Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Hardware-In-The-Loop Emulator For In-Vehicle Controller Area Networks

Changhong Li
Supervisor: Shreejith Shanker

Date 23/05/2024

Background & Project goals

Problem Recap

Background

Hackers Remotely Kill a Jeep on the Highway

Increased complexity → **Increased risk**

Project goals

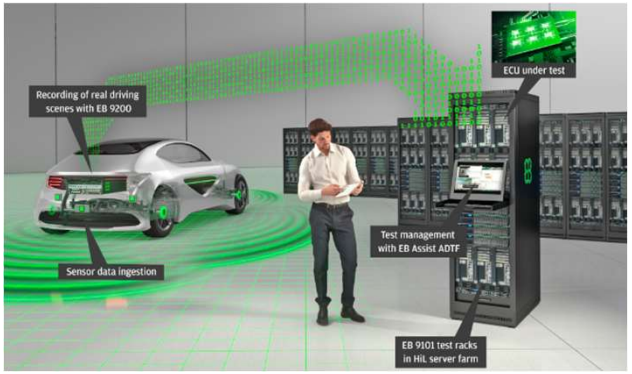
In this project, we will build **a hardware in the loop runtime for vehicular ECUs** using a RISC-V / Microblaze softcore processor as the compute core for each ECU. We will integrate the **ability to inject errors (like DoS, replay and spoofing) into the emulation framework** with multiple ECUs spread across one or more FPGA development boards.

A hardware-in-the-loop emulator for vehicle bus attack simulation & prevention

Hacker exposed the security vulnerabilities in automobiles by hacking into cars remotely, controlling the cars' various controls from the radio volume to the brakes on Wednesday, July 1**, 2015 in Ladue**

Previous work & Novelty

Previous work - HWIL



Hardware-in-the-loop simulation was first used in the 1960s in the **aerospace field**, it was used to test flight control systems, and now it is also widely used in the automotive field.[1]

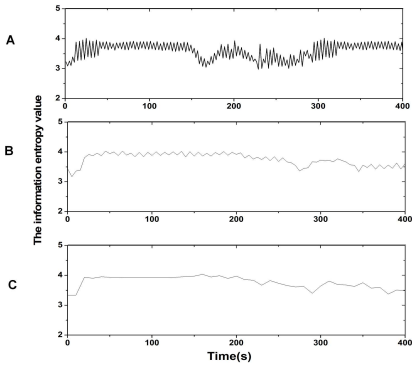
The applications in the **automotive field**, HIL provides a virtual vehicle for system-level verification.

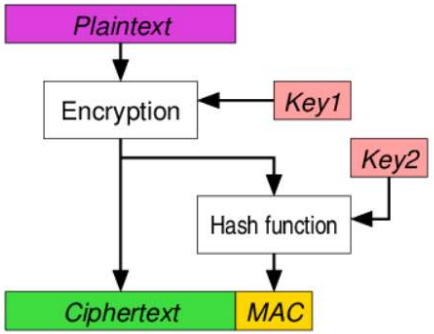
Virtual ECU technology is in the ascendant

[1] S Raman, N Sivashankar, W Milam, W Stuart, and S Nabi. Design and implementation of hil simulators for powertrain control system software development. In Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251), volume 1, pages 709–713. IEEE, 1999

