Iowa State University

R2U2 Demo (Jan.2018) Documentation

January 1, 2018

Quick Demo Setup

System input: combined sensor data in binary format
System output: verdict and time stamp for each future time monitor assembly instruction

1 Brief Setup Process

This section talks about how to generate required files and steps to run the demo.
Note: 1) The text mark with means the command typed in terminal.
2) The text marked in green are the files that require manually modify.

1.1 Preprocessing

1. Generate checker associated file
   (a) Modify inputs.py (section 2.1) to specify the input data format and name, then run `python inputs.py`
   (b) Generate at_checkers.vhd and log_input_pkg.vhd by running `python transformer.py`
   (c) Write atomic assertion configuration in input.ast (section 2.2), run `python assert_convert.py` to convert the assertion into binary configuration file named res.atc.
2. Generate binary instruction assembly code and its interval file (.imem and .int)
   (a) write assembly code in casestudy.ftasm, run `sh convert.sh` in folder
3. Generate UART byte data
   (a) Modify parameters associated with data byte size in python gen_uart_data.py (section 2.3). These parameters should be the same as R2U2_pkg.vhd. Run command `python gen_uart_data.py`

1.2 Hardware Connection

1. FPGA Board Connection
   (a) Connect the JTAG with PC for downloading bitfile. See Figure 1.
2. UART Connection
   (a) Connect the UART module with PC, connect UART module PIN **DIN** with Zedboard **JP4**, PIN **DOUT** with Zedboard **JP3**. See Figure 2.
3. Reset button is located in Zedboard **BTNC(P16)**.

1.3 Run

1. Download bitfile into FPGA
2. Modify ser.py (section 2.4) to specify the correct uart port on PC and .dat byte size (should be the same size as in R2U2_pkg.vhd.
3. Run `sudo python ser.py`. This script will send atomic checker's and future time monitor's configuration to the board automatically.
4. Type binary data as input (same format as each line in logged_data.dat) and press enter. You will see the result displayed in the terminal window.

Figure 1: Zedboard connection setup

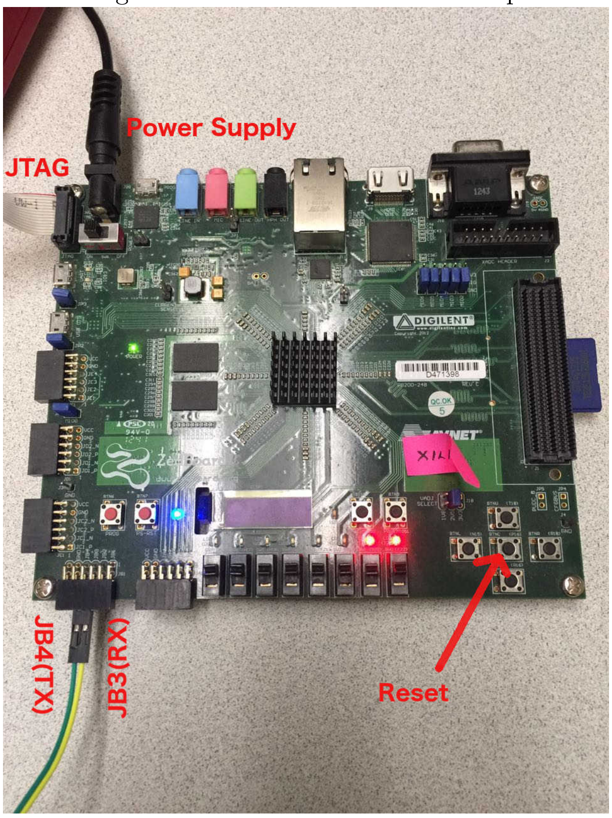
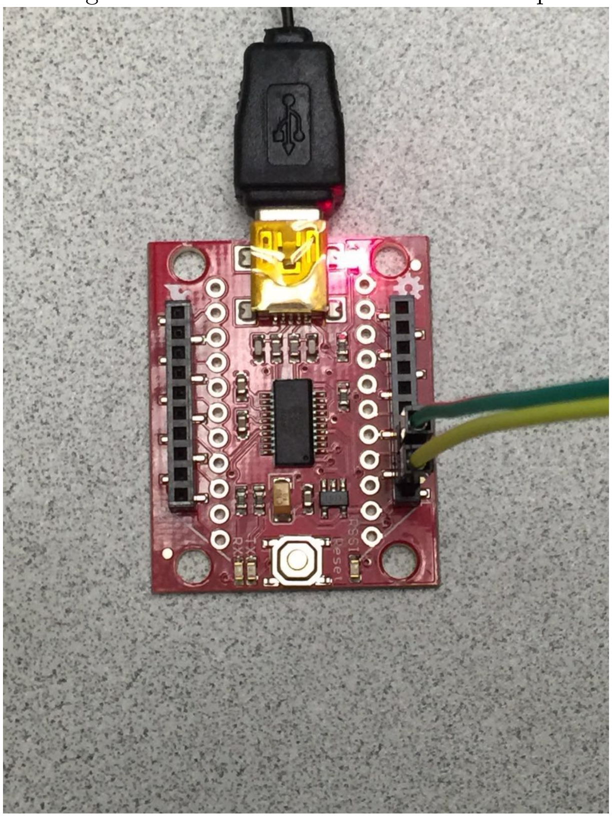


