Proyecto Diseño Lógico 2

CAN Opener

Integrantes: José Bentancour
Damián Vallejo

Tutor: Leonardo Etcheverry

Introducción:

- El proyecto se basa en la implementación de un “sniffer” de una red CAN de un automóvil
- CAN es un protocolo de comunicación utilizado ampliamente en la industria automotriz desarrollado por la empresa alemana Bosch
- Mediante este protocolo la CPU (ECU) del auto se comunica con los varios periféricos del mismo (tablero, luces, sensores, etc.)

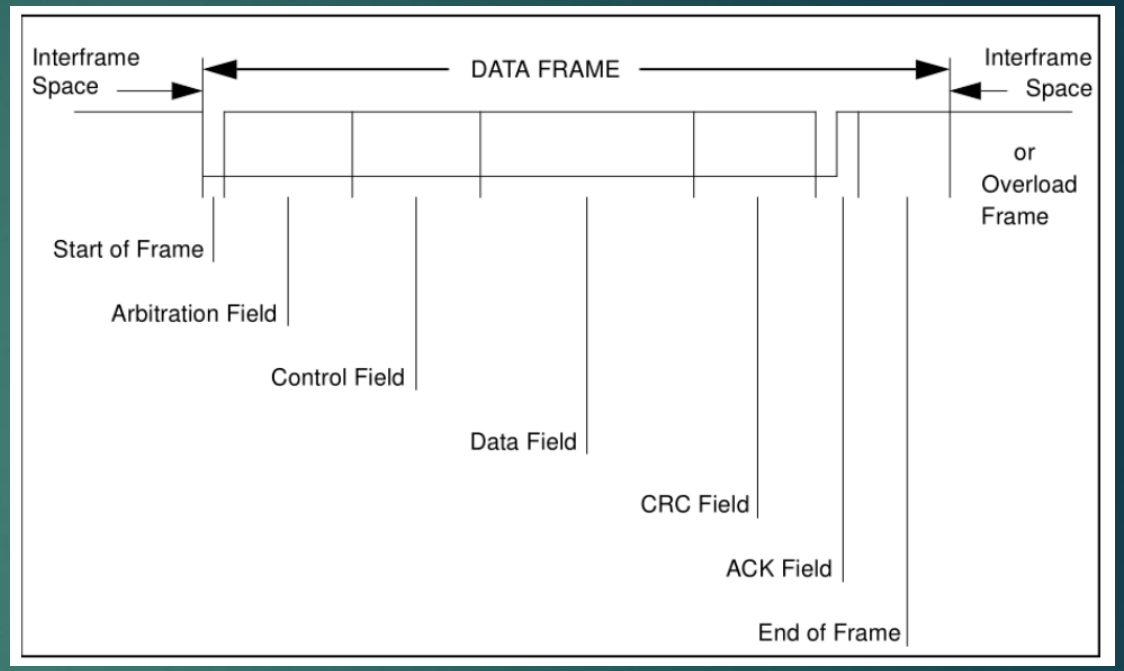
Objetivos:

- Implementar un receptor CAN en VHDL
- Implementar un programa en el procesador NIOS II de la placa DE0 y visualizar los mensajes recibidos en una consola en una computadora
- Establecer la comunicación entre las dos etapas a través de una interfaz Avalon

Receptor CAN:

