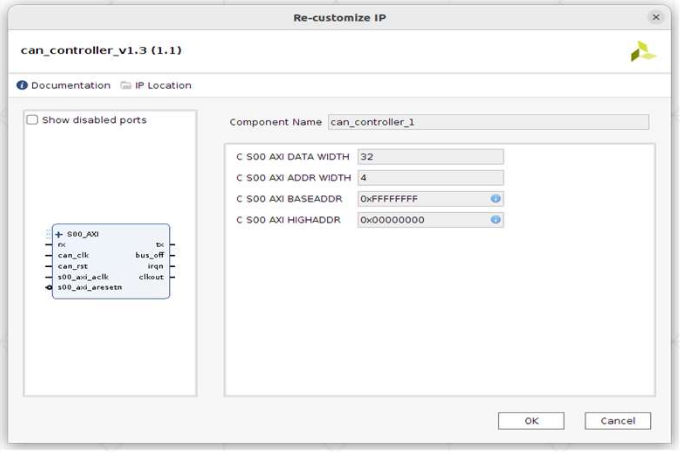
Hardware In Loop Project Report

M3W1

Current Progress

Work	Description	Status
Build CAN Controller IP	pack the can controller to an AXI IP core and build its driver	Finished, could be called by microblaze or risc-v core
Build CAN Network	Build a CAN network with multiple ECUs and a control node	Finished, 3 ECU(based on microblaze) work together on one Nexys Board connected by CAN bus
Build ECU basic logic	build some basic logic of ECUS	Finished, ECUs could interact with physical switches and leds to simulate real devices and communicate with CAN messages
Build Control program on PC	build an UI program to control the HWIL process , monitor status and execute test	Finished, based on PyQt, could monitor and send command to the Control node and ECUs
Implement Dos, Replay and Spoofing Attack	Use the user interface to inject attacks to the CAN network	In progress, could inject Spoofing and Replay attack to the CAN network

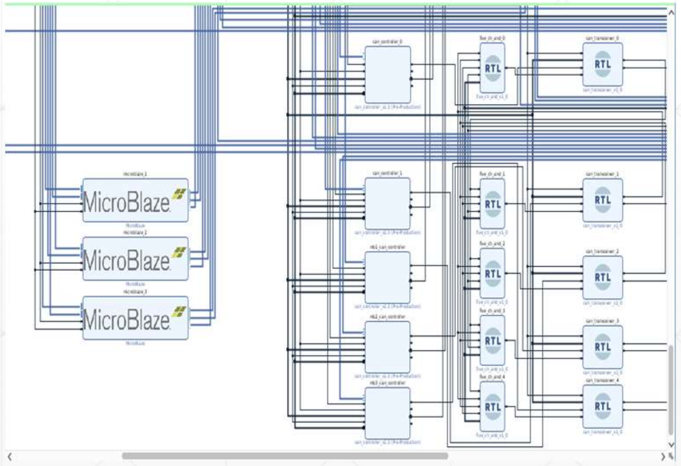
Current Progress - CAN Control IP and Network


Re-customize IP

can_controller_v1.3 (1.1)

Component Name: can_controller_1

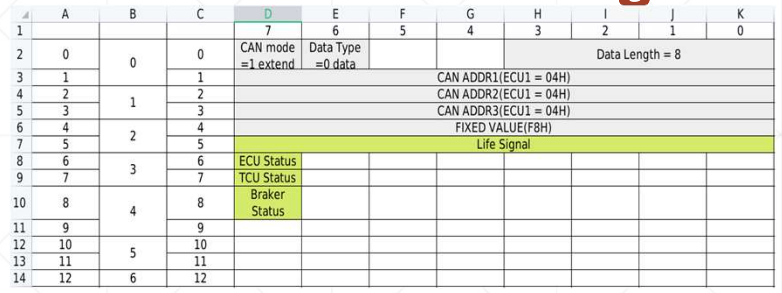
C S00 AXI DATA WIDTH	32
C S00 AXI ADDR WIDTH	4
C S00 AXI BASEADDR	0xFFFFFFFF
C S00 AXI HIGHADDR	0x00000000



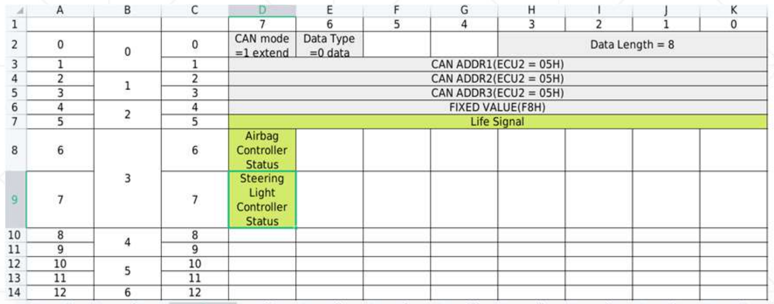
Pack the can controller to an AXI IP core that could be used by microblaze the risc-v core instead of using gpio control.

