

Getting started with Vitis & Vivado

06/27/2025

Setting Up

Tools

Download:

- Vitis
- Vivado (I used the 2025.1 version for both)

Physical:

- PYNQ Z2 board
- USB cables to connect to your computer

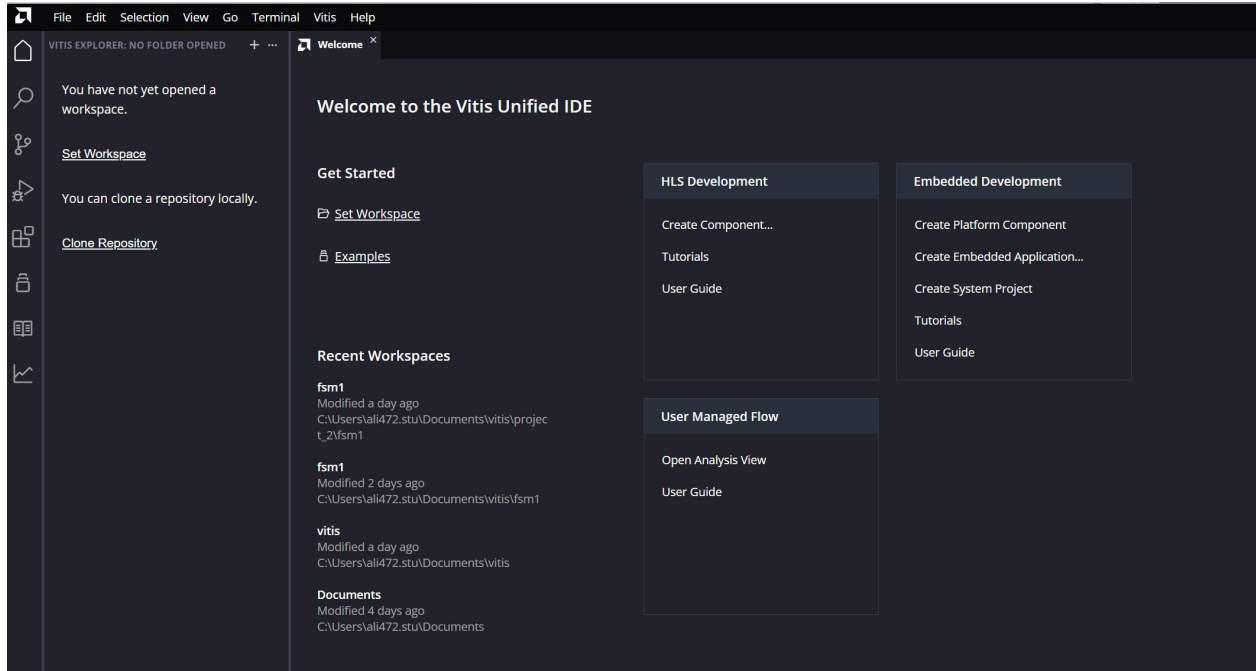
Set up your PYNQ board

- Boot the PYNQ board with the SD card image
- Physically connect the board to your computer with a USB-C cord or other
- Power the board on (little black switch located near the edge of the board)
- Enable the board to access Wifi:
 - Desktop settings -> network and internet -> advanced network settings -> Ethernet 3 -> properties -> enter IP address (192.168.2.99, or google “PYNQ jupyter IP address”)
 - Only need to do this once
- Open a browser to the board's IP
 - In the search bar, type “192.168.2.99” -> this should automatically open JupyterLab
 - If prompted for a username and/or a password, type ‘xilinx’ (all lowercase)
- Now you can use PYNQ's Python APIs to interact with the board!

Vitis + Vivado Workflow

Step 1: Create new Vitis component

Includes specifying the part and the directory containing header files and the top level function

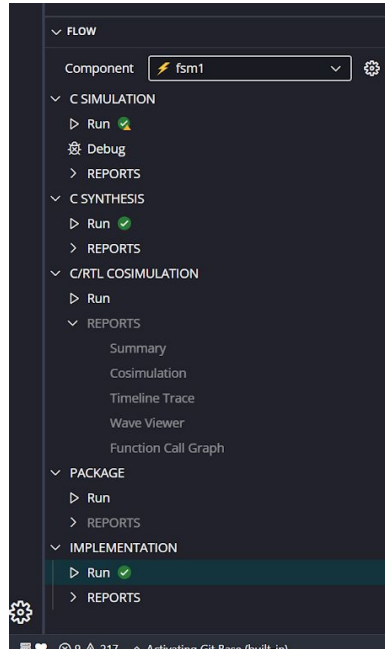


Step 2: Edit, simulate, synthesize, and implement C files in Vitis

(1) 