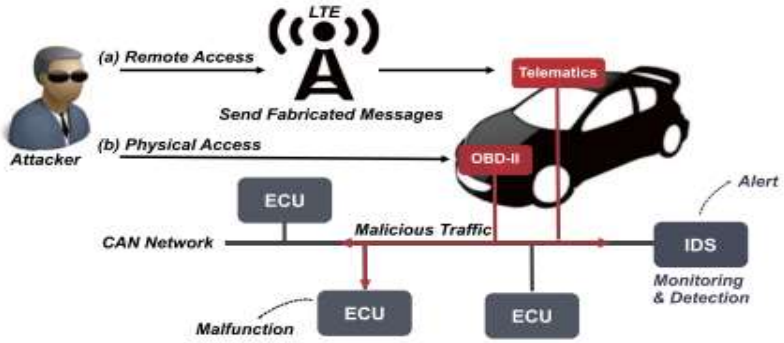
Previous work & Novelty

Previous work – attack detection





Traditional Detection & Encryption



IDS & DNN detection

Neural networks have greatly improved attack detection performance,

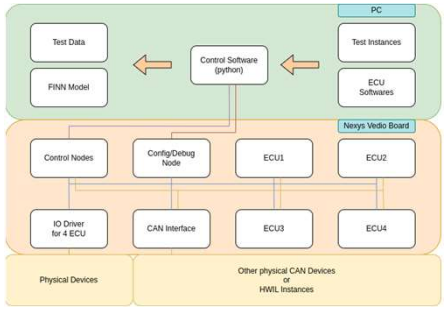
But how to deal with flexible system changes and long development cycle

Agile attack simulation & model building toolchain to be developed

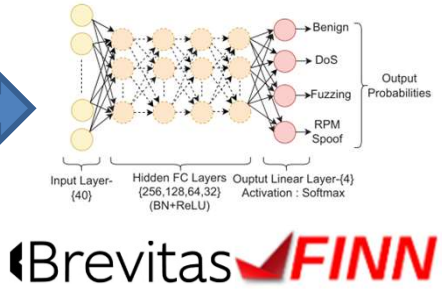
Method	Precision (%)		Recall (%)		F1-Score (%)	
	Known	*New*	*Known*	*New*	*Known*	*New*
OCSVM	35.43	10.83	71.15	35.12	47.30	16.55
IF	43.58	15.62	73.42	31.56	54.69	20.90
OTIDS	**99.82**	70.81	71.68	42.01	83.44	52.73
RNN+Heuristics	98.69	70.25	**99.49**	50.53	**99.09**	58.78
CANTransfer	94.93	**87.97**	95.57	**88.97**	95.25	**88.47**
Gain	-4.89	**17.16**	-3.92	**38.44**	-3.84	**26.69**