Build the On board CAN bus network, could be extended with physical CAN network or other simulation instance boards, include risc-v based ECU controller

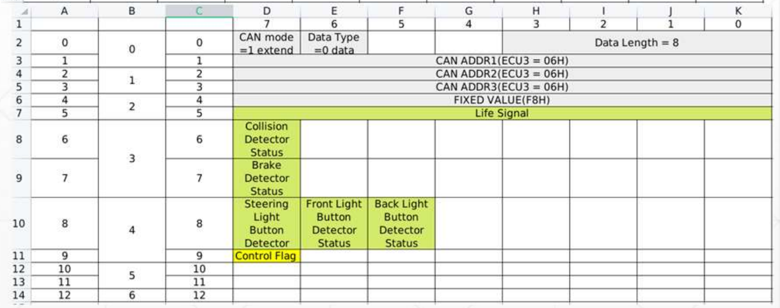
Current Progress - ECU logics



	A	B	C	D	E	F	G	H	I	J	K
1				7	6	5	4	3	2	1	0
2	0	0	0	CAN mode =1 extend	Data Type =0 data			Data Length = 8			
3	1		1	CAN ADDR1(ECU1 = 04H)							
4	2	1	2	CAN ADDR2(ECU1 = 04H)							
5	3		3	CAN ADDR3(ECU1 = 04H)							
6	4	2	4	FIXED VALUE(F8H)							
7	5		5	Life Signal							
8	6	3	6	ECU Status							
9	7		7	TCU Status							
10	8	4	8	Braker Status							
11	9		9								
12	10	5	10								
13	11		11								
14	12	6	12								



	A	B	C	D	E	F	G	H	I	J	K
1				7	6	5	4	3	2	1	0
2	0	0	0	CAN mode =1 extend	Data Type =0 data			Data Length = 8			
3	1		1	CAN ADDR1(ECU2 = 05H)							
4	2	1	2	CAN ADDR2(ECU2 = 05H)							
5	3		3	CAN ADDR3(ECU2 = 05H)							
6	4	2	4	FIXED VALUE(F8H)							
7	5		5	Life Signal							
8	6	3	6	Airbag Controller Status							
9	7		7	Steering Light Controller Status							
10	8	4	8								
11	9		9								
12	10	5	10								
13	11		11								
14	12	6	12								

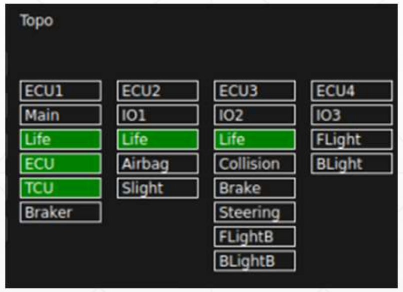


	A	B	C	D	E	F	G	H	I	J	K
1				7	6	5	4	3	2	1	0
2	0	0	0	CAN mode =1 extend	Data Type =0 data			Data Length = 8			
3	1		1	CAN ADDR1(ECU3 = 06H)							
4	2	1	2	CAN ADDR2(ECU3 = 06H)							
5	3		3	CAN ADDR3(ECU3 = 06H)							
6	4	2	4	FIXED VALUE(F8H)							
7	5		5	Life Signal							
8	6	3	6	Collision Detector Status							
9	7		7	Brake Detector Status							
10	8	4	8	Steering Light Button Detector	Front Light Button Detector Status	Back Light Button Detector Status					
11	9		9	Control Flag							
12	10	5	10								
13	11		11								
14	12	6	12								

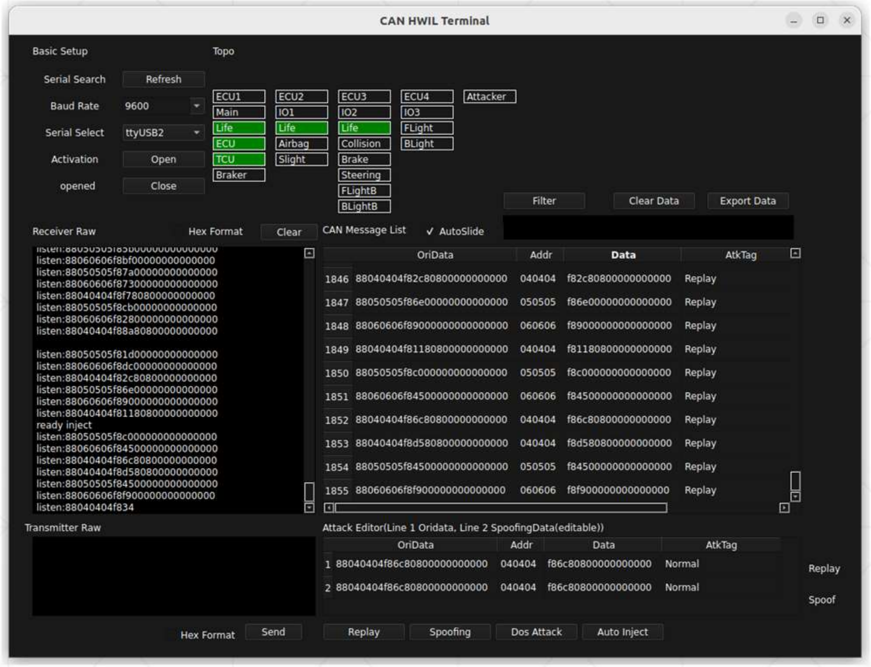
Build up 3 ECUs with different logic for the test.
All ECUs: Generate Life signal to indicate that they are alive.
periodically send status messages to report their status.