(2) 

(3) 

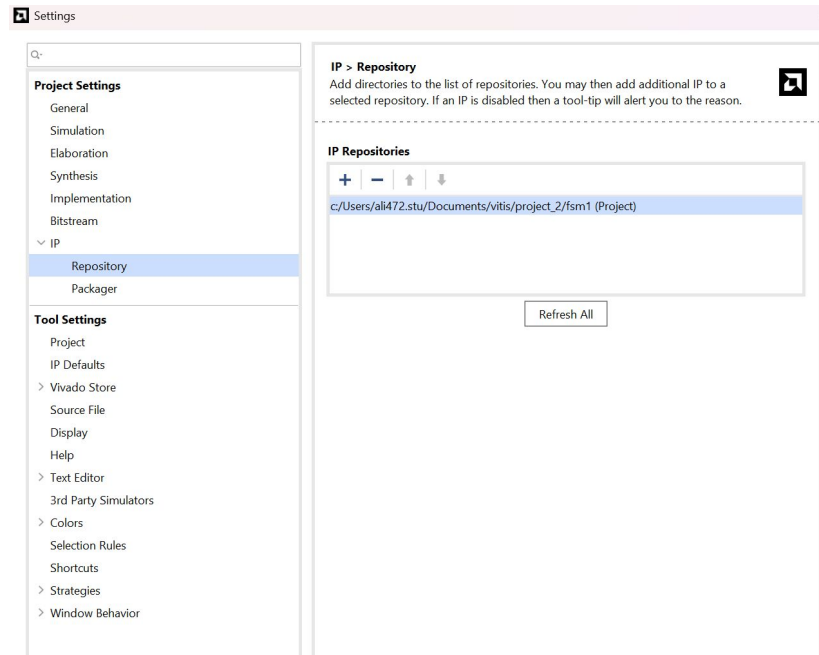


```
10
11
12 #include "algo.h"
13
14 void algo_main()
15 {
16     volatile data_t input_data; // AXI4 burst source
17     state_t user_context; // Read/write, Read/Write, Read/Write, Read/Write
18     flag_t busy; // Output: 1 when active, 0 when idle
19     row_t read_row, col_t read_col;
20     data_t read_data;
21     count_t *states_visited_out; // Output: number of states visited
22     state_t *state_out;
23     flag_t *done;
24 }
25
26 #pragma HLS INTERFACE s_axi4 port=input_data offset=slave bundle=green depth=ROM2_COLS
27 #pragma HLS INTERFACE s_axiwrite port=input_data bundle=control
28 #pragma HLS INTERFACE s_axiwrite port=user_context bundle=control
29 #pragma HLS INTERFACE s_axiwrite port=busy bundle=control
30 #pragma HLS INTERFACE s_axiwrite port=state_out bundle=control
31 #pragma HLS INTERFACE s_axiwrite port=read_row bundle=control
32 #pragma HLS INTERFACE s_axiwrite port=read_col bundle=control
33 #pragma HLS INTERFACE s_axiwrite port=read_data bundle=control
34 #pragma HLS INTERFACE s_axiwrite port=states_visited_out bundle=control
35 #pragma HLS INTERFACE s_axiwrite port=done bundle=control
36 #pragma HLS INTERFACE s_axiwrite port=return bundle=control
37
38 static data_t bram[ROWS][COLS];
39
40 //void _magma_hls_initialize_variable(variable_type_t* var, int dim)
41 #pragma HLS BIND_STORAGE variable=bram type=RAM_2P impl=BRAM
42
43 #pragma HLS ARRAY_RESHAPE variable=bram complete dim=2
44
45 // FSM state variable
46 static volatile fsm_state_t state = IDLE; //static means the variable's value persists between function calls
47 static count_t states_visited = 0;
48
49 #define g // Initialize to non-zero
```