Progress Review

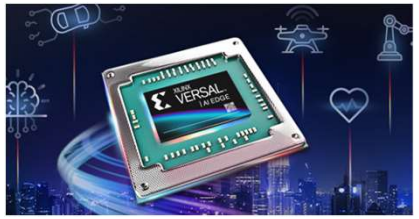
Project Structure



Flexible
HWIL Emulator



High-Performance
QNN Accelerators



Deployable
Stitched IP for Edge

Flexibly Changeable
System Structure

Complete CAN bus Test & Detection Solution

Empower

Automotive Safety

Progress Review

Milestone Review

Gantt Chart

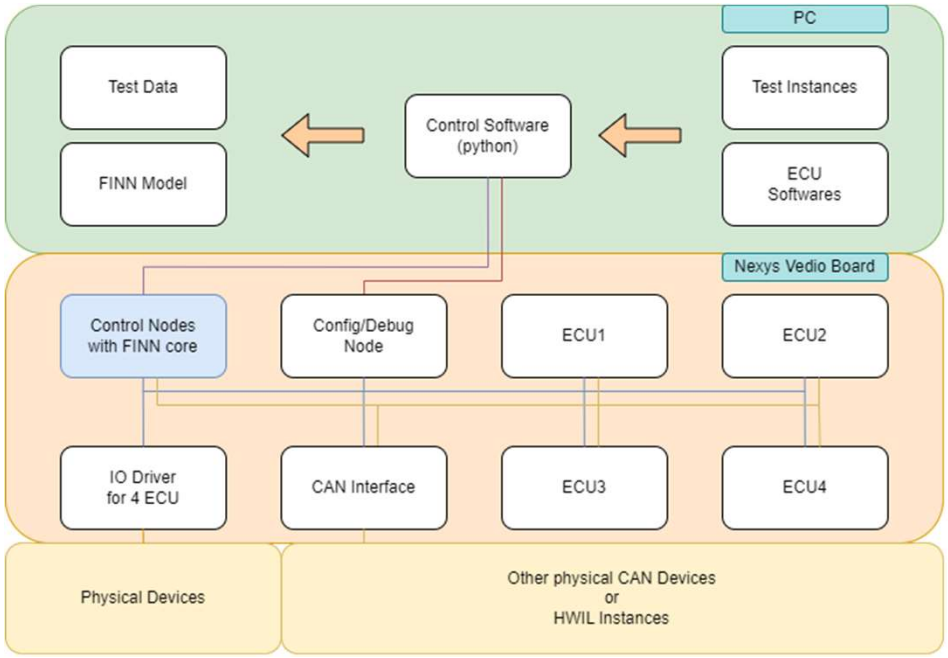
No	Month Task	Task	Description	Nov				Dec				Jan				Feb				Mar				April				May				June				July			
				1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	build workflow, get familiar to the Env	Linux Dev Env and board	get Nexys-Vedio board and build development enviroment of the project on linux																																				
		Risc-V Vivado Project	build risc-v vivado project and finish basic simulation			*																																	
2	build Risc-V core and testify the communication between cores	Risc-V core	build risc-v core and dowload it to the FPGA board																																				
		CAN controller core	build the CAN controller core form opencore																																				
		bare-metal program and test	build the bare-metal program of risc-v, basic BSP program to drive gpio like switches and leds and																																				
		CAN transceiver core	to make it possible to communicate on the board without a physical CAN transceiver, we need to																																				
		testify comm	verify the communication between two cores with CAN bus IP and its driver code										*																										
3	build 4-core risc-v and mircoblaze project and Wrap the can cantroller to a AXI mode to speed it up	build 4-core prj	build 4-core risc-v block design and adjust the structure of the project																																				
		wrap CAN controller	wrap the CAN controller to AXI mode in order to speed up the control process																																				
		Interim Report(Jan 26)	Finish the Interim report and determine the structure of the project												*																								
4	build Comm to PC and get familiar to the DNN ip core	build Ethernet node and interface	build the Ethernet node on the board to send command and monitor the risc-v cores																																				
		build basic python script	build the python script on PC to operate the risc-v core, make it possible to download software and																																				
		build DNN core	get familiar and build the DNN core on the FPGA																																				
5	error Injection and DNN detection data preparation	inject errors to the CAN bus	inject errors like Dos, Replay, Spoofing to the CAN net with the software on PC																		*																		
		data gathering	gather the data of CAN bus for training the DNN model and for pre-processing																																				
		DNN core utilization	utilize the DNN core to detect the attack on the CAN bus, build the basic framework																																				
6	build benchmark of the DNN model	build benchmark of the DNN model	build benchmark of the DNN model																																				
		Milestone Progress Report(April 24)	Milestone Progress Report																								*												
7	deploy the core on the FPGA progress Presentation	Progress Presentation(May 17)	Progress Presentation																										*										
8	Written Dissertation	Dissertation(1st version)	Written Dissertation(1st version)																																*				
9	Written Dissertation	Dissertation(final version)(July 12)	Written Dissertation(final version)																																		*		

Main goals

- Build virtual ECU cores
- Build virtual CAN bus network
- Build ECU control logic
- Build ECU attack functions
- Build attack detection model
- Integrate the model onto hardware
- Benchmarking

Progress Review

HWIL Emulator - structure



HWIL Emulator structure

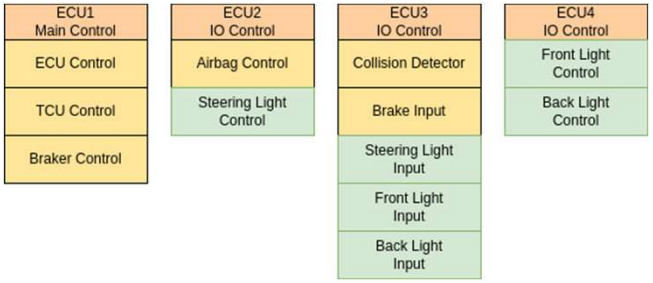
- **Flexible nodes(type, scale)**
- **Physical device access**
- **Automatic testing**
- **Complete tool chain**

Network Change	Test&Data	Model Deployment
Software Define	**Auto Annotation**	**FINN Tool chain**