Figure 2: UART module connection setup



2 Details of Scripts and Files

2.1 inputs.py

(only compatiable with python 2.x)
Processing raw data file (similar to .csv) to the data format we want.
Parameters and functions:

1. DATA_SET: Raw data set file

2. class subclass(CsvParser):
   e.g.

```
class Gs111m(CsvParser):
        def __init__(self):
                CsvParser.__init__(self)
                self.file = DATA_SET + "/gs111m_0.csv"
                self.addConfig(9, "float", 10, 24, "roll_angle", "in_1/2^10_rad")
                self.addConfig(8, "float", 10, 24, "pitch_angle", "in_1/2^10_rad")
```

   Comment: Create a subclass. It will subtract the data you mentioned in **self.addConfig** and output the processed data into **self.file** The self.file will look like:

roll_angle	pitch_angle
-42	12
-34	6
-25	3
...	...

   ▷ self.addConfig(channel, type, comma, width, name, comment)
   @ channel: Column of the data in the raw data file.
   @ type: Float or not. If it's float, write "float", else leave null.
   @ comma: Only used for floating data type. It specifies how many fraction bit in binary you want to reserve during the conversion.
   @ width: Number of bits for this data.
   @ name: Specify the name of the column data. This will affect the name used in the .vhd code.
   @ comment: Add any comment in string as you want.

3. class subclass(AtChecker):
   e.g.

```
class AtCheckerConfig(AtChecker):
        def __init__(self, inputFiles):
                AtChecker.__init__(self, inputFiles)
                self.add("pitch_angle", "", 1, "", "", "", "")
                self.add("roll_angle", "", 1, "", "", "", "")
```

   Comment: Create a subclass. The subclass specifies the filters, number of atomic checkers, etc.. The class will affect at_checker.vhd
   ▷ self.add(input1, input2, count, filter1, filter2, rate1, rate2)

@ input1: First input to the at_checker
@ input2: Second input to the at_checker, usually used for compare with @input1. We can leave it as a null string ””.
@ count: How many atomic checkers you want for this signal.
@ filter1: Hardware filter you want to use for input1. The filter name should be the same as the hardware filter component
@ filter2: Hardware filter you want to use for input2.
@ rate1: Signal delta during each sampling clk for input1. Leave null if you want to monitor signal delta.
@ rate2: Signal delta during each sampling clk for input2.