Mensaje CAN

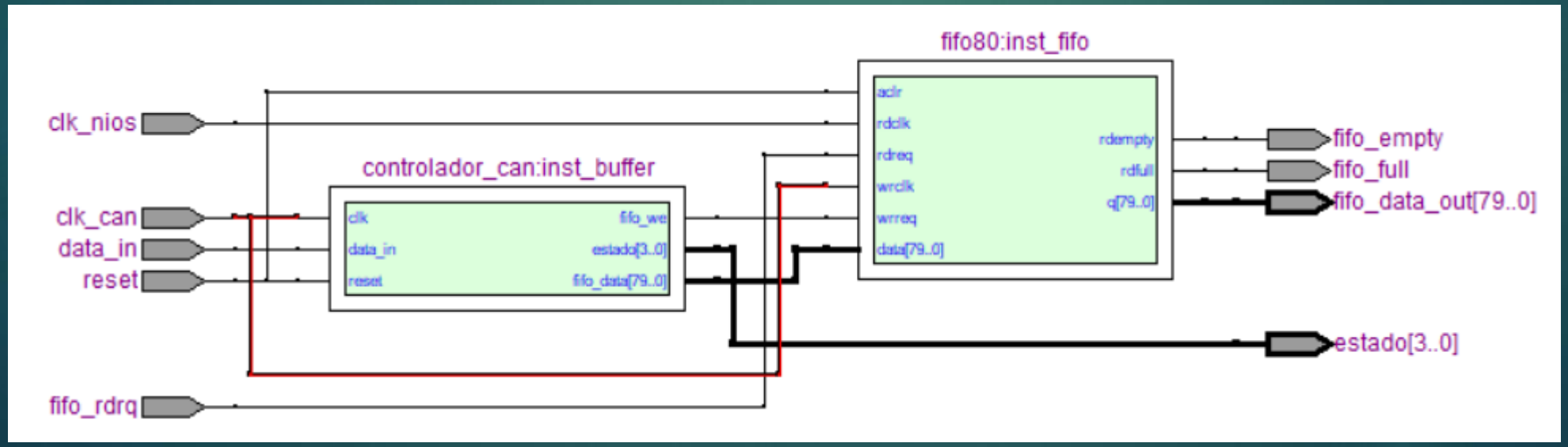
• Un mensaje CAN está constituido por distintas partes o “frames”
• Es un protocolo “activo por nivel bajo”, esto es que al no haber actividad en el bus, el mismo mantiene una señal igual a “1”
• El mensaje está compuesto por un “0” inicial (start of frame), un campo de arbitraje, control, payload, CRC, acuso de recibo y end of frame.
• Para evitar errores no se permite más de 5 bits con la misma polaridad y se rellena forzosamente con un bit de polaridad opuesta (bit stuffing) los campos del mensaje (salvo el acuso de recibo y end of frame)



Receptor CAN:

Implementación

- Se crea una máquina de estados con un estado por frame que guarda en un shift register por estado el contenido del frame
- Se verifica el CRC y se guarda en una bandera su correctitud
- Se identifican y desechan los bits de stuffing, además de comprobar su correcta polaridad
- Al terminar el mensaje se guarda en un registro FIFO la información de interés (arbitraje, largo de data, payload y correctitud de CRC)



Interfaz Avalon:

Implementación

• Se utiliza una interfaz Avalon Memory Mapped Interface (Avalon-MM), la cual permite lectura y escritura basada en direcciones, es una interfaz típica Maestro-Esclavo
• Esta interfaz mapea los datos recibidos y los estados del FIFO y receptor CAN en direcciones de memoria
• Como el bus de datos del procesador NIOS es de 32 bits se utilizaron 3 direcciones para transmitir los datos

Offset	Byte3	Byte2	Byte2	Byte0
0x00	Status	DataLen	AddresH	AddresL
0x01	Data3	Data2	Data1	Data0
0x02	Data7	Data6	Data5	Data4

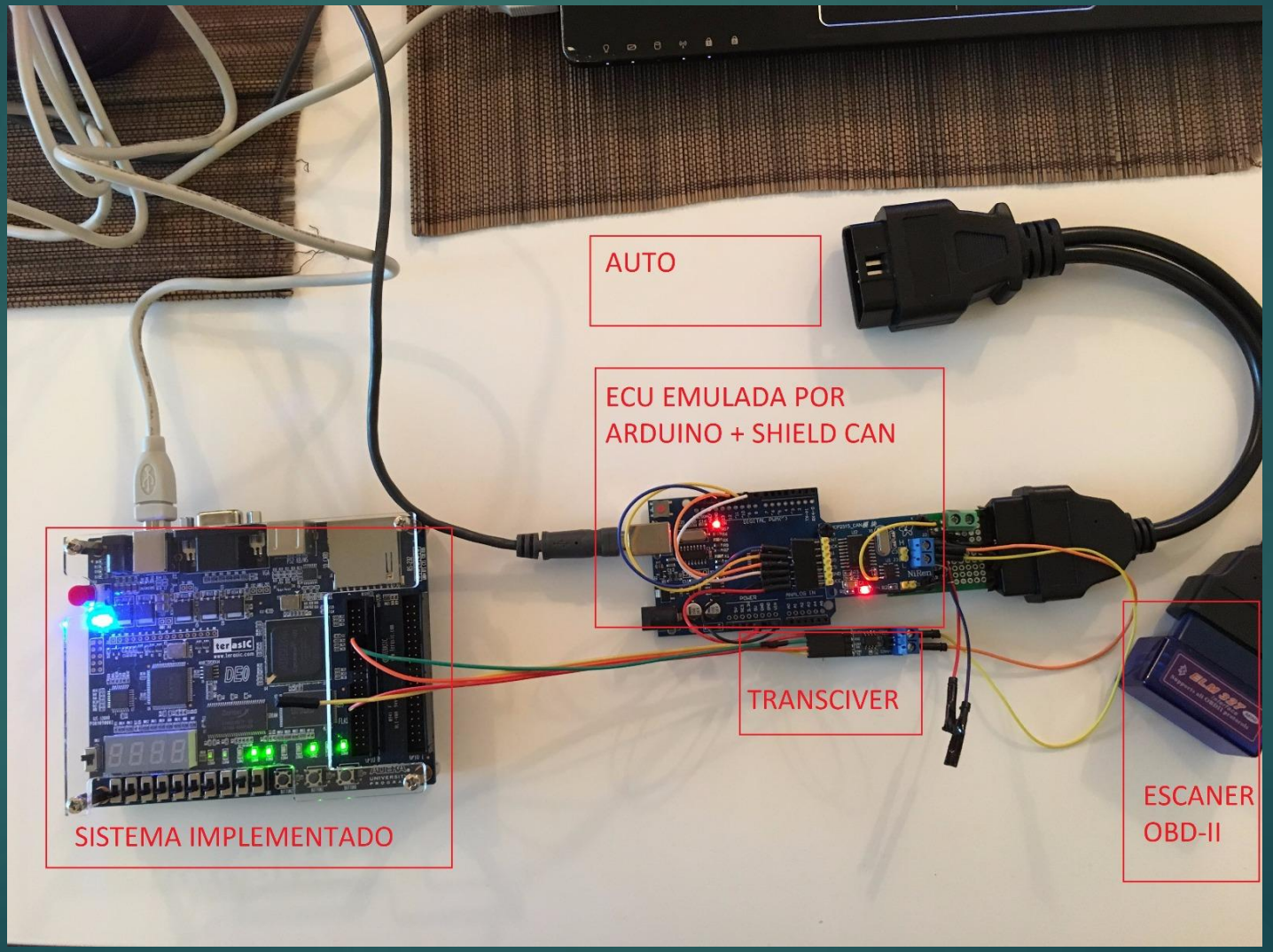
SoC:

Implementación

• Se utiliza el microprocesador NIOS II de la placa DE0 corriendo el software que permite la visualización de datos
• OnChip memory memoria RAM y ROM donde se almacena y corre la aplicación
• Parallel IO interface para manejar los LEDs de la placa para debugueo
• JTAG UART interface para la comunicación con la computadora a través de USB Blaster
• El software que corre en el microprocesador comprueba constantemente la presencia de nuevos datos recibidos e imprime en la consola su contenido

⊟ **cpu**	Nios II Processor				
instruction_master	Avalon Memory Mapped Master	**clk_0**			
data_master	Avalon Memory Mapped Master			IRQ 0	IRQ 31
jtag_debug_module	Avalon Memory Mapped Slave			**0x00004800**	0x00004fff
⊟ **jtag_uart**	JTAG UART				
avalon_jtag_slave	Avalon Memory Mapped Slave	**clk_0**		**0x00005020**	0x00005027
⊟ **onchip_memory2**	On-Chip Memory (RAM or ROM)				
s1	Avalon Memory Mapped Slave	**clk_0**		**0x00002000**	0x00003fff
⊟ **pio_led**	PIO (Parallel I/O)				
s1	Avalon Memory Mapped Slave	**clk_0**		**0x00005000**	0x0000500f
⊟ **can_opener_inst**	can_opener				
slave	Avalon Memory Mapped Slave	**clk_0**		**0x00005010**	0x0000501f

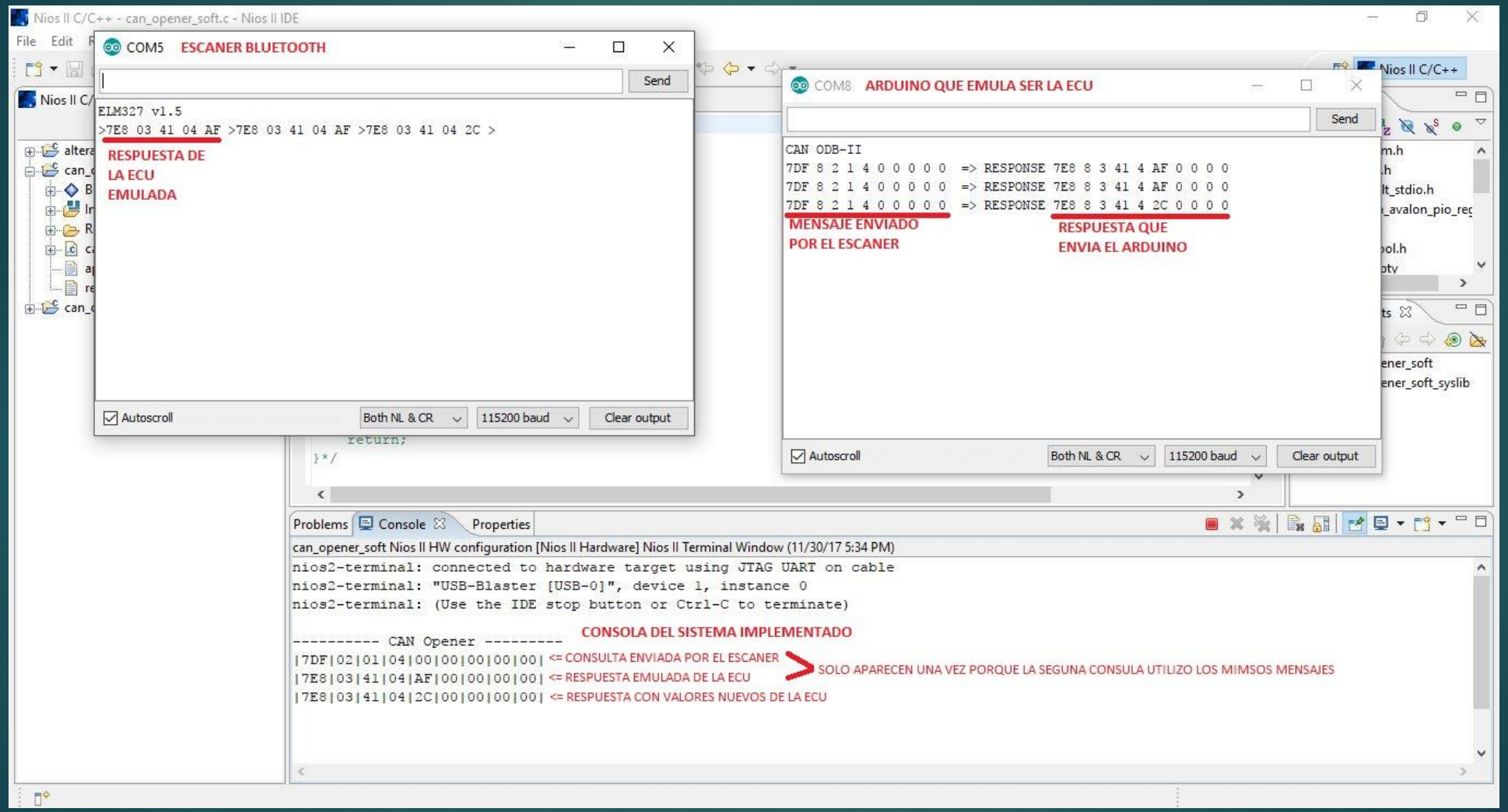
Pruebas:


AUTO
ECU EMULADA POR
ARDUINO + SHIELD CAN
TRANSCIVER
SISTEMA IMPLEMENTADO
ESCANER
OBD-II

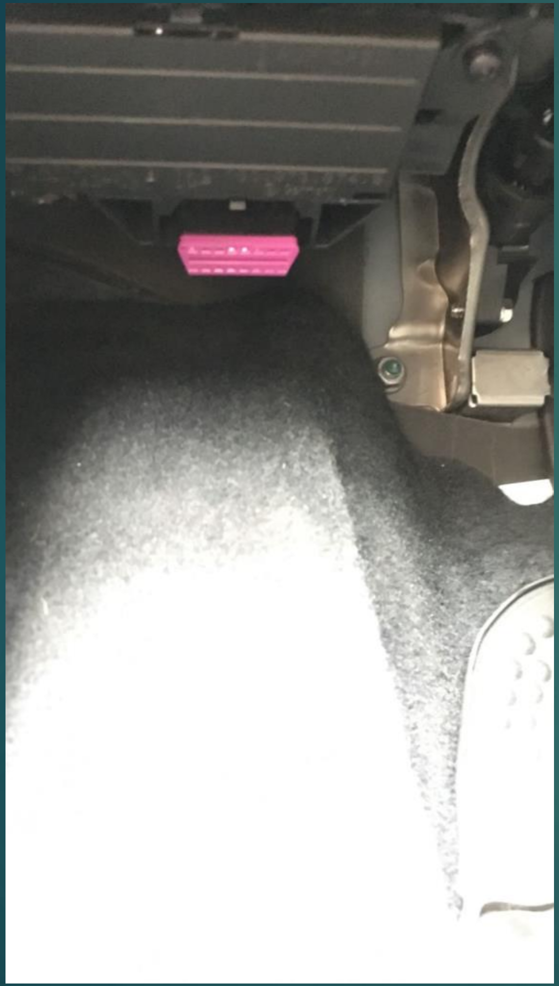
Pruebas:

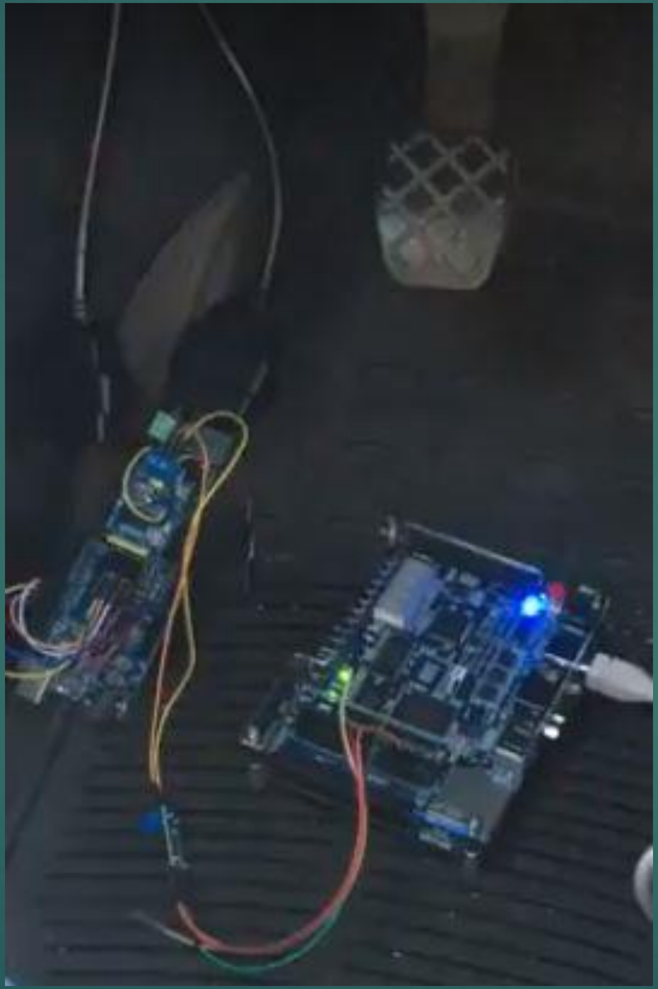
- Para el desarrollo y validación del bloque VHDL se utilizaron formas de onda (.vwf) conteniendo un mensaje fijo en el entorno de desarrollo Quartus
- Para la validación del sistema completo se realizaron varias pruebas:
  - Se creó un bloque VHDL que inyecta una señal con un mensaje conocido de manera cíclica en la entrada can_rx
  - Se conectó el escáner ELM327 y configuró el MCP2515 en modo recepción y verificó que el sistema recibiese el mensaje
  - Con el escáner desconectado y el MCP2515 en modo transmisión constante se verificó que el sistema recibiese los mensajes
  - Finalmente se conecta el sistema al puerto CAN de un auto y se verifica la recepción de mensajes

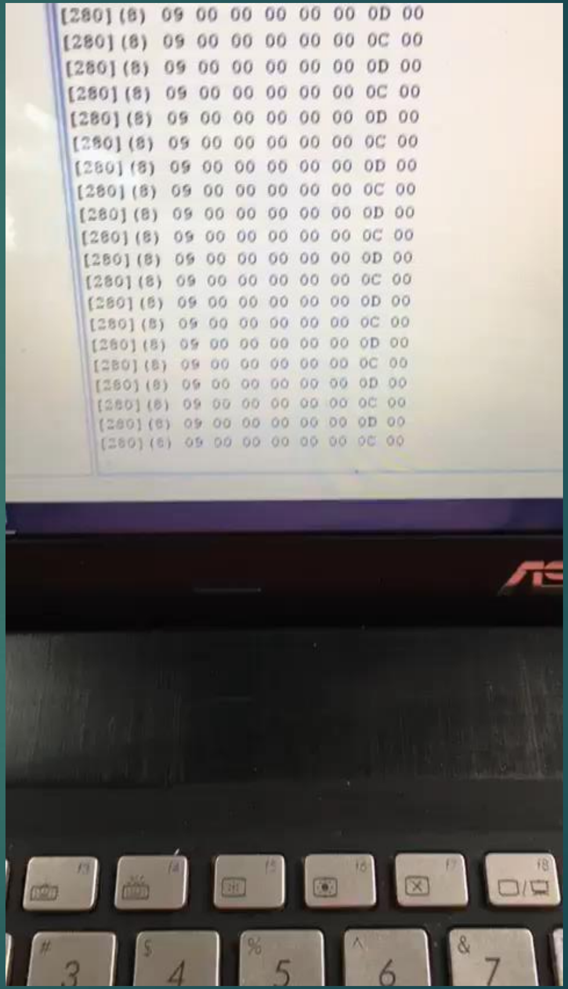
Pruebas:


COM5 ESCANER BLUETOOTH
Send
ELM327 v1.5
>7E8 03 41 04 AF >7E8 03 41 04 AF >7E8 03 41 04 2C >
RESPUESTA DE
LA ECU
EMULADA
Autoscroll
Both NL & CR
115200 baud
Clear output
COM8 ARDUINO QUE EMULA SER LA ECU
Send
CAN ODB-II
7DF 8 2 1 4 0 0 0 0 0 => RESPONSE 7E8 8 3 41 4 AF 0 0 0 0
7DF 8 2 1 4 0 0 0 0 0 => RESPONSE 7E8 8 3 41 4 AF 0 0 0 0
7DF 8 2 1 4 0 0 0 0 0 => RESPONSE 7E8 8 3 41 4 2C 0 0 0 0
MENSAJE ENVIADO
POR EL ESCANER
RESPUESTA QUE
ENVIA EL ARDUINO
Autoscroll
Both NL & CR
115200 baud
Clear output
Problems
Console
Properties
can_opener_soft Nios II HW configuration [Nios II Hardware] Nios II Terminal Window (11/30/17 5:34 PM)
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)
CONSOLA DEL SISTEMA IMPLEMENTADO
---------- CAN Opener ---------
|7DF|02|01|04|00|00|00|00|00| <= CONSULTA ENVIADA POR EL ESCANER
|7E8|03|41|04|AF|00|00|00|00| <= RESPUESTA EMULADA DE LA ECU
|7E8|03|41|04|2C|00|00|00|00| <= RESPUESTA CON VALORES NUEVOS DE LA ECU
SOLO APARECEN UNA VEZ PORQUE LA SEGUNA CONSULA UTILIZO LOS MIMSOS MENSAJES

Pruebas:






[280] (8) 09 00 00 00 00 00 0D 00
[280] (8) 09 00 00 00 00 00 0C 00
[280] (8) 09 00 00 00 00 00 0D 00
[280] (8) 09 00 00 00 00 00 0C 00
[280] (8) 09 00 00 00 00 00 0D 00
[280] (8) 09 00 00 00 00 00 0C 00
[280] (8) 09 00 00 00 00 00 0D 00
[280] (8) 09 00 00 00 00 00 0C 00
[280] (8) 09 00 00 00 00 00 0D 00
[280] (8) 09 00 00 00 00 00 0C 00
[280] (8) 09 00 00 00 00 00 0D 00
[280] (8) 09 00 00 00 00 00 0C 00
[280] (8) 09 00 00 00 00 00 0D 00
[280] (8) 09 00 00 00 00 00 0C 00
[280] (8) 09 00 00 00 00 00 0D 00
[280] (8) 09 00 00 00 00 00 0C 00
[280] (8) 09 00 00 00 00 00 0D 00
[280] (8) 09 00 00 00 00 00 0C 00
[280] (8) 09 00 00 00 00 00 0D 00
[280] (8) 09 00 00 00 00 00 0C 00

Conclusiones:

- Se implementó un receptor CAN con verificación de CRC en VHDL
- Se implementó una interfaz Avalon Memory Mapped para comunicar el bloque VHDL con el procesador NIOS II de la placa DE0
- Se lograron observar distintos mensajes en la consola y el cambio de los mismos al realizar diferentes acciones en un auto (accionar el acelerador, prender luces, abrir puertas, etc.)

Aspectos a mejorar:

- Sincronización del reloj del bloque VHDL con el inicio de los mensajes CAN
- Transmisión de datos
- Configuración de parámetros en tiempo de ejecución

Gracias