Virtual Vaccine Passport Scanner for Remote Entry Approval

ECE 532 Group 3:

Mustafa Kanchwala

Guoxian Wu

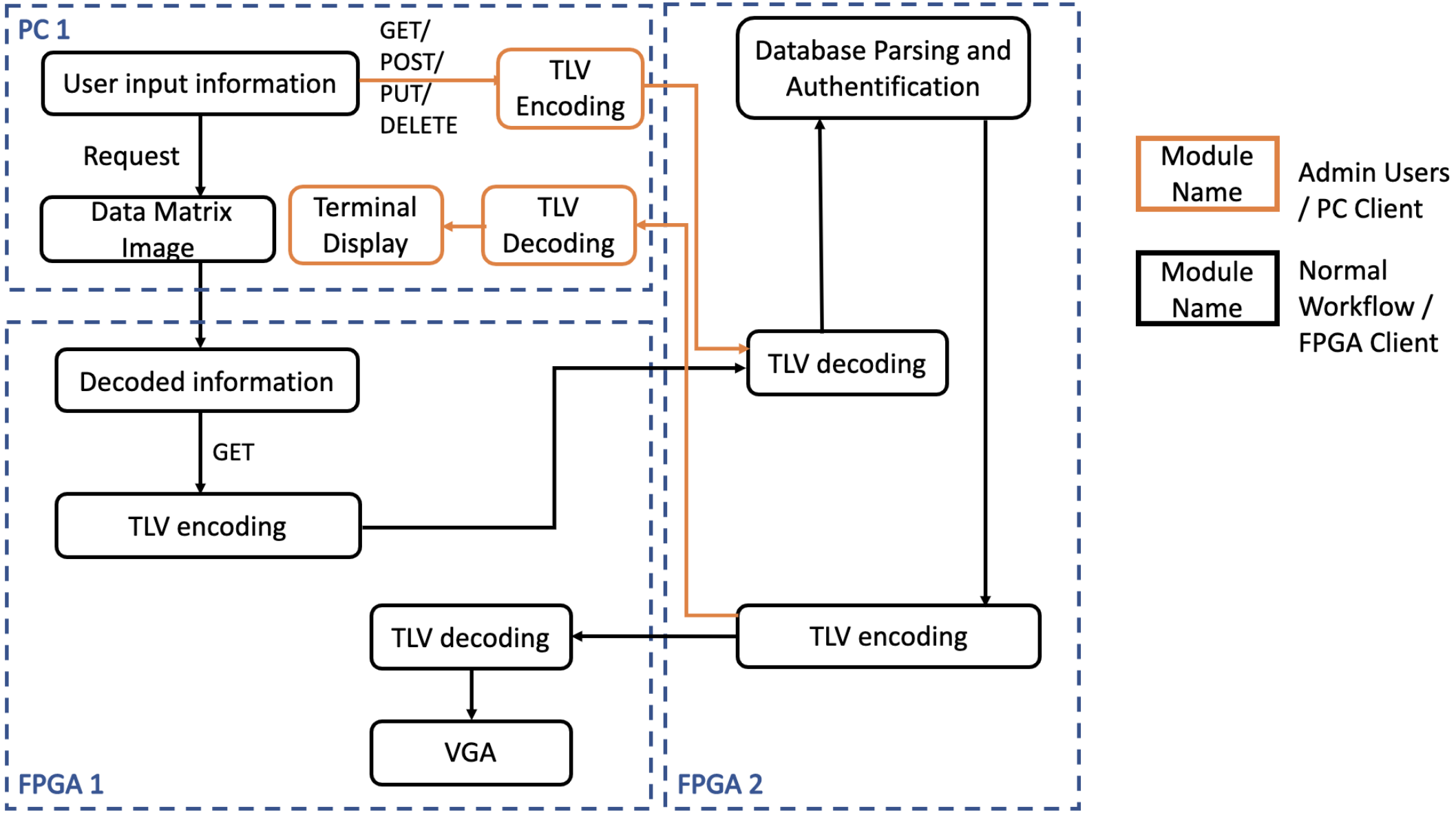
Eduardo Stecca Ortenblad

Xuening Dong

Project Overview

- What's out there:
  - A Human scans the vaccine passport and allows or denies entry
- Why that’s bad:
  - Risk of Exposure – Very High!
  - Conflicts arising from enforcement – High!
- What can be done:
  - A Hardware based - remote server-controlled scanning & verification