Accelerate Deployment

Progress Review

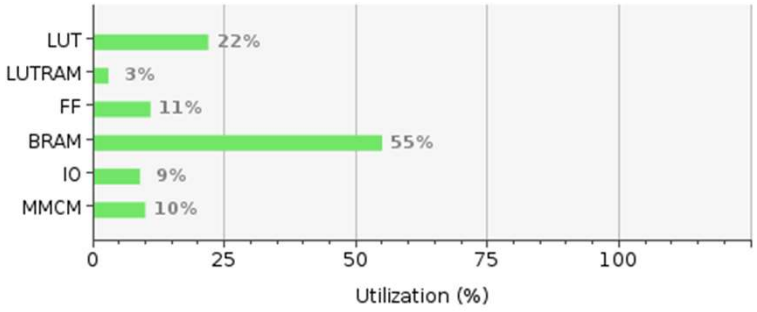
Virtual ECUs



Simulating CAN bus topology

Hardware Implementation Platform

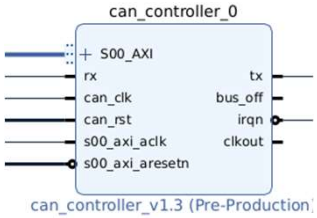
Soft MCU Core: AMD Microblaze



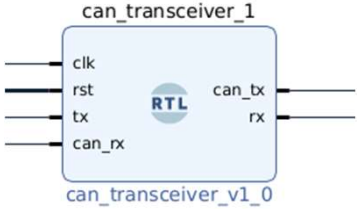
Effective Resource Utilization - Scalable

Progress Review

Virtual CAN BUS network



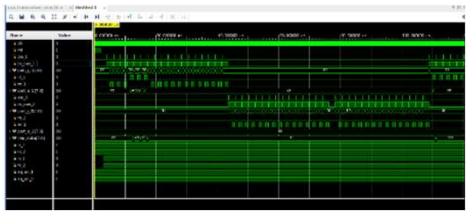
AXI CAN controller IP core



CAN transceiver IP core

```
245 void can_init(uint32_t can_addr);
246 void can_set_pin(uint32_t can_addr, uint8_t pin, uint8_t value);
247 void can_set_data(uint32_t can_addr, uint8_t value);
248 uint8_t can_read_data(uint32_t can_addr);
249 void write_register(uint32_t can_addr, uint8_t reg_addr, uint8_t reg_data);
250 uint8_t read_register(uint32_t can_addr, uint8_t reg_addr);
251 void can_txd(uint32_t can_addr);
252 void can_rxd(uint32_t can_addr);
253 void can_txd_frame(struct can_data_frame *frame);
254 struct can_data_frame initialize_can_data_frame(addr1,addr2,addr3,can_addr);
255 void can_rxd_frame(struct can_data_frame *frame);
```

Standard driver

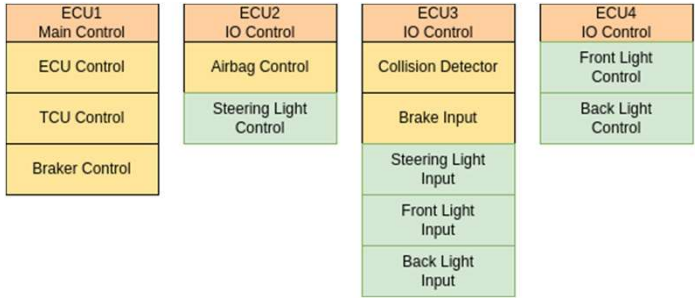


IP core timing simulation

Progress Review

Virtual ECU control logic

Virtual communication protocol



			7	6	5	4	3	2	1	0	
0	0	0	CAN mode =1 extend	Data Type =0 data			Data Length = 8				
1		1	CAN ADDR1(ECU3 = 06H)								
2	1	2	CAN ADDR2(ECU3 = 06H)								
3		3	CAN ADDR3(ECU3 = 06H)								
4	2	4	FIXED VALUE(F8H)								
5		5	Life Signal								
6	3	6	Collision Detector Status								
7		7	Brake Detector Status								
8	4	8	Steering Light Button Detector	Front Light Button Detector Status	Back Light Button Detector Status						
9		9	Control Flag								
10	5	10									
11		11									
12	6	12									

Build up 4 ECUs with different logic for the test.
All ECUs: Generate Life signal to indicate that they are alive. periodically send status messages to report their status.