Example C code snippet

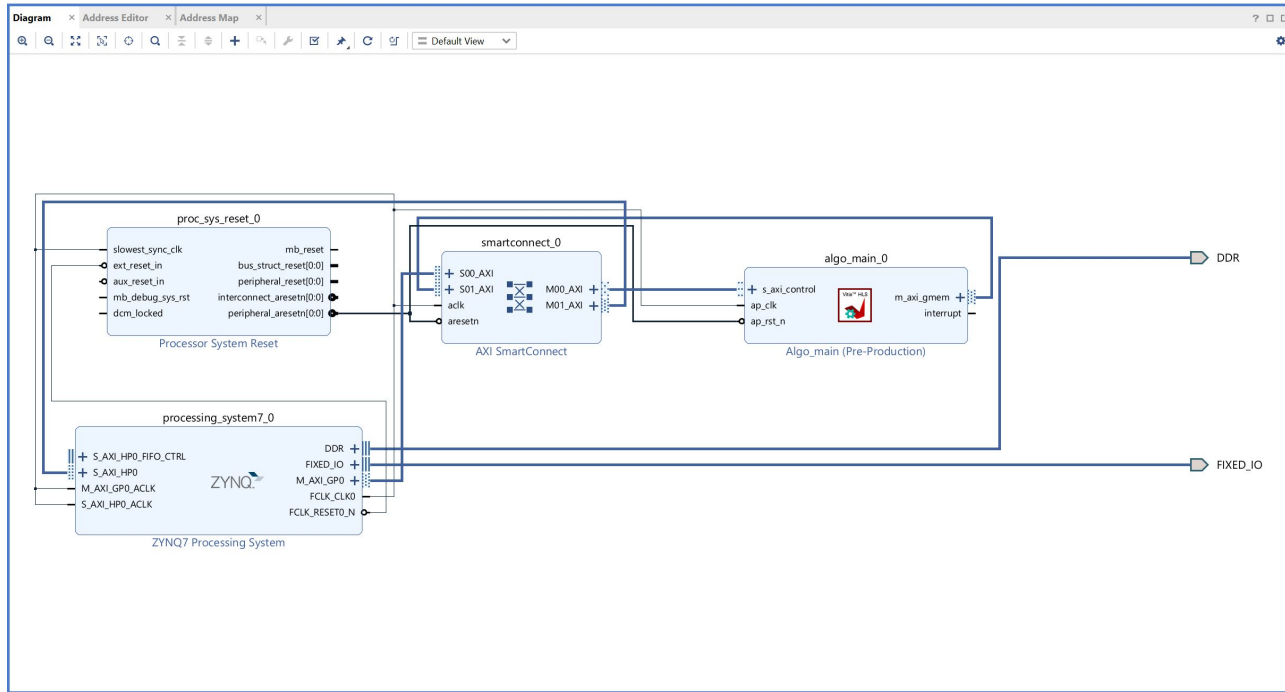
Step 3: Create new project in Vivado

Set repository to the upper-level folder containing the top level function and header files (from Project Manager -> settings)



Step 4: Create Block Diagram

Can accept automatic connections, but may need to create new connections and add ports (eg. for m_axi_gmem on the algo_main block)



Step 5: Run synthesis, implementation, and generate bitstream

- ▼ RTL ANALYSIS
 - ▶ Run Linter
 - > Open Elaborated Design
- ▼ SYNTHESIS
 - ▶ Run Synthesis
 - > Open Synthesized Design
- ▼ IMPLEMENTATION
 - ▶ Run Implementation
 - > Open Implemented Design
- ▼ PROGRAM AND DEBUG
 - ▶ Generate Bitstream
 - > Open Hardware Manager



(1)



(2)



(3)

Found under the
left-hand flow
navigation bar

Step 6: Test hardware implementation with JupyterLab

- Log in to the correct IP address
- Upload .bit and .hwh files from Vivado to the same folder as the python script, ensuring that both files have the same name
- Run desired test cases and see outputs
- Note: repeat the workflow every time a change is made to the top function in Vitis

In [9]:

```
1  # This should not increment nstates_visited in idle, otherwise should increment fsm
2  ip.register_map.user_control = 0
3  ip.register_map.CTRL.AP_START = 0
4  ip.register_map.CTRL.AP_START = 1
5  print("state_out =", ip.register_map.state_out)
6  print("nstates_visited", ip.register_map.nstates_visited_out)
7  print("busy =", ip.register_map.busy)
8  print("CTRL.AP_DONE =", ip.register_map.CTRL.AP_DONE)
9  print("CTRL.AP_START =", ip.register_map.CTRL.AP_START)
10 print("Test output (for debugging purposes) = ", ip.register_map.test_output)
```

```
state_out = 0x5
nstates_visited 0x7
busy = 0x1
CTRL.AP_DONE = 1
CTRL.AP_START = 0
Test output (for debugging purposes) = 0x4
```

Debugging tips and solutions to common problems

Problems I ran into:

C level (Vitis)

- Unexpected outputs from Jupyter test cases
 - Debug using the test bench (first level of debugging: assess the C code)
 - Co-sim requires specification of fifo depth (under m_axi pragma)

Vivado level

- FSM getting stuck in “load_matrix” state
 - Block diagram not properly connected
- FSM loading unexpected data
 - Ensure addresses have been properly assigned (in the block diagram); right click anywhere in the address editor and unassign and re-assign all

Jupyter level

- Overlay() referencing old files or outputting “resource busy”
 - Reboot the board or run PL.reset()

Debugging process

- 1) Assess correctness of C code with Vitis test bench (run C simulation)

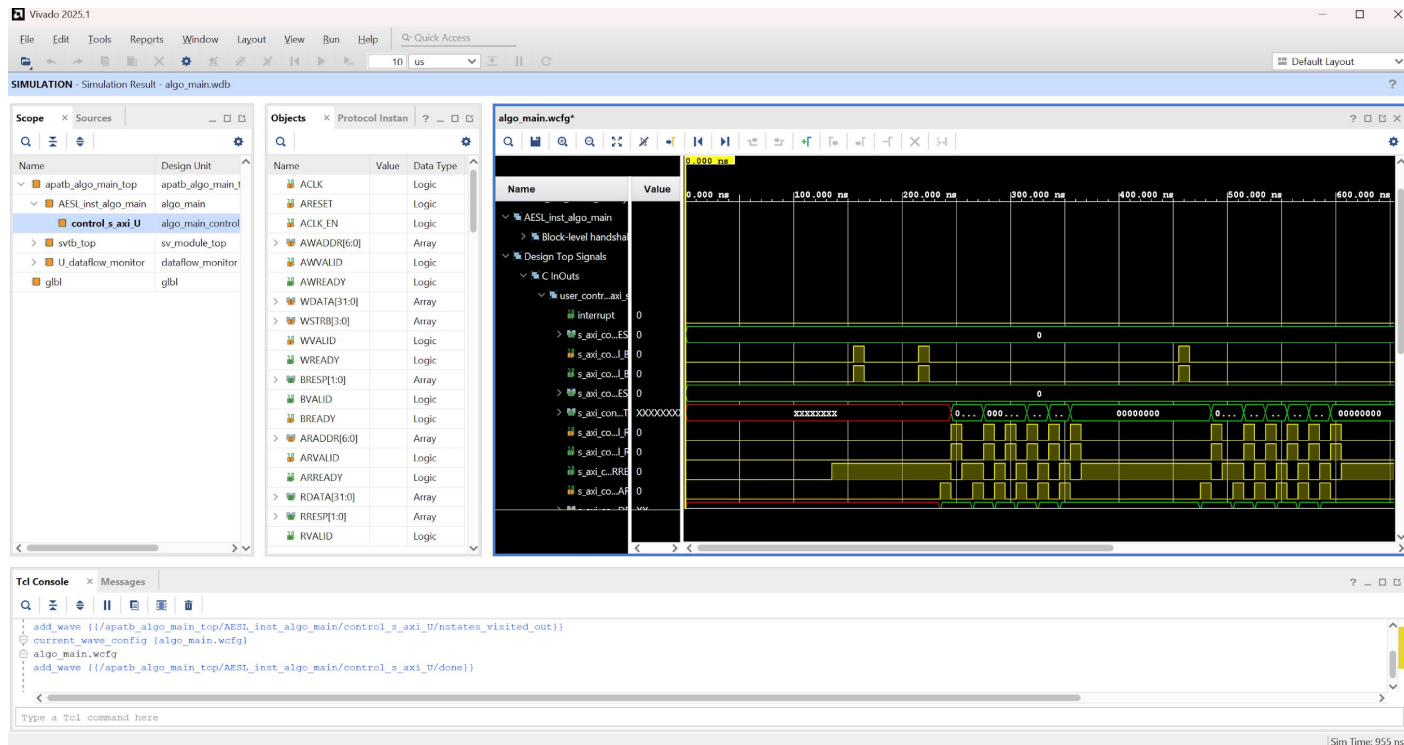
```
54
55     algo_main(array,
56               ctrl,
57               &busy,
58               row_to_read, col_to_read,
59               &output_data,
60               &nstates_visited_out,
61               &state_out,
62               &done);
63
64     printf("Number of states visited: %d\n", nstates_visited_out&0xffff);
65
66     printf("Output data: %d\n", output_data&0xffff);
67     printf("Expected output: %d\n", array[row_to_read * COLS + col_to_read]&0xffff);
68
69
70     return 0;
71 }
72
```

OUTPUT x PROBLEMS x fsm1:c-simulation

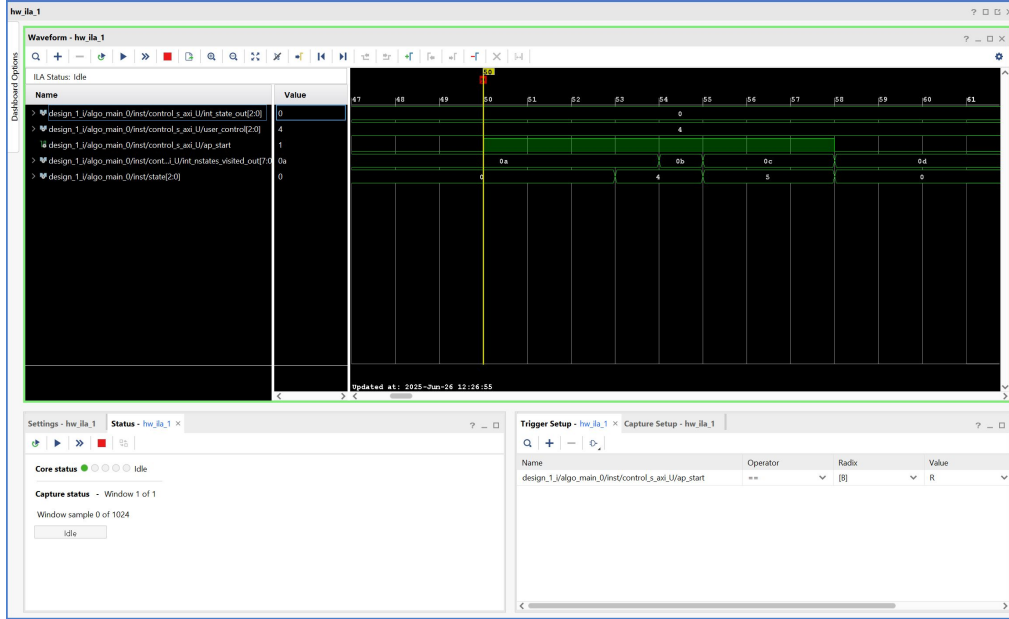
Vitis Messages fsm1:c-simulation x

```
44 In file included from C:/Xilinx/2025.1/Vitis/include/floating_point_v7_1_bitacc_cmodel.h:150:
45 C:/Xilinx/2025.1/Vitis/include/gmp.h:58:9: warning: '___GMP_LIBGMP_DLL' macro redefined [-Wmacro-redefined]
46 #define ___GMP_LIBGMP_DLL 0
47   ^
48 C:/Xilinx/2025.1/Vitis/include/floating_point_v7_1_bitacc_cmodel.h:142:9: note: previous definition is here
49 #define ___GMP_LIBGMP_DLL 1
50   ^
51 1 warning generated.
52 Starting from ctrl = 1
53 Loading matrix...
54 Number of states visited: 3
55 -----
56 Reading from matrix...
57 Number of states visited: 6
58 Output data: 300
59 Expected output: 300
60 INFO: [SIM 211-1] CSim done with 0 errors.
61 INFO: [SIM 211-3] ***** CSIM finish *****
62 INFO: [HLS 200-112] Total CPU user time: 1 seconds. Total CPU system time: 1 seconds. Total elapsed time: 5.893 seconds; peak allocated memory: 150.750 MB.
63 INFO: [vitis-run 60-791] Total elapsed time: 0h 0m 10s
64 C-simulation finished successfully
65
```

2) Assess predicted signal behaviour with the Vitis Co-simulation



3) Assess actual signal output with Vivado ILA debug option



- Select nets to debug under Synthesis
-> set up debug
 - When prompted, select Capture Control and Advanced Trigger
- Generate bitstream (will automatically ask to run implementation)
- Open Hardware Manager. If this is the first time configuring debug ILA cores for the project, select Open Target; else click Program Device
- Set up trigger to capture based on the value of a certain variable (if desired)
- Re-upload the .bit file to Jupyter, run the Overlay() code, enable the trigger in Vivado, and run the test case

Resources used:

Official PYNQ Z2 board set-up tutorial:

https://pynq.readthedocs.io/en/v2.3/getting_started/pynq_z2_setup.html

Official Vivado tutorials for setting up debugging using ILA cores:

[Running the Set Up Debug Wizard • Vivado Design Suite Tutorial: Programming and Debugging \(UG936\) • Reader • AMD Technical Information Portal](#)

[Using the Vivado Integrated Logic Analyzer • Vivado Design Suite Tutorial: Programming and Debugging \(UG936\) • Reader • AMD Technical Information Portal](#)