EEDG/CE 6302

Microprocessors and Embedded Systems

Electrical and Computer Engineering

Erik Jonsson School of Engineering and Computer Science

The University of Texas at Dallas

Dr. Tooraj Nikoubin

Project 1 (Part 3): Instruction Decoder and Constant Unit

Submitted By:

Muripa Uppaluri (MXU220008)

Chandanam Sai Nived (SXC210186)

Table of Contents

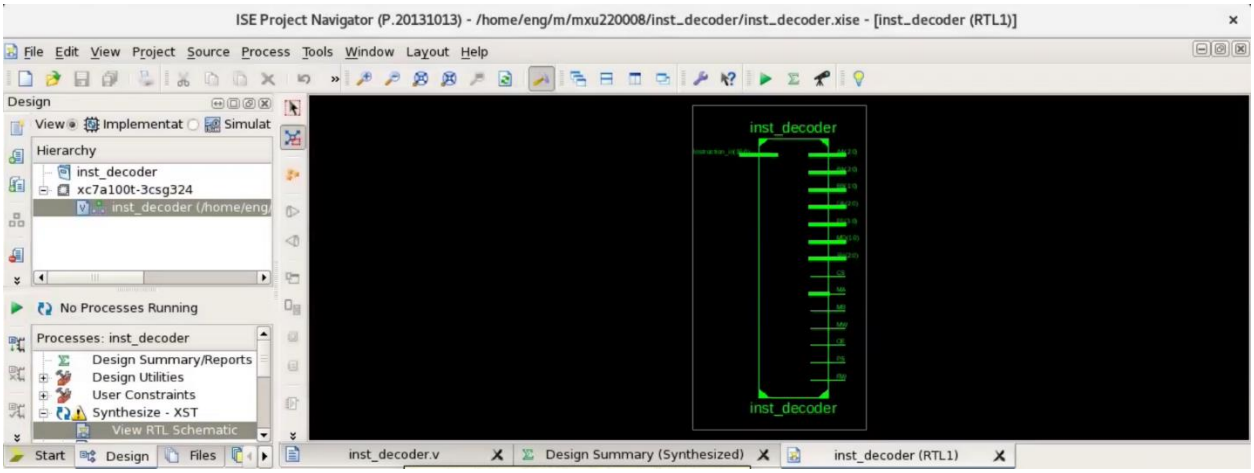
In this third part of first lab project, we've designed an instruction decoder and a constant unit. We will be explaining about the implementation of each in the following sections.

Instruction Decoder:

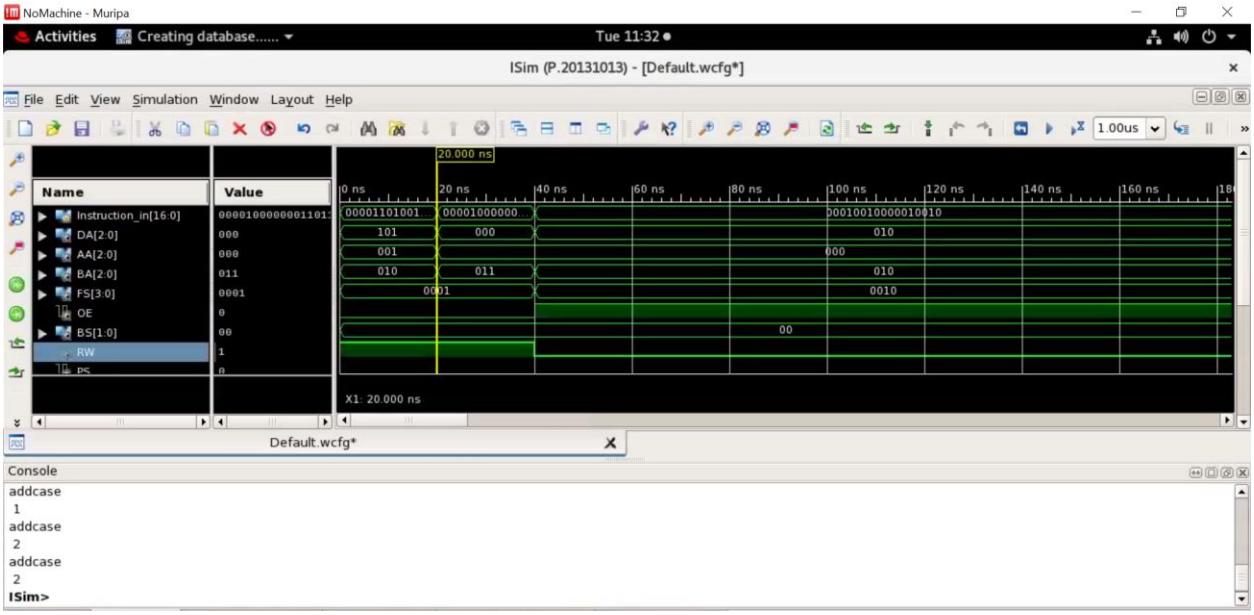
The instruction decoder decodes the instruction received from instruction register and generates the necessary control signals to other components of MCU. Based on the type of instruction received, the instruction decoder asserts the conditional outputs that perform the required operation. We have 17 bit instructions as input to the decoder. The implementation of the design is simple, the last 5 bits of the instruction is decoded as the opcode and based on opcode, the operation is performed and required control signals are asserted as given in table.