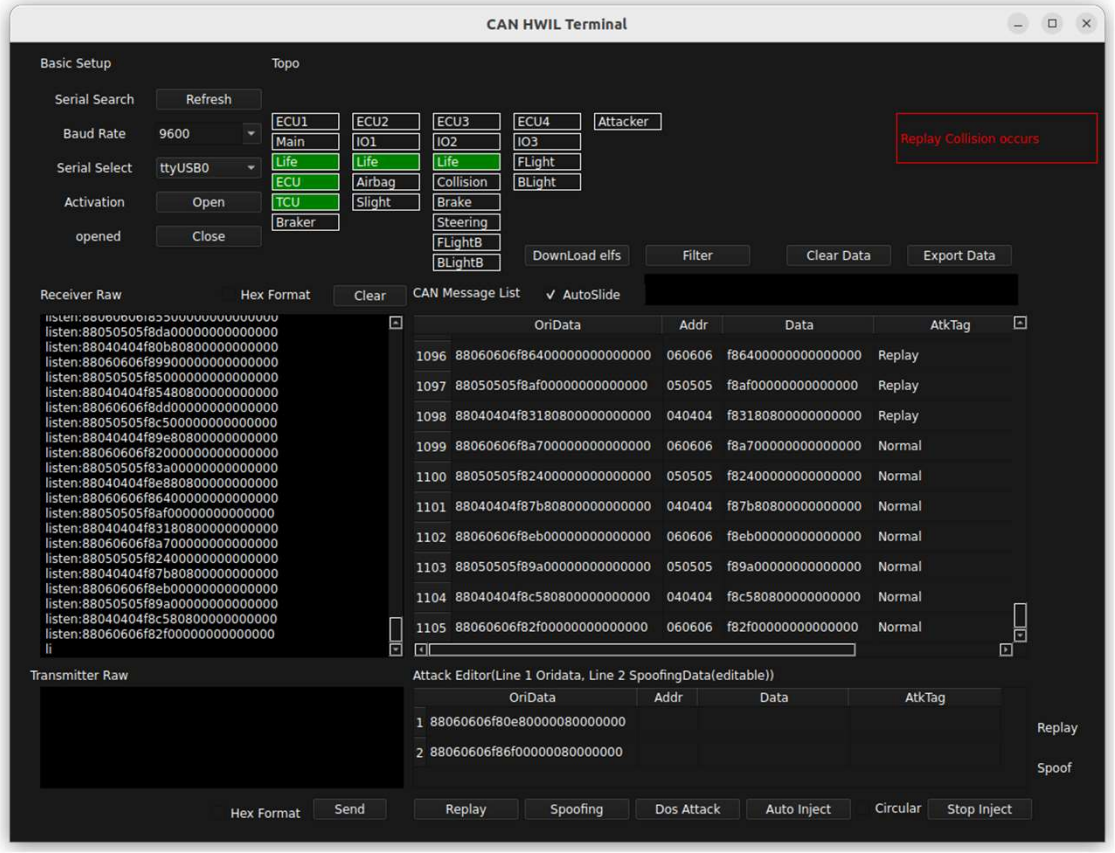
ECU1: Control Engines, transmission unit and brake unit.
ECU2: Control Airbags and Steering Lights.
ECU3: Physical IO inputs, receive the signal from the sensors like Collision detector, brake detector, Lights control switches
ECU4: Control Front light and back light of the car.

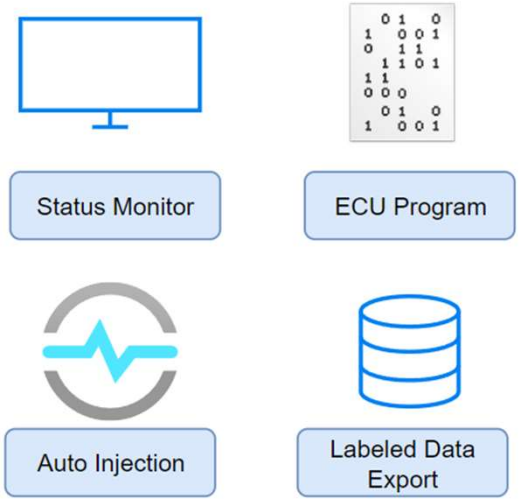
All the ECUs communicate with CAN bus.

