet setup and hold timing requirements. This results in the data not successfully being passed through the flip-flop and result in the output being neither the new or old data and the flip-flop won't settle on a value until after the clock edge hold time. There is no guarantee that the value that is settled on is the new value that you wanted passed through. This can occur when two clocks are used as this will cause the clock edges to eventually line up causing a flip-flop on one clock domain to switch and then very quickly after, the next flip-flop on another domain could flip. This can also occur if the input data is asynchronous as say a switch, could change state from an external user which happens to be inside the setup and hold windows causing the same issue. It can also happen should the clock signal arrive at different times between devices which causes the same issue as all of a sudden the timing requirements can't be met. A final way that metastability can happen is that the signal has to travel through many devices which accumulates

propagation delay. This propagation delay can get to the point where it causes the signal to change at the same time as a clock edge. The main causes of metastability are accumulated propagation delay and different clock domains (whether that be from separate clocks or an asynchronous input).

3.4. Dealing with metastability

The way the metastability is typically handled is by sticking two or more series flip flops on the input or output of an offending device. This ensures that any out of sync data eventually becomes synchronised so long as the flip-flops are on the same clock line of the device that requires a stable input.

3.5. The main difference between a Mealy and a Moore state machine is how the output state is determined. A Mealy state machine has its output determined by the current state the machine is in and the current input. A Moore state machine has its output determined solely by the current state the machine is in.

4. VHDL Code for a compare and add circuit

4.1. Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

entity ass1_q4 is
   Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
          b : in STD_LOGIC_VECTOR (3 downto 0);
         comp : out STD_LOGIC;
        sum : out STD_LOGIC_VECTOR (4 downto 0));
end ass1_q4;

architecture Behavioral of ass1_q4 is

Begin
   comp <= '1' when a = b else
           '0';
   sum <= std_logic_vector(unsigned('0' & a) + unsigned('0' & b));
end Behavioral;
```

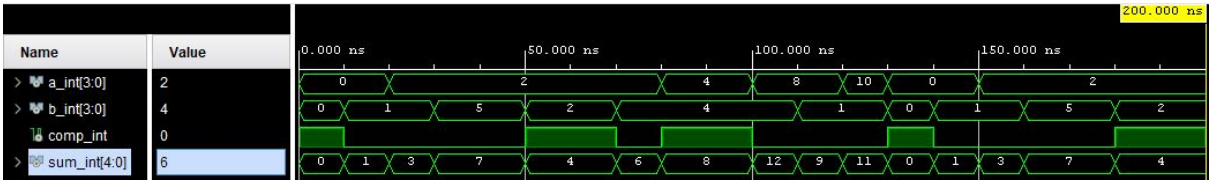
4.2. Testbench Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library UNISIM;
```

```
use UNISIM.VComponents.all;

entity ass1_q4_tb is
end ass1_q4_tb;

architecture Behavioral of ass1_q4_tb is

signal a_int : STD_LOGIC_VECTOR (3 downto 0);
signal b_int : STD_LOGIC_VECTOR (3 downto 0);
signal comp_int : STD_LOGIC;
signal sum_int : STD_LOGIC_VECTOR (4 downto 0);

component ass1_q4 is
    Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
           b : in STD_LOGIC_VECTOR (3 downto 0);
           comp : out STD_LOGIC;
           sum : out STD_LOGIC_VECTOR (4 downto 0));
end component;

begin

add_comp : ass1_q4 port map (
    a => a_int,
    b => b_int,
    comp => comp_int,
    sum => sum_int
);


testbench : process
begin
    a_int <= "0000";
    b_int <= "0000";
    wait for 10ns;
    b_int <= "0001";
    wait for 10ns;
    a_int <= "0010";
    wait for 10ns;
    b_int <= "0101";
    wait for 10ns;
    a_int <= "0010";
    wait for 10ns;
    b_int <= "0010";
    wait for 10ns;
    a_int <= "0010";
    wait for 10ns;
```