We've created a simple testbench to verify this instruction decoder. In this testbench module, we've declared all **the input ports as reg datatype** so that they can hold the changing values of stimulus and the output port as wire data type. The device under test is instantiated using **port-map instantiation**. We are assigning random values to the instruction and checking the outputs flags assertion.

Schematic view of the Instruction Decoder –

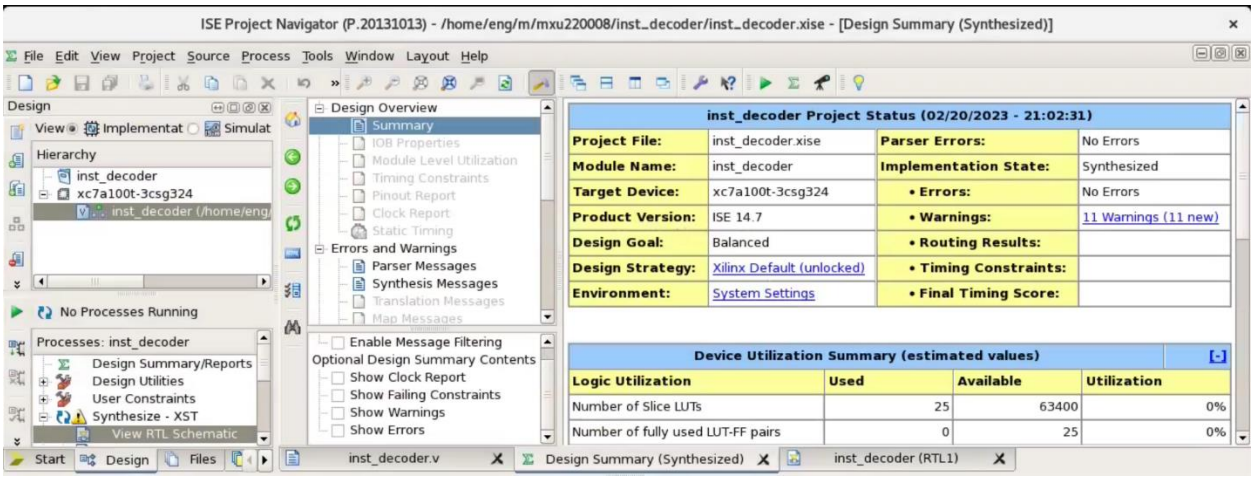


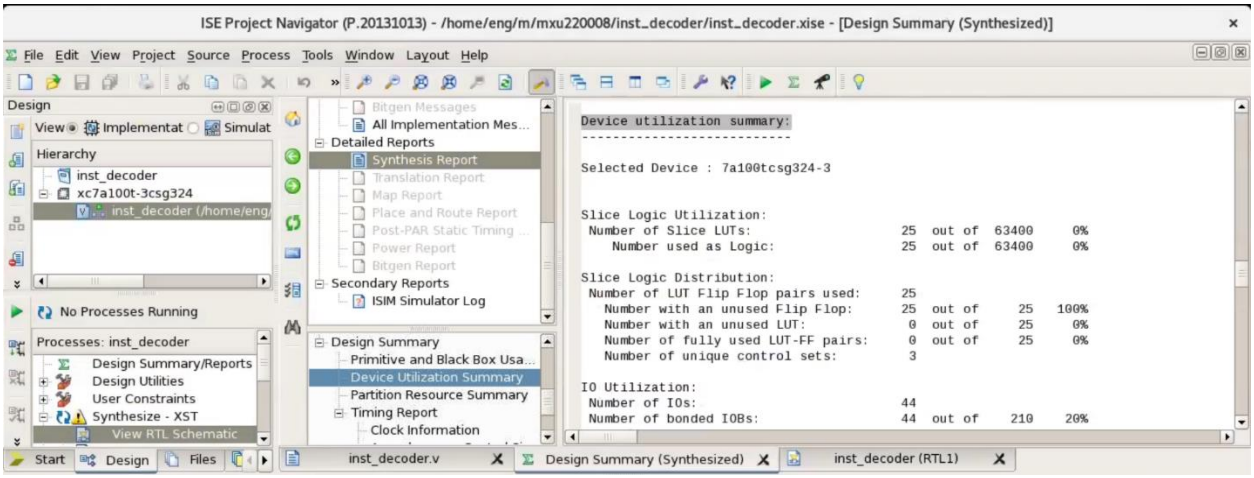
Instruction Decoder Waveform –