Building a virtual communication protocol And build ECU collaborative applications

Progress Review

ECU Attack Test Platform






```
1  No,Type,Msg
2  1,Dos,88000000f8000000000000000
3  2,Spoofing,88040404f81580880400800030
4  3,Replay,88040404f81580800000000000
5  4,Dos,88000000f8000000000000000
6  5,Dos,88000000f8000000000000000
7  6,Dos,88000000f8000000000000000
8  7,Spoofing,88040404f81580880400800030
```

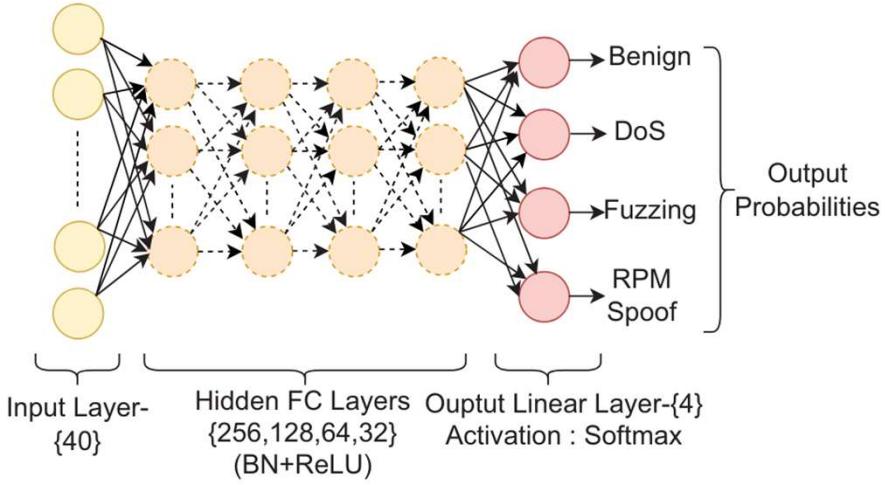
Attack Sequence

```
239  237,0505,8,21,00,00,00,00,00,00,Normal
240  238,0606,8,85,00,00,00,00,00,00,Normal
241  239,0404,8,af,80,80,00,00,00,00,Dos
242  240,0505,8,96,00,00,00,00,00,00,Dos
243  241,0606,8,c9,00,00,00,00,00,00,Dos
244  242,0404,8,f9,80,80,00,00,00,00,Dos
245  243,0505,8,0b,00,00,00,00,00,00,Dos
```

Labeled data for training

Progress Review

Model and performance



Attack	Model	Precision	Recall	F1	FNR
DoS	GIDS [30]	-	99.9	-	-
	DCNN [13]	100	99.89	99.95	0.13
	MLIDS [36]	99.9	100	99.9	-
	G-IDCS [28]	99.81	98.86	99.33	-
	TAN-IDS [29]	100	100	100	-
	HyDL-IDS [33]	100	100	100	0
	NovelADS [32]	99.97	99.91	99.94	-
	TCAN-IDS [31]	100	99.97	99.98	-
	iForest [35]	-	-	-	-
	GRU [34]	99.93	99.91	99.92	-
	CQMLP-IDS	99.92	99.88	99.90	0.11
Fuzzing	GIDS [30]	-	98.7	-	-
	DCNN [13]	99.95	99.65	99.80	0.5
	MLIDS [36]	99.9	99.9	99.9	-
	G-IDCS [28]	99.71	99	99.35	-
	TAN-IDS [29]	99.99	99.99	99.99	-
	HyDL-IDS [33]	99.98	99.88	99.93	
	NovelADS [32]	99.99	100	100	-
	TCAN-IDS [31]	99.96	99.89	99.22	-
	iForest [35]	95.07	99.93	97.44	-
	GRU [34]	99.32	99.13	99.22	-
	CQMLP-IDS	99.93	99.69	99.81	0.27
RPM-Spoof	GIDS [30]	-	99.6	-	-
	DCNN [13]	99.99	99.94	99.96	0.05
	MLIDS [36]	100	100	100	-
	G-IDCS [28]	99.85	98.69	99.27	-
	TAN-IDS [29]	99.99	99.93	99.96	-
	HyDL-IDS [33]	100	100	100	0
	NovelADS [32]	99.9	99.9	99.9	-
	TCAN-IDS [31]	99.9	99.9	99.9	-
	iForest [35]	98.9	100	99.4	-
	CQMLP-IDS	99.96	100	99.98	0