System Overview



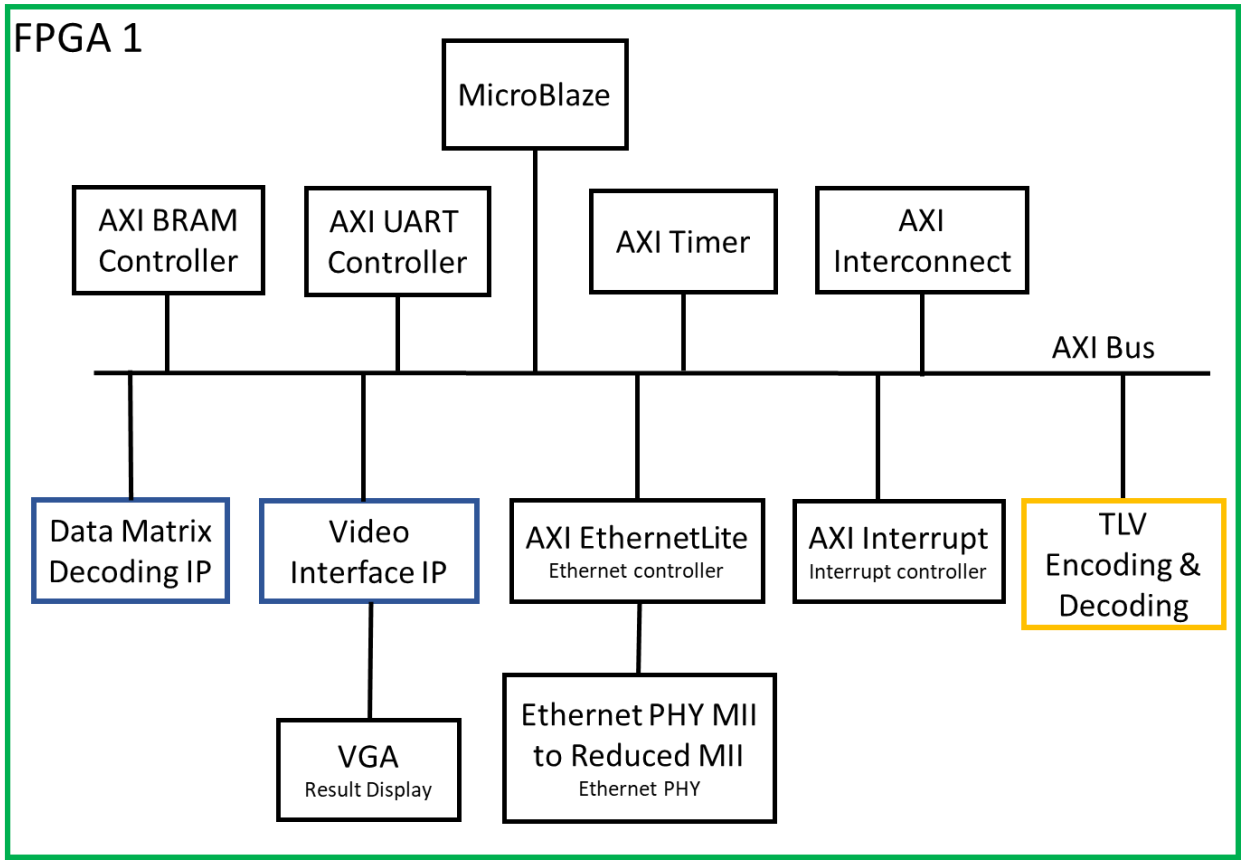
Proposed Block Diagram

Module Name — Custom IP

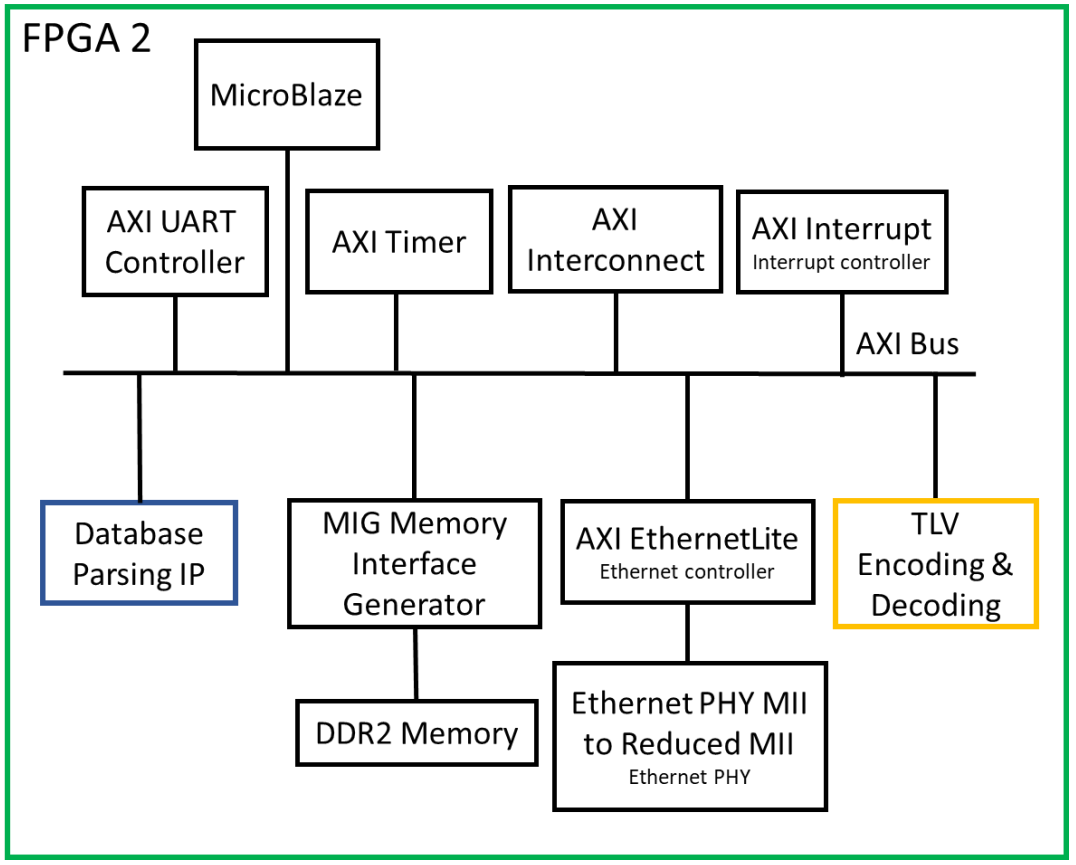
Module Name — Tentative IP

Module Name — Existing IP

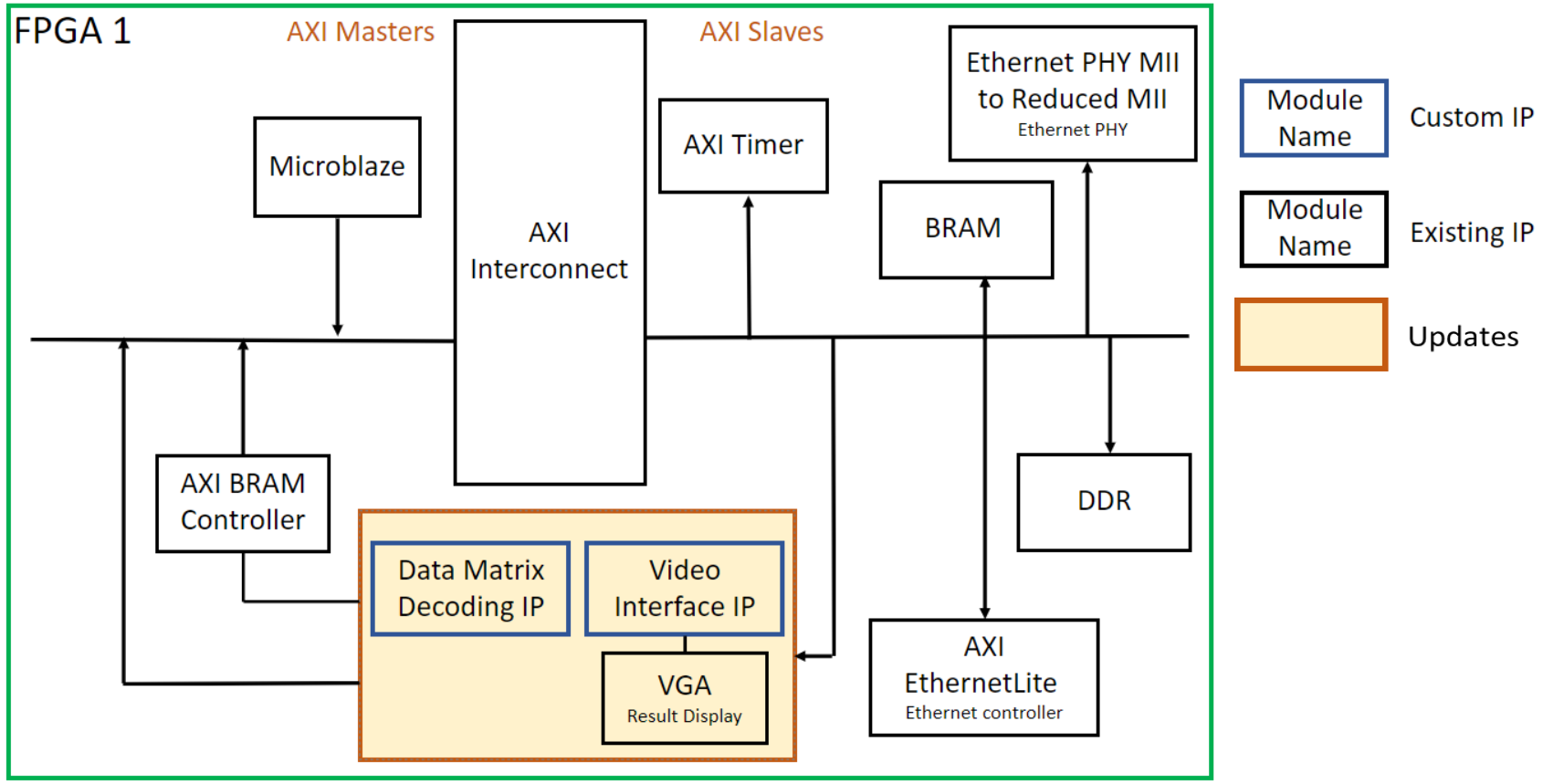
Client



Server

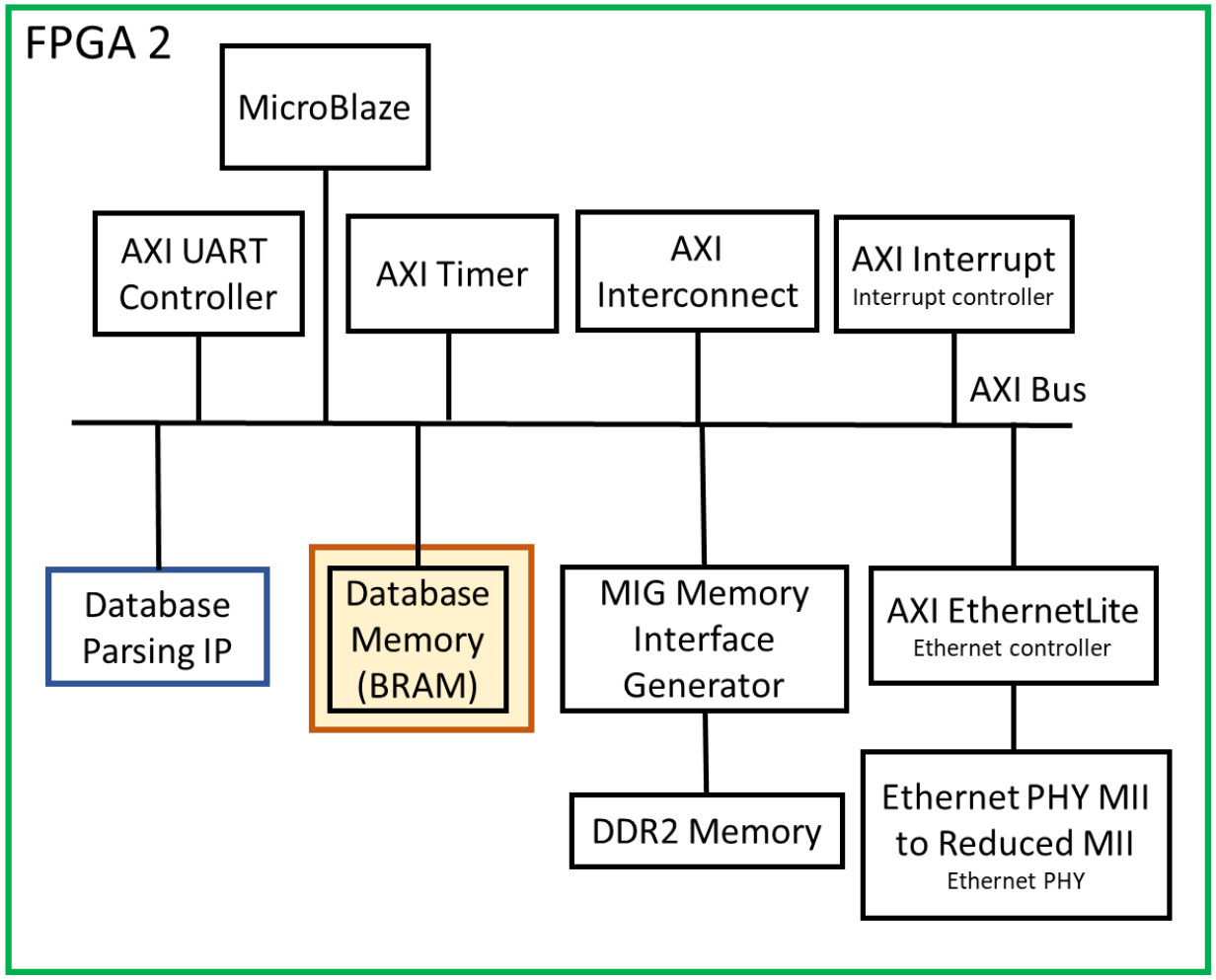


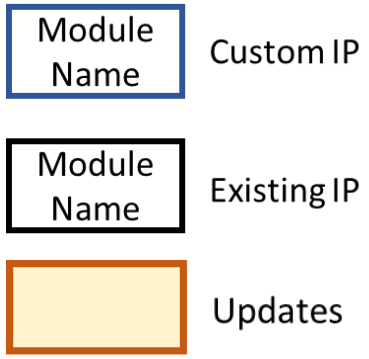
Current System Block Diagram (Client)



- Reason for the updates:
  - TLV block removed as encoding and decoding is performed in Software
