Lab 6: RISCV Fault Injection in Matrix Multiplication Unit

In this lab you will perform fault injections to the matrix multiplication unit of the RISCV Neorv32 design,with the use of the FREtZ.

1)Vivado:

1. Find the matrix multiplication unit of your RISCV processor, in your Vivado project
2. Ensure that the logic location (.ll) and masked file (.msk) will be created during bitstream during the bistream generation, from the settings.
3. In the XDC use the following lines in order to create the .bit, .ebd, .ebc files during synthesis/implementation

```
# Necessary contstaints for conf. memory fault injection
# set_property BITSTREAM.SEU.ESSENTIALBITS yes [current_design]
set_property BITSTREAM.GENERAL.PERFRAMECRC YES [current_design]
set_property BITSTREAM.CONFIG.INITSIGNALSERROR DISABLE [current_design]
```

4. Create a pblock in XDC which will contain only the matrix multiplication module. Use the following SLICES.

```
# Add muldiv unit in pblock1
# Resize pblock1 to provide adequate resources to implement muldiv unit
create_pblock pblock1
add_cells_to_pblock [get_pblocks pblock1] [get_cells -quiet [list PATH_TO_MULT_MODULE]
resize_pblock [get_pblocks pblock1] -add {SLICE_X22Y50:SLICE_X31Y99}
resize_pblock [get_pblocks pblock1] -add {RAMB18_X1Y20:RAMB18_X1Y39}
resize_pblock [get_pblocks pblock1] -add {RAMB36_X1Y10:RAMB36_X1Y19}
# # Isolated Region/Pblock-A physical area to implement logic.
set_property HD.ISOLATED true [get_cells PATH_TO_MULT_MODULE]


```

5. Create Bitstream

2) Creation of software mutrix multiplication application

You need to create a sofware application in C which will perform matrix multiplications between an array.

1. Use the template of Hello world to build your application
2. Define a constant ARRAY_SIZE (3 *1024 /sizeof(int))
3. Fill an array of size ARRAY SIZE
   a. For i in ARRAY_SIZE
      i. array[i]=i
4. Calculate the total sum of this array and print it through UART
5. Implement the following code:
   a. result=0
   b. repetitions=1000
   c. for k<10, k++
      i. for i<repettitions,i++
         1. for j <repetitions,j++
            a. results+=(i %255)*(j*255)
      ii. print result

6. After writing this code make this application the default application upon boot.

3)FREtZ Fault Injections

1. Find the app_frames.txt files from Eclass which contains the addresses of the frames of the Pblock1
2. Create a project with FREtZ and copy the essential files in the folder (bit, ebd, ebc, ll, msk)
3. Change the UserApplication.py file
   a. In class UserApplication Define the following constants:
      i.
```
#: The serial port to be used by :class:SerialPort for the test results
__SERIAL_PORT_TEST_REPORT = '/dev/ttyUSB0' # OR COM9 (find from device manager for windows)

#: The baud rate for :class:SerialPort
__SERIAL_PORT_BAUDRATE = 19200
#: Timeout (in milliseconds) for a test to be completed
__TEST_COMPLETED_TIMEOUT_MSEC = 5000
```
   b. In class UserApplication add two functions
      i.
```
  def __TimerTestCompletedTimeout(self):
    self._testResponseError = True
    if self._eventLoop:
        self._eventLoop.quit()
```
      ii.
```
  @Slot(bytes)
   def TestResultReceived(self, dataBytes : bytes):
       """Callback function for the serial data received from the FPGA when test results are reported

       :param dataBytes: The received bytes
       :type dataBytes: bytes
       """
       try:
           value = dataBytes.decode('utf-8')
           print(f'{value}')
           f=open(self._project.Path+"/results.txt","a+")
           f.write(value)
           f.close()
           self._eventLoop.quit()

       except Exception as e:
           Log.PrintException(f'UserApplication.TestResultReceived: {str(e)}')
```
   c. In the RUN() function
      i. Open the serialPort of the UART module
         1.
```
#Open the serial port for the rests results
self._serialPortTestReport = SerialPort(UserApplication.__SERIAL_PORT_TEST_REPORT, UserApplication.__S
self._serialPortTestReport.Open(self.TestResultReceived)
```
      ii. Define the command manager
         1.
```
self._commandManager = CommandManager(self._project.FpgaDevice, self._project.IpAddress, self._project
self._execute = True
```
      iii. Read and print ID of the board
         1.
```
# Read S/N ans SW version - use allways this mwthod to inform the main application about SW version ar
boardInfo = self._commandManager.ReadId()
serialNumberHex = '{0:0{1}X}'.format(boardInfo.SerialNumber, 8)
```

```
self._uiSignals.RemoteDeviceSwIdReceived.emit(f'{serialNumberHex} : {boardInfo.Version}')
```

iv. Download the bitstream of the FPGA

```
status = self._commandManager.ConfigureDevice(self._project.BitFilePath)
```

v. load the frames of the app_frames.txt to an array and for every frame create 3232 (number of total bits of each frame) elements.

vi. In this way you have created the complete fault list of your Design Under Test (DUT) since you have every frame and every bit of it.

1.
```
        #Load from the external app_frames.txt file all the frames from pblock
        app_frames=[]
        all_fault_list=[]
        f=open(self._project.Path+"/app_frames.txt","r")
        for line in f:
            tmp=line.replace("\\n","")
            frm=int(tmp, 16)
            #make the overall fault list (every frame has 3232 bits)
            for i in range(0,3232):
                value=(frm , i)
                all_fault_list.append(value)
            app_frames.append(frm)
        f.close()
```
2. Note each element of app_frames has two items
   a. frameaddress
   b. bitindex

vii. Define a constant of TOTAL_BITS (the total number of fault injections to perform)

viii. Select randomly n frames to inject from all frames (where n=TOTAL_BITS)

ix. Create a fault injection loop where
1. define the frame and bitindex to inject
2. Print the number of iteration and the frameaddress, bitindex that you will inject.
3. read the frame (frames = self._commandManager.ReadFrame(frameAddress, 1))
4. bitflip the bitindex (frames[0].BitFlip(bitIndex))
5. Write frame back (status = self._commandManager.WriteFrame(frames))
6. Setup a timer
   a.
   ```
   self._testResponseError = False
   self._eventLoop = QEventLoop()
   self._timerTestCompleted = QTimer()
   self._timerTestCompleted.setSingleShot(True)
   self._timerTestCompleted.timeout.connect(self.__TimerTestCompletedTimeout)
   self._timerTestCompleted.start(UserApplication.__TEST_COMPLETED_TIMEOUT_MSEC)
   ```
7. Send through serial the character “X”
8. Start the timer
9. Check if timer is active
   a. if TRUE -->stop the timer
10. Check if self._testResponseError is True
    a. If True, it means that your system crashed you need to download the bitstream again
    b. If false, it means that you received the results and the system is still running. Now, you need to correct the fault that you injected in order to go to the next fault injection.
       i. Read frame again
       ii. Bitflip again

        iii. Write frame back

4. After the completion of your script, run the fault injection experiments
5. Now you need to analyse the results.txt file in order to find the AVF of your design

TIP: Debug of your Project

1. Add the following lines:
    a.
    ```
    pydevd.connected = True
    pydevd.settrace(suspend=False)
    ```
2. Add breakpoints
3. And run your code from RUN->Run with debug