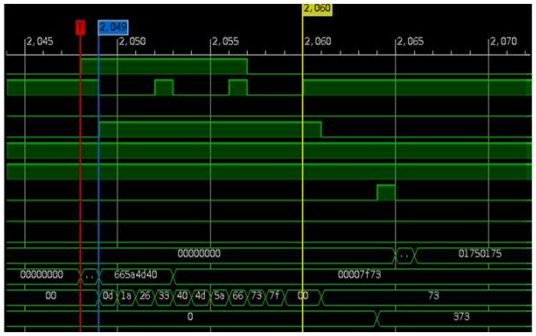
Integrate Mature
Flexible & Deployable & High-performance
CAN bus Attack detection model [1]

[1] Shashwat Khandelwal and Shanker Shreejith. Exploring highly quantised neural networks for intrusion detection in automotive can. In 2023 33rd International Conference on Field-Programmable Logic and Applications (FPL), pages 235–241. IEEE, 2023

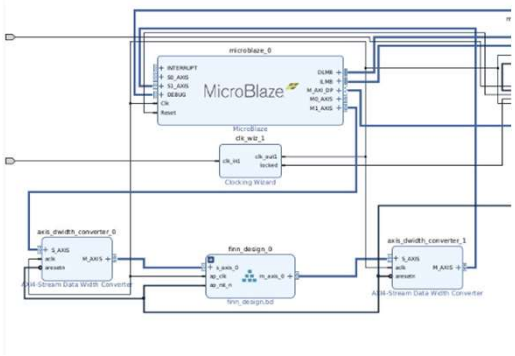
Progress Review

Model integration and testing

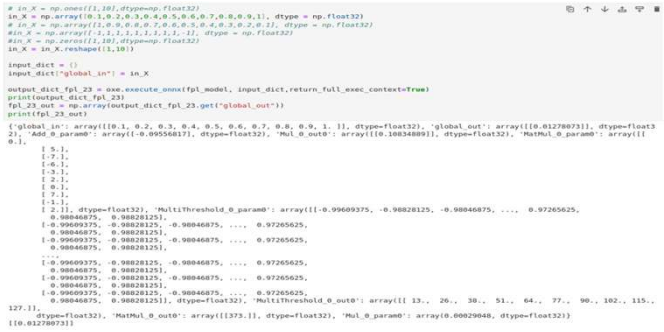
FINN Flow for traditional FPGA Not only for ZYNQ series
Also for traditional light-weight platforms