  - Datamatrix decoder and VGA are integrated because they are controlled by the same FSM and enable signals.

Current System Block Diagram (Server)





- Reason for the updates:
  - TLV block removed as encoding and decoding is performed in Software
  - Switched to using a large BRAM block instead of using DDR for Database, as DDR is already used by program memory blocks (See challenges)

Challenges

- Overall:
  - **Problem**: AXI Bus Integration of multiple slave and masters
    - The auto connection messes up the AXI bus clock signals with different frequencies
    - Sometimes AXI bus is stuck somewhere and transactions are not transferred as expected
  - **Solution:** Manually connect everything about AXI. Monitor handshakes and transactions using VIP and ILA to ensure the data comes in order

  - **Problem:** Hardware IP integration with Microblaze program (both server and client sides) needed a lot of debugging
  - **Solution:** Performed extensive testing using memory monitors and ILA. Made necessary changes to the IP to resolve the issue

Challenges continued…

- Server:
  - **Problem:** Could not use DDR as the database memory
    - Program code blocks could not fit to other BRAM memory regions when using Ethernet
    - Tried memory splitting for DDR block (not successful)
    - Working with TA to find a possible solution
  - **Solution:** Used a large BRAM block as the database memory (can store 32 unique user's information)

Mid Project Demo - Overall

- Mid Project Demo was shown in 3 parts:
- All parts were demonstrated in FPGA Hardware!