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.



Current Progress - User Interface



Baud Rate: 9600

Serial Select: ttyUSB2

opened

```
listen:88060606f8bf0000000000000
listen:88050505f87a0000000000000
listen:88060606f8730000000000000
listen:88040404f8f78080000000000
listen:88050505f8cb0000000000000
listen:88060606f8280000000000000
listen:88040404f88a8080000000000

listen:88050505f81d0000000000000
listen:88060606f8dc0000000000000
listen:88040404f82c8080000000000
listen:88050505f86e0000000000000
listen:88060606f8900000000000000
listen:88040404f8118080000000000
ready inject
listen:88050505f8c00000000000000
listen:88060606f8450000000000000
listen:88040404f86c8080000000000
listen:88040404f8d58080000000000
listen:88050505f8450000000000000
listen:88060606f8f90000000000000
listen:88040404f834
```

CAN Message List

	OriData	Addr	Data	AtkTag
1846	88040404f82c80800000000000	040404	f82c80800000000000	Replay
1847	88050505f86e00000000000000	050505	f86e00000000000000	Replay
1848	88060606f89000000000000000	060606	f89000000000000000	Replay
1849	88040404f81180800000000000	040404	f81180800000000000	Replay
1850	88050505f8c000000000000000	050505	f8c000000000000000	Replay
1851	88060606f84500000000000000	060606	f84500000000000000	Replay
1852	88040404f86c80800000000000	040404	f86c80800000000000	Replay
1853	88040404f8d580800000000000	040404	f8d580800000000000	Replay
1854	88050505f84500000000000000	050505	f84500000000000000	Replay
1855	88060606f8f900000000000000	060606	f8f900000000000000	Replay

Attack Editor(Line 1 Oridata, Line 2 SpoofingData(editable))

	OriData	Addr	Data	AtkTag
1	88040404f86c80800000000000	040404	f86c80800000000000	Normal
2	88040404f86c80800000000000	040404	f86c80800000000000	Normal

GUI Test terminal built by PyQt, could implement the functions as below:
1. Monitor and analysis the status of each ECU
2. Monitor and send raw data to the CAN bus
3. Excute Replay, Spoofing and Dos Attack(Dos Attack in under developing) and mark the messages with attack tags
4. Record and export the CAN messages to csv files for model training.

Next Step

Work	Description	Deadline
Dos Attack	Implement dos attack with GUI command	3/22
Auto Injection	GUI could accept injection test sequence and execute them automatically and periodically	3/29
Object code download	GUI could download the elf file to the ECUs	4/5
Data Sorting	Sort the data generated by the auto injection	4/5
Model Training	Train the model to detect the injection(Existing dataset[1] and generated dataset, generated dataset for flexible specific and reconfigurable application scenarios)	4/15
Interface expansion(To be discussed)	Extend the interface to be compatible for physical can bus interface and other boards with risc-v cores	4/30

[1] Car Hacking dataset mentioned in "Exploring Highly Quantised Neural Networks for Intrusion Detection in Automotive CAN" from "CAR Hacking dataset, "https://ocslab.hksecurity.net/datasets/can-intrusion-dataset," 2020."

Needed Support

Support	Description
Dos Attack method	Regarding the DoS attack implemented by a message mentioned earlier, could you please provide me more relevant reference materials?
Advice on model training	For model training, I am thinking about this CAN bus hardware-in-the-loop simulation network that can quickly expand configuration changes. This application can quickly establish an attack diagnosis model adapted to the established network. Should we produce data sets based on this idea or still use existing data sets as we mentioned before. Or should I do both?
Advice on interface expansion	Regarding the interface, in order to meet the scalability of the simulation environment, including adapting to the Risc-V core or actual CAN devices, do we need to expand the physical CAN interface, or expand another FPGA with Risc-V (as I mentioned before Yes, it is challenging to implement the entire project in a short time using Risc-V without BSP, but it is possible to make it only serve as an ECU node and download the application, but it needs to be on another board. Because jtag will conflict with microblaze)
Ideas about model hardware deployment	If we have time, could we consider deploying the model to a hardware environment? Referring to previous work, maybe we can use HLS to build the basic neural network layer, or is there a better solution?(Of course, if time permits)