Design Summary Report:

Since this is pure combinational design, so we don't have any FFs.





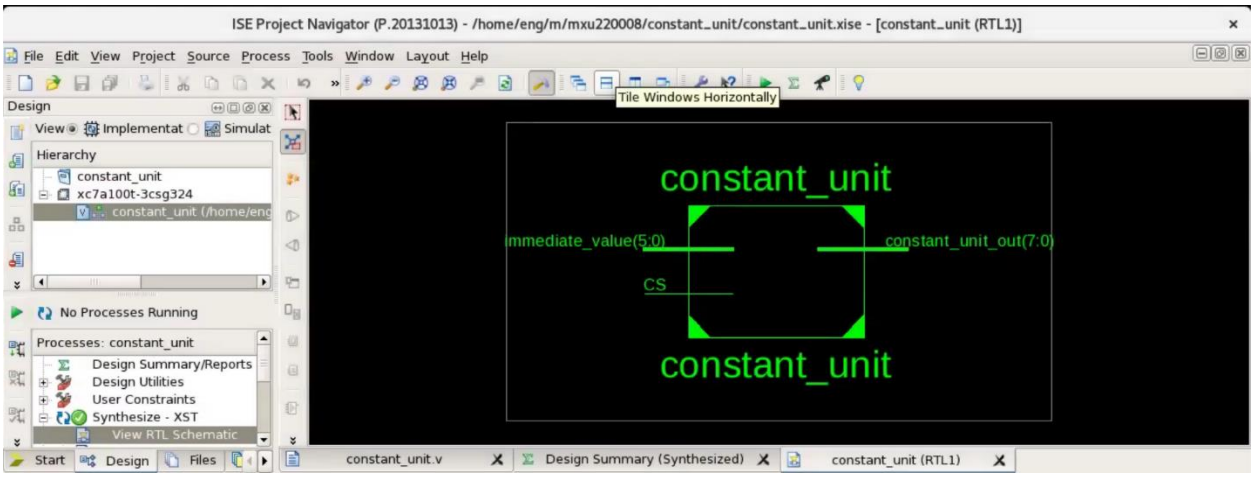
Constant Unit Implementation:

In this design, the constant unit takes 6 bit immediate value and 1 bit CS value as input and gives out 8 bit updated immediate value with 2 extra bits. If CS==’0’, zero fills to the left of the immediate value. If CS==’1’, the sign extension should be performed by the Constant Unit on the immediate value. We’ve used always@(*) block to trigger the constant unit and if-else constructs to code the necessary conditions.