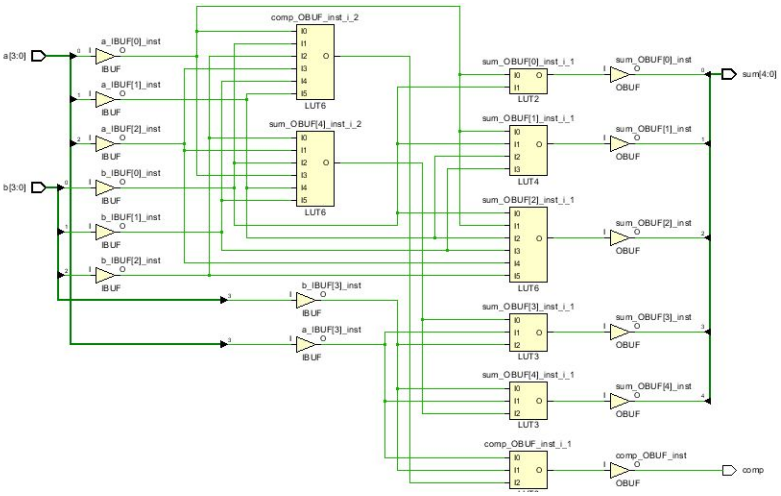
```
    b_int <= "0100";
    wait for 10ns;
    a_int <= "0100";
    wait for 10ns;
    b_int <= "0100";
    wait for 10ns;
    a_int <= "1000";
    wait for 10ns;
    b_int <= "0001";
    wait for 10ns;
    a_int <= "1010";
    wait for 10ns;
    b_int <= "0010";
end process;

end Behavioral;
```

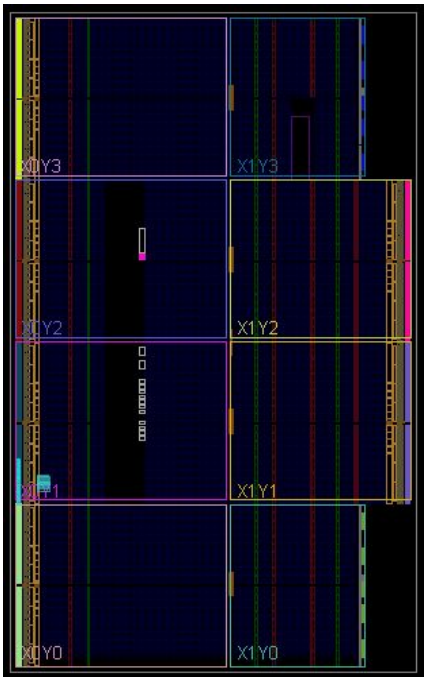
4.3. **Testbench Results**



5. Development Environment

5.1. Schematic Generation of Synthesis and Routed Designs





Synthesis and routed designs can generate a schematic like the one above that helps us figure out what the FPGA is going to implement to perform its task. The above left image is of the synthesized design, synthesising the design is the process that vivado undertakes where it decides what will be used in the design. From this we can see that the FPGA will have many lookup tables to perform the compare and add task. This allows us to evaluate how fast the design will be as we can see how many tables data has to go through before it propagates all the way through. The implemented design provides an accurate depiction of what segments of the FPGA will be used when the design is properly implemented. This process is when vivado attempts to map all the LUTs and signals to where they would be on the physical chip. From this we can also gauge how fast it will be based on the physical distance it has to travel and by performing accurate timing analysis. In addition to this it provides useful metrics on how much of the FPGA is utilised and the amount of noise that will be present. The benefits of these is that we can figure out how to best optimise the program, particularly as well with the use of the I/O planning both provide as we can select pins that are close together.

5.2. Testbench Simulation

Testbenches are written so that they provide input and read the output of your modules. This is done so that we can verify the functionality of the program. It's very useful to us for also identifying states in the program where a variable is undefined. We can easily simulate a clock as well which allows us to

perform another check on the timing of the signals.

5.3. Timing Analyser

When running a timing analysis, vivado will time different routes in the program to determine if they met the required setup and hold time restrictions. It reports back whether they don't based on the current constraints file and the user is able to easily see this information and then make changes accordingly to fix the timing problems. The user is able to inspect all the timing issues that appear by clicking on the issue which will then take them to the component in the schematic that is causing these issues which makes the process a lot smoother for a user.

5.4. ILA Core

The ILA is used to perform hardware debugging operations. It can be added to a project by using the IP catalog which makes it very simple. The IP Catalog entry is very flexible in what you can implement into your design. This then gets instantiated in the VHDL module that you want to include it in. You can also add an ILA through the synthesised design, just mark which pins you want to set up debug for and make sure the clock domains for the debugger and the block are the same. This can then be uploaded to the FPGA and once that is done, in Vivado you can add the created probes to triggers, and using a terminal and a serial link, you can send data to the FPGA (Or can use on board pins) to provide input to the system and vivado will log the output. The benefit of something like this is that it allows a designer to functionally test their program on the physical hardware, which is the best place to test it, because it will provide the most accurate results on how it will actually perform.