1. FPGA Client integrated with VGA and Datamatrix decoding
2. Operation of Database IP
3. FPGA Server integrated with TLV encoding/decoding scripts

Demo – Part 1 (Datamatrix Decoder)

Software

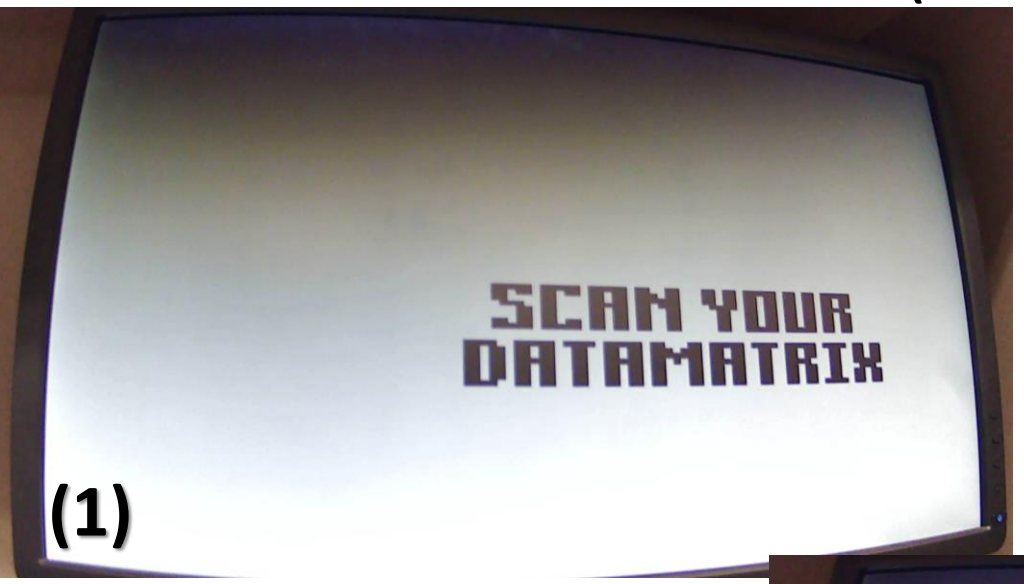
Bitstream of Datamatrix

Hardware IP

Name : BCDEFGHIJK

UID (For database Entry): 0x4141

Demo – Part 1 (Client + VGA & Datamatrix)