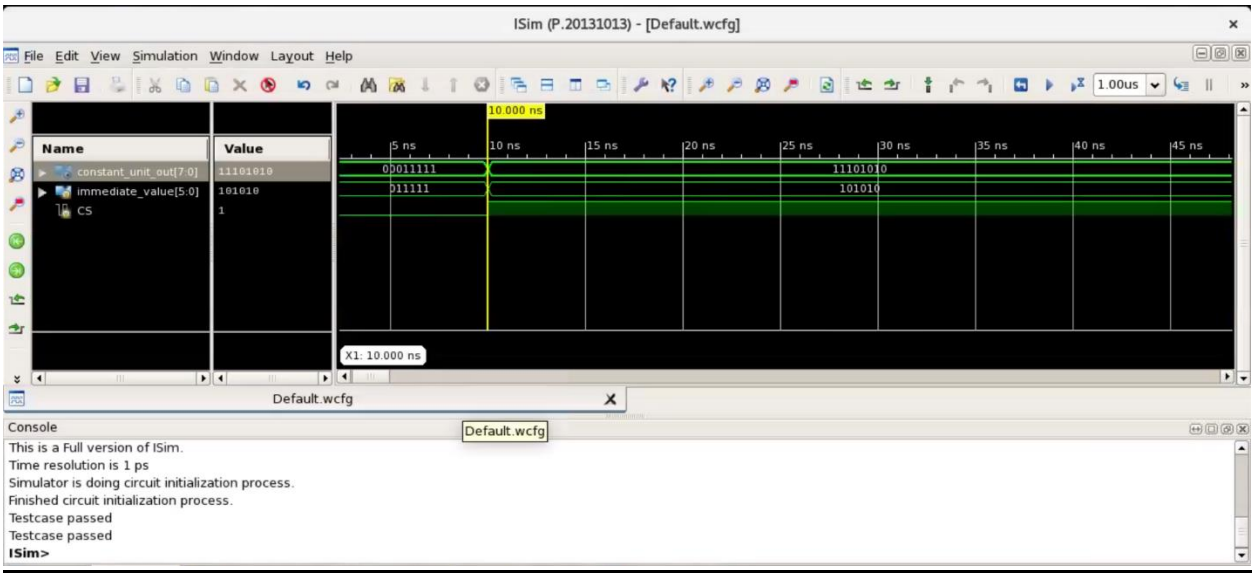
Constant Unit Testbench –

We’ve created a simple testbench to verify this constant unit. In this testbench module, we’ve declared all the input ports as reg datatype so that they can hold the changing values of stimulus and the output port as wire data type. The device under test is instantiated using port-map instantiation. The testbench performs simple checking of constant unit. Firstly, we drive the CS to 0 and check the addition of 0 bits to the immediate value driven. The testbench displays whether this addition of 2 0 bits is successful or not. Secondly, we drive CS to 1 and check the constant unit for signed extension. The result is displayed using $display statement.