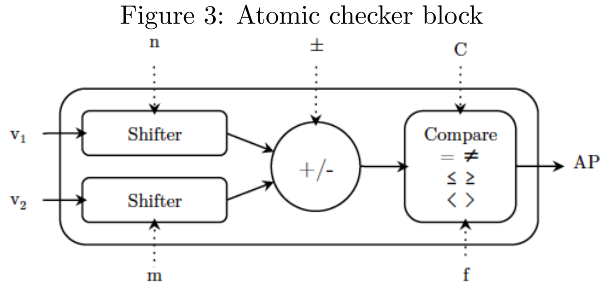
2.2 input.ast

Atomic assertion file. Specify the at_checker.vhd operation mode. For details, refer to **Performance_Aware_Hardware_Runtime_Monitors.pdf**.
e.g.

```
-16;0;0;+;>
1;0;0;+;=
100;0;0;+;>
```

Each line specifies the configuration for one atomic checker block. Consider the structure of an

Figure 3: Atomic checker block



atomic checker, depicted in Figure.3, then each line in the configuration file reads as: C;n;m;+/-;f.

2.3 gen_uart_data.py

Generate uart data byte by byte. Requires **.atc**, **.imem**, **.int** and **.dat** as input. I suggest leave the **.dat** file empty for the demo purpose.
Parameters:
@ SETUP_DATA_WIDTH_extend_byte: **.atc** file configuration data width.
@ SETUP_ADDR_WIDTH_extend_byte: **.atc** file configuration address width.
@ DATA_BYTE_WIDTH_extend_byte: binary logged data bit width (each line width of logged_data.dat).
These three parameters should be the same as in R2U2_pkg.vhd.

2.4 ser.py

1. serial.Serial()
   e.g.

```
ser = serial.Serial(
    port='/dev/ttyUSB0',
    timeout=0,
    # baudrate=9600,
    # parity=serial.PARITY_ODD,
    # stopbits=serial.STOPBITS_TWO,
    # bytesize=serial.SEVENBITS
 )
```

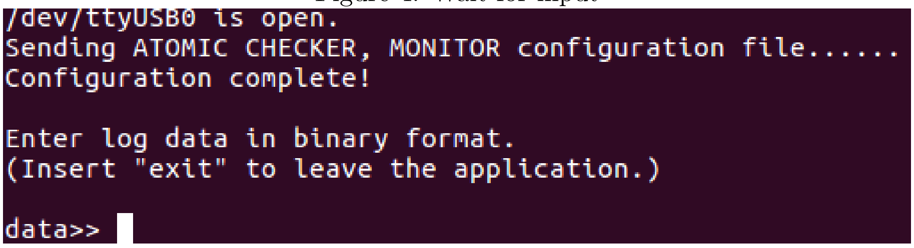
Comment: By default, it is 9600 baud rate 8IN1 mode. You only need to specify the PC port that the UART is connecting to.

2. parameters: @ DATA_BYTE_WIDTH_extend_byte: (same as gen_uart_data.py)

3 Demo Screenshot

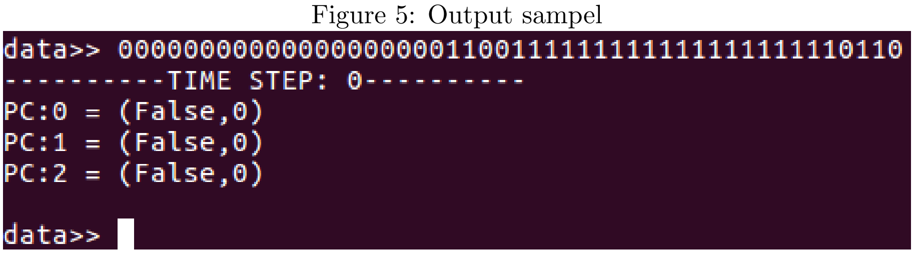
Suppose all data is preprocessed properly. How are the screenshot sample:
1. Run ser.py, it will wait for sensor input as in Figure 4.

Figure 4: Wait for input



2. Copy one line of data from logged_data.dat as sensor input:
The future time monitor output is shown in Figure 5.

Figure 5: Output sampel



You can keep feeding sensor input and see output in each time step.