EMBEDDED SYSTEMS

BASED ON CORTEX-M4 AND THE RENESAS SYNERGY PLATFORM

2020

PROF. DOUGLAS RENAUX, PHD
PROF. ROBSON LINHARES, DR.
UTFPR / ESYSTECH

RENESAS ELECTRONICS CORPORATION

BIG IDEAS FOR EVERY SPACE

9 – CAN

- Introduction
- Block Diagram
- Registers
- SW Stack

9.1 – INTRODUCTION

CAN is an acronym for Controller Area Network. It is defined by the ISO-11808: 2003 standard and has been mainly motivated by the needs of the automotive industry, such as the ever increasing use of embedded sensors into the vehicles and the need to optimize the internal space and reduce costs with cabling.

Characteristics of CAN:

- Two-wire multi-master serial bus
- Message-based protocol
- Contention resolution via decentralized arbitration
- All messages are broadcast and processed by the nodes only if needed
- Speeds up to 1 Mbps

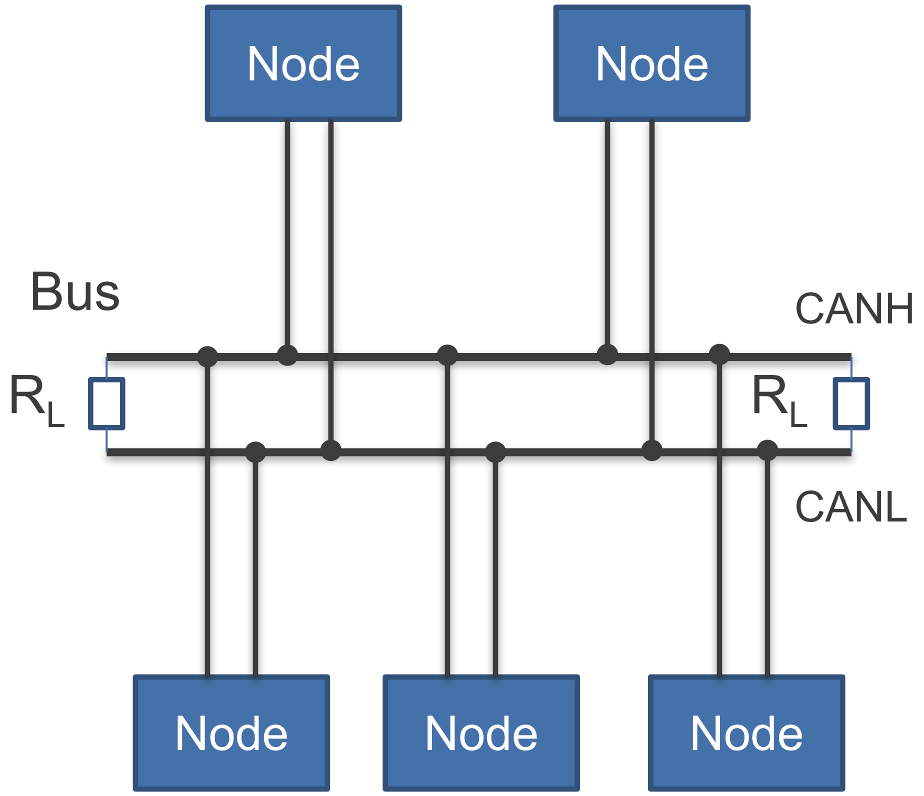
CAN TOPOLOGY

CAN nodes are interconnected in a bus topology.

CAN physical layer is implemented with two wires (CANH and CANL).

A logical 0 (called “dominant”) is obtained when CANH is approx. 3.5V and CANL is approx. 1.5V.

A logical 1 (called “recessive”) is obtained when both CANH and CANL are at approx. 2.5V.

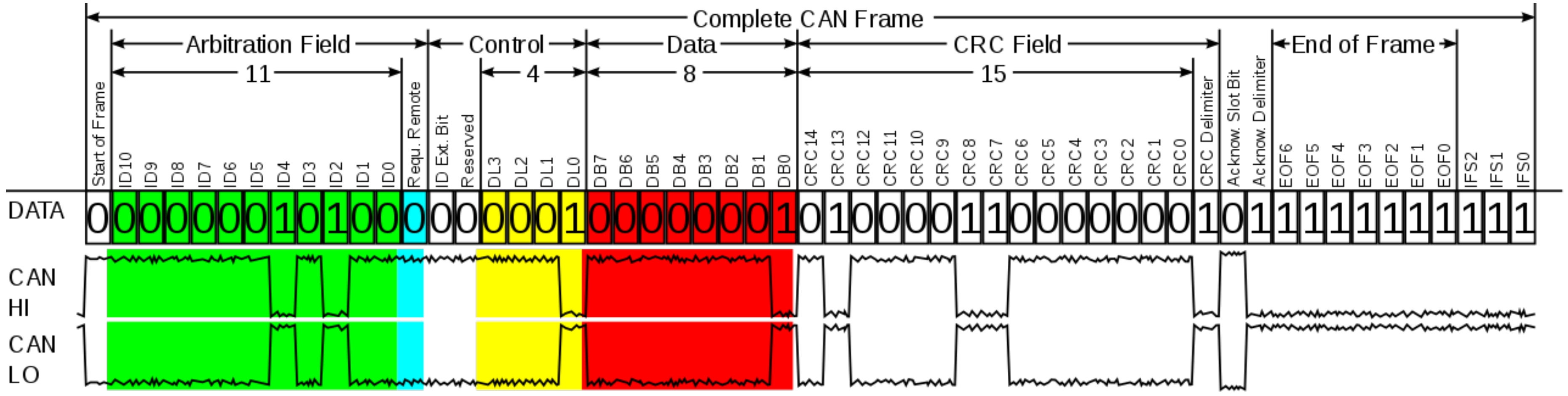


source: Authors

CAN BUS FRAME

The standard CAN frame is defined as follows:

- Arbitration field (Identifier) → defines the message identifier and its priority;
- Data → data to be transmitted (0 to 64 bits);
- SOF, CRC, ACK and End of Frame → error checking and synchronization;
- IFS (Interframe Space) → idle time used to process buffers.



Source: CAN Bus (https://en.wikipedia.org/wiki/CAN_bus)

CAN BUS FRAME (CONT.)

Control → define the following sub-fields:

- Data length (0 to 8 bytes),
- Requ. Request (RTR) → used to identify a Remote Frame → see following slides,
- ID Ext. (IDE) → used to identify a Standard or Extended Frame:
  - Standard Frame → IDE is dominant "0", 11-bit identifier as shown in picture,
  - Extended Frame → IDE is recessive "1", 29-bit identifier. The remaining 18 bits of the identifier are placed right after the IDE bit, followed by an extra RTR bit. The original RTR is called SRR (Substitute Remote Request) and acts as a placeholder.

CAN MESSAGE TYPES

- **Data** Frame → carries data sent by a Node:
  - The RTR bit is dominant “0” to identify a Data Frame;
  - It is always preceded by an Interframe Space.
- **Remote** Frame → carries a request for a transmission of data from another Node:
  - The Arbitration/Identifier field carries the Identifier of the requested Node;
  - The RTR bit is recessive “1” to identify a Remote Frame;
  - The Data field is empty and the Data Length part of Control field determines the length of the requested message;
  - It is always preceded by an Interframe Space.

CAN MESSAGE TYPES

- **Error** frame → special format used to signal an error;
  - Not preceded by an Interframe Space.
- **Overload** frame → special format used to provide an extra delay between messages (receiver too busy);
  - Not preceded by an Interframe Space.

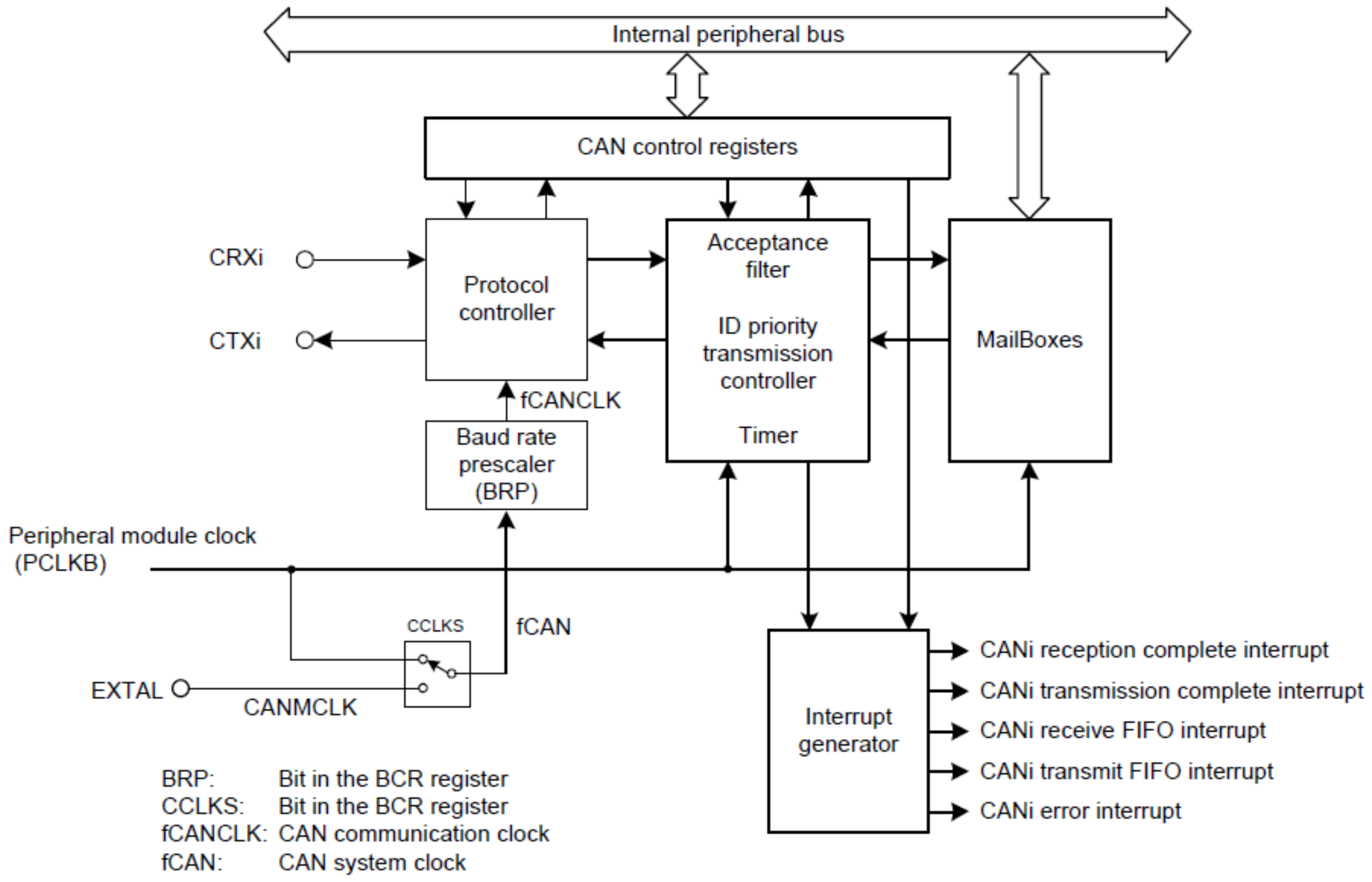
CAN ARBITRATION PROCESS

- Every message has a priority level corresponding to its identifier (arbitration field) → the lower the value, the higher the priority.
- When two or more nodes try to transmit at the same frame time:
  - Identifier bits 0 are "dominant" over identifier bits 1 "recessive";
  - The node that sends a 1 and reads back a 0 stops transmitting on that frame → retries on the next frame;
  - The node that sends a 0 and reads back a 0 retains control and goes on to send the next identifier bit;
  - After all the identifier bits are tested, the node that keeps on retaining control (i. e. the node whose message identifier has the highest priority) sends the message contents.

9.2 – BLOCK DIAGRAM

Implementation for the CAN Module of the S7G2 MCU.

Message reception and transmission organized in *mailboxes* → configurable as single or FIFOs for different types of messages.


Internal peripheral bus
CAN control registers
CRXi
CTXi
Protocol controller
Acceptance filter
ID priority transmission controller
Timer
MailBoxes
fCANCLK
Baud rate prescaler (BRP)
Peripheral module clock (PCLKB)
CCLKS
fCAN
EXTAL
CANMCLK
Interrupt generator
CANi reception complete interrupt
CANi transmission complete interrupt
CANi receive FIFO interrupt
CANi transmit FIFO interrupt
CANi error interrupt
BRP: Bit in the BCR register
CCLKS: Bit in the BCR register
fCANCLK: CAN communication clock
fCAN: CAN system clock

Source: Renesas Synergy MCUs User's Manual: Hardware

9.3 – REGISTERS – CASE STUDY

Implementation for the CAN Module of the R7FS7G27H3A01CFC Renesas ARM Cortex-M4 MCU:

- CTLR → mailbox mode, bus operation and/or reset, timestamp configuration;
- BCR → data transfer rate configuration;
- MKR[0…7] → define masks for reception of specific messages (IDs) into specific mailboxes (depending on MKR index);
- FIDCR[0..1] → similar to MKR but for FIFO mailboxes;
- MKIVLR → enables or disables masking (via MKR) for message acceptance;
- MIER → enable/disable mailbox interrupts;
- MCTL_TX[0..31] → transmission control for each mailbox;
- MCTL_RX[0..31] → reception control for each mailbox;

9.3 – REGISTERS – CASE STUDY

- MB[0..31] → register groups for each mailbox:
  - MB[0..31].ID → received or transmitted message identifiers;
  - MB[0..31].DL → data length;
  - MB[0..31].D[0..7] → received or transmitted data;
  - MB[0..31].TS → timestamp for received messages.
- RFCR, TFCR → receive and transmit FIFO control;
- RFPCR, TFPCR → increment of the CPU-controlled pointer for receive and transmit FIFOs;
- STR → global CAN status register (new data received, receive and transmit FIFO status, CAN mode status and error status);
- EIER → enable / disable error interrupts;
- EIFR → status of error detection.

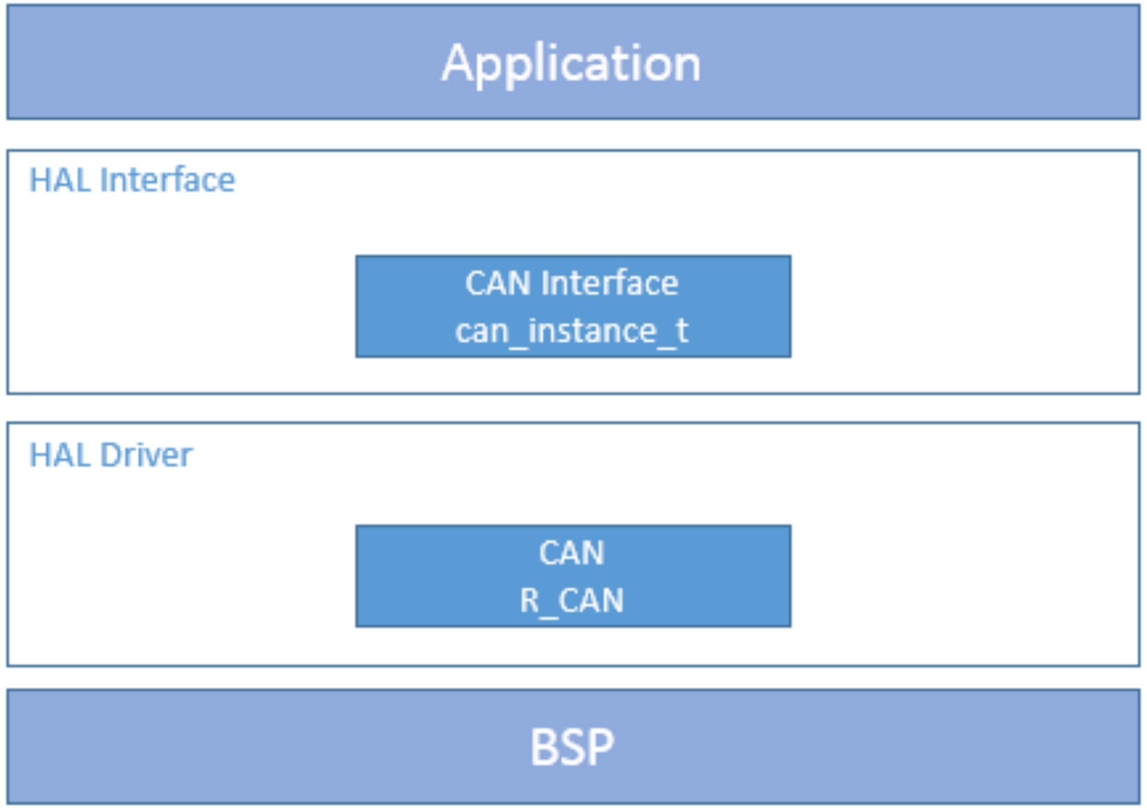
9.3 – REGISTERS – CASE STUDY

- RECR, TECR → receive / transmit error count;
- ECSR → status of CAN bus errors;
- TSR → stores the timestamp;
- TCR → test control.

9.4 – SOFTWARE STACK – CASE STUDY

Example of CAN stack for Renesas Synergy microcontroller hardware.