Schematic View of Constant Unit –

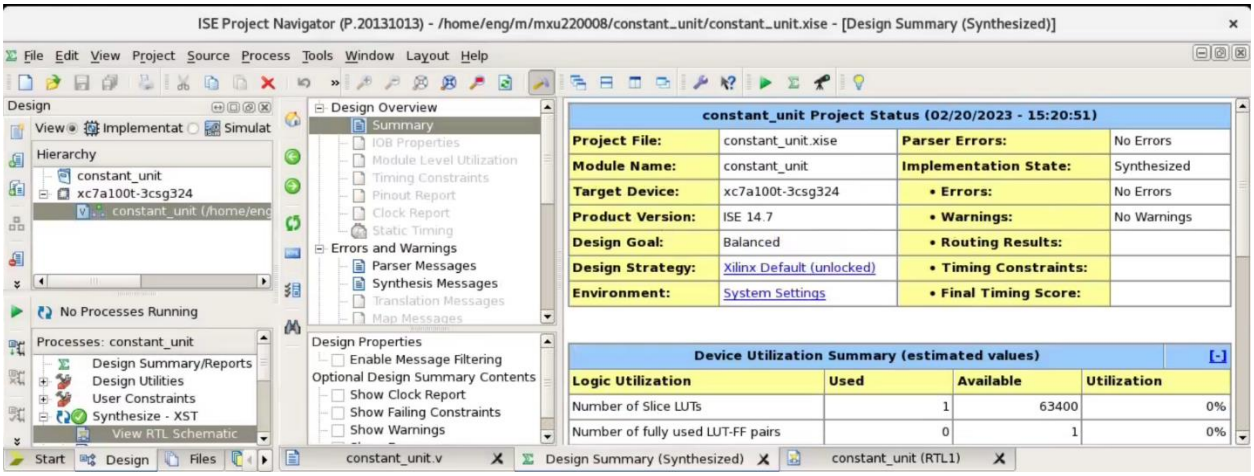


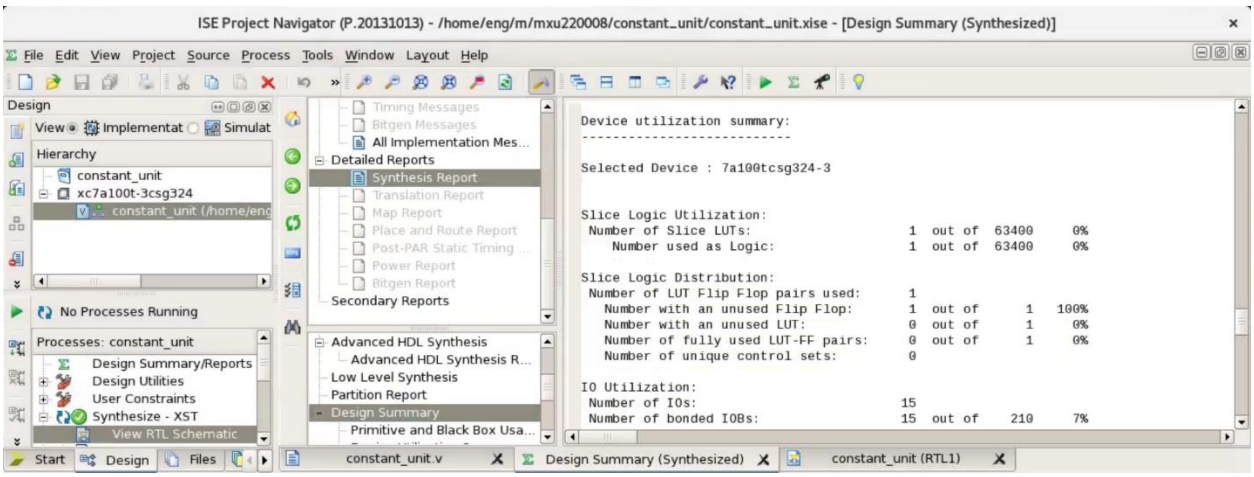
Constant Unit waveform –



Constant Unit Design Summary Report:

No FF are used in this design since this is pure combinational circuit.





Lab Handout Questions

1. Sign extension is a process of adding extra bits to a number while preserving its sign. For example, if a 8bit number needs to be extended to 16 bits, then we need to add another 8 bits more based on the MSB. The processors needs to fill the additional 8 bits with 0s or with 1s to preserve the positive or negative sign of the number.

2. Given that ADN has two inputs, inverts one of the inputs and adds to the other. This comes into the category of arithmetic instructions. Arithmetic instructions are used to perform mathematical operations such as ADD, SUB, MUL, etc.

3. Incase of AND, the control signals could be –

	RW	MD	BS	PS	MW	FS	MB	MA	CS	OE
ADN	1	00	00	X	0	ARB	0	0	X	0