MLP Model IP core ILA data capture



Microblaze calls FINN IP via AXI-Stream



```
# in_X = np.ones((1,10),dtype=np.float32)
in_X = np.array([0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1], dtype = np.float32)
# in_X = np.array([1,0.9,0.8,0.7,0.6,0.5,0.4,0.3,0.2,0.1], dtype = np.float32)
#in_X = np.array([-1,1,1,1,1,1,1,1,1,-1], dtype = np.float32)
#in_X = np.zeros((1,10),dtype=np.float32)
in_X = in_X.reshape(1,10)

input_dict = {}
input_dict["global_in"] = in_X

output_dict_fpl_23 = oxe.execute_onnx(fpl_model, input_dict,return_full_exec_context=True)
print(output_dict_fpl_23)
fpl_23_out = np.array(output_dict_fpl_23.get("global_out"))
print(fpl_23_out)
```

```
{'global_in': array([[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]], dtype=float32), 'global_out': array([[0.012780731]], dtype=float32), 'Add_0_param0': array([-0.09556817], dtype=float32), 'Mul_0_out0': array([[0.10834889]], dtype=float32), 'MatMul_0_param0': array([[
0.],
       [ 5.],
       [-7.],
       [-6.],
       [-3.],
       [ 2.],
       [ 0.],
       [ 7.],
       [-1.],
       [ 2.]], dtype=float32), 'MultiThreshold_0_param0': array([[-0.9969375, -0.9882812, -0.9884687, ...,  0.9726562,
        0.9884687,  0.9882812],
       [-0.9969375, -0.9882812, -0.9884687, ...,  0.9726562,
        0.9884687,  0.9882812],
       [-0.9969375, -0.9882812, -0.9884687, ...,  0.9726562,
        0.9884687,  0.9882812],
       ...,
       [-0.9969375, -0.9882812, -0.9884687, ...,  0.9726562,
        0.9884687,  0.9882812],
       [-0.9969375, -0.9882812, -0.9884687, ...,  0.9726562,
        0.9884687,  0.9882812],
       [-0.9969375, -0.9882812, -0.9884687, ...,  0.9726562,
        0.9884687,  0.9882812]], dtype=float32), 'MultiThreshold_0_out0': array([[ 13.,  26.,  38.,  51.,  64.,  77.,  90., 102., 115.,
127.]],
      dtype=float32), 'MatMul_0_out0': array([[373.]], dtype=float32), 'Mul_0_param0': array(0.00029048, dtype=float32)}
[[0.012780731]]
```

Comparison of consistency of model software running results

Progress Review

JLR Testing Tech Week - Share Fair

Participate in Jaguar Land Rover testing technology sharing fair and demonstrated the project

Progress Review

Milestone modification

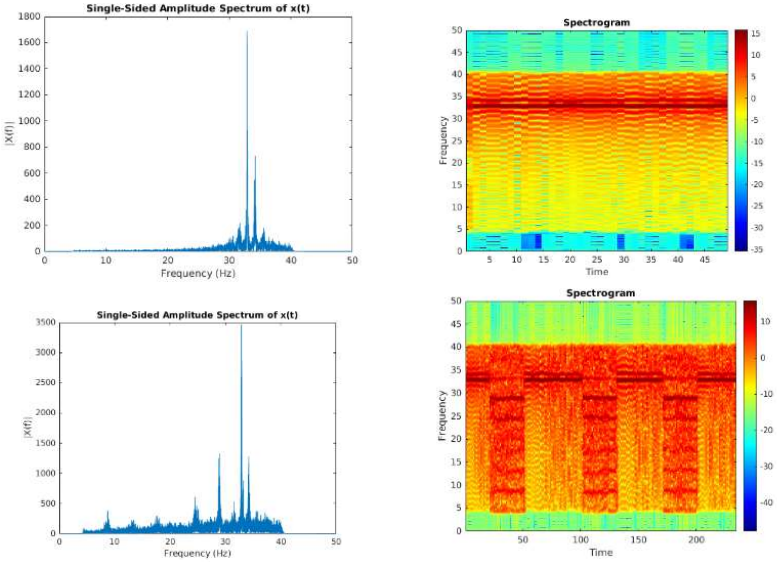
- **Original plan:** Using **Risc-V rocket** soft core simulation ECU
- **Plan after the change:** Using **Microblaze** as ECU soft core
- **Reason for change:** Microblaze is more suitable for Xilinx ecosystem, which is **convenient for debugging and driver** coding. **Microblaze** is gradually beginning to **support Risc-V** instruction set.

- **Original plan**: Adopt the standard FINN integrated flow integrated attack detection core
- **Plan after the change:** Slightly behind schedule with custom build flows that can target both lightweight and traditional FPGAs
- **Reason for change:** Makes the project more versatile, but the custom workflow is slightly more complicated than the standard integration process

Functions such as automated attack injection, data export, and attack detection IP core hardware deployment were not originally planned, but were added for engineering integrity and usability.

Plan for the rest of the project

Next step



Testing and comparison of other methods (like using STFT with DNN for attack detection)

- Model accuracy performance optimization evaluation
- Resource Utilization optimizing
- Model ablation experiment, quantization accuracy change experiment
- Acceleration test

Model benchmarking and optimizing

Of course, there is also thesis writing, refer to the previous Gantt chart plan

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Thank You

Changhong Li
Supervisor: Shreejith Shanker

Date 23/05/2024