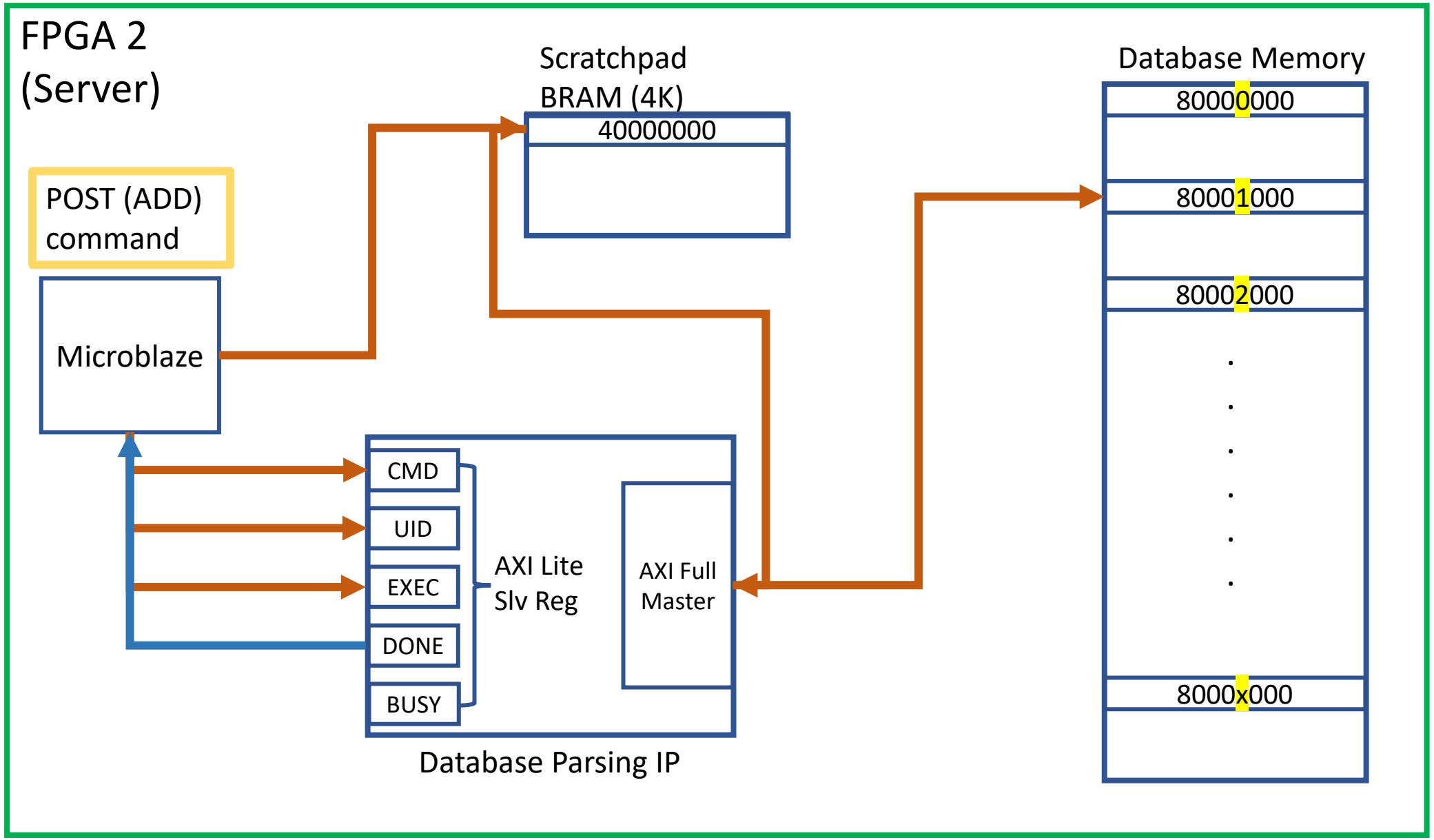


(1) Wait for Datamatrix entry

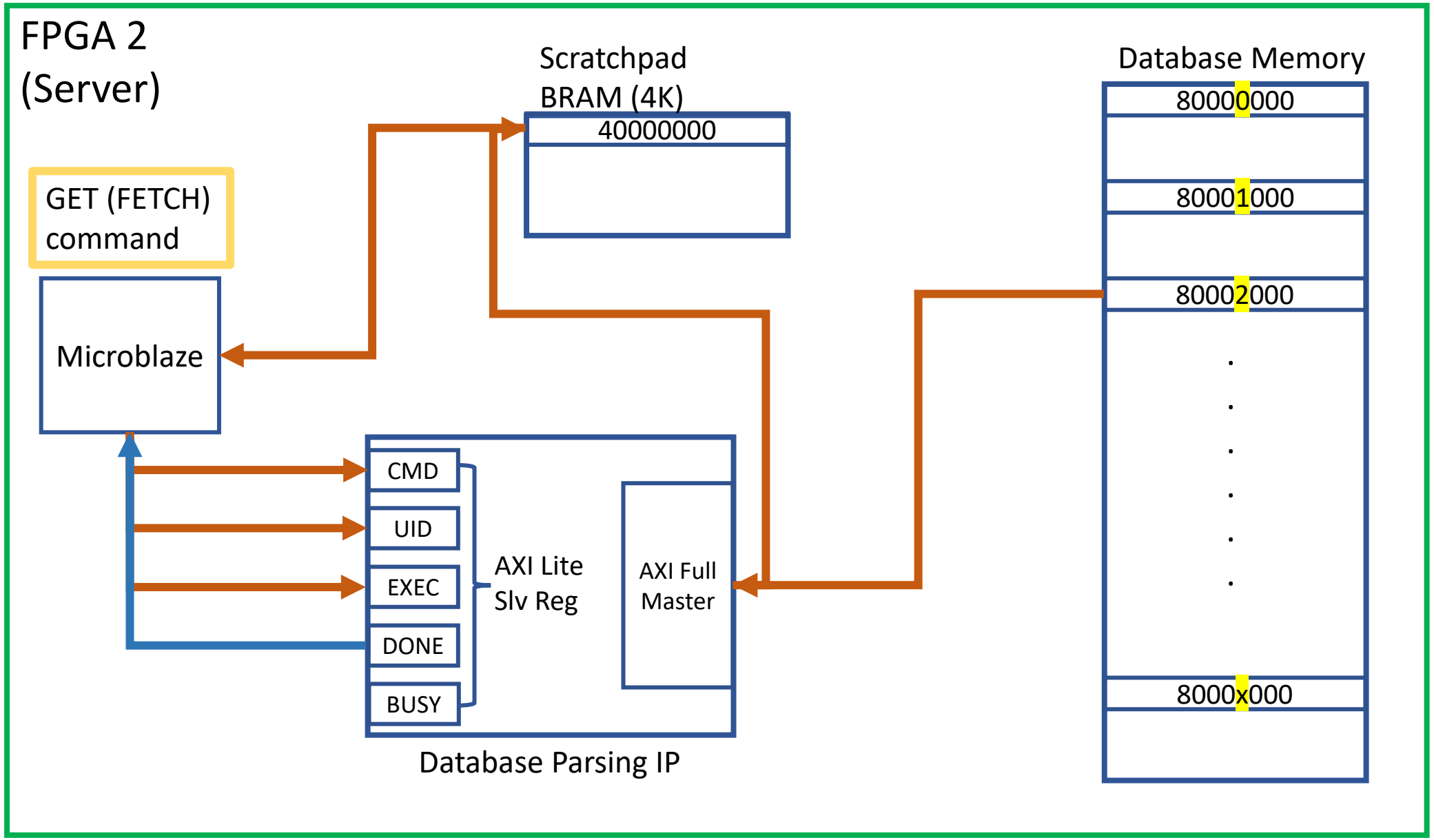
(2) Display decoded name from Datamatrix
Send UID to database and perform Fetch request for database

(3) Display information fetched from database

Demo – Part 2 (Database IP)



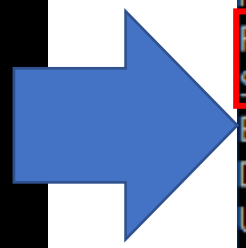
Demo – Part 2 (Database IP)



Demo – Part 3 (Server + TLV script)

C:\Windows\py.exe

```
Sending Request
DBRequest
HEADER:
Request Type = POST
BODY:
DBUser
UID: 1
First Name: Eduardo
Last Name: Stecca
Date of Birth: 1999-02-20
Vaccination Status: Fully Vaccinated
Dose 1 Date = 2021-05-05
Dose 1 Type = Pfizer-BioNTech
Dose 2 Date = 2021-08-05
Dose 2 Type = Pfizer-BioNTech
```



```
Received Raw Bytes
['1', '2', '0', '5', '2', '32', '0', '4',
75', '61', '72', '64', '6f', '2', '6', '5
'36', 'ce', '41', '50', '4', '1', '3', '5
', '7', '1', '5', '8', '4', '61', 'b', '6

Decoded Response:
DBRequest
HEADER:
Request Type = Response
Status Code = 0 | Success
BODY:
DBUser
UID: 1
First Name: Eduardo
Last Name: Stecca
Date of Birth: 1999-02-20
Vaccination Status: Fully Vaccinated
Dose 1 Date = 2021-05-05
Dose 1 Type = Pfizer-BioNTech
Dose 2 Date = 2021-08-05
Dose 2 Type = Pfizer-BioNTech
Press Enter to continue...
```

Things To be done

Must have:

- Integration of FPGA 1 and FPGA 2 – TCP/IP connection (***Initial integration completed on 16-March!***)
- An overall Python script for controlling the workflow

If time permits:

- Process PUT and DELETE requests on FPGA 2 server
- Store database in DDR rather than BRAM
- Display more user information on VGA

Final Demo Plan

PC 1 → FPGA 2 (Database Server)

- Have .json files to send commands to the database server
- Client Python script:
  1. Read JSON from file
  2. Encode to TLV
  3. Send request to FPGA 2
  4. Display decoded response on terminal
  5. Upon successful response to POST request, generate Datamatrix for new user

Final Demo Plan

PC 1 → FPGA 1 (Client)

- User input their message string or data matrix image on PC 1
- Data matrix is sent to FPGA 1 and decoded.
- Decoded message is sent to FPGA 2 (Database Server) to search for permission in database
- Permission is received by FPGA 1 and displayed on VGA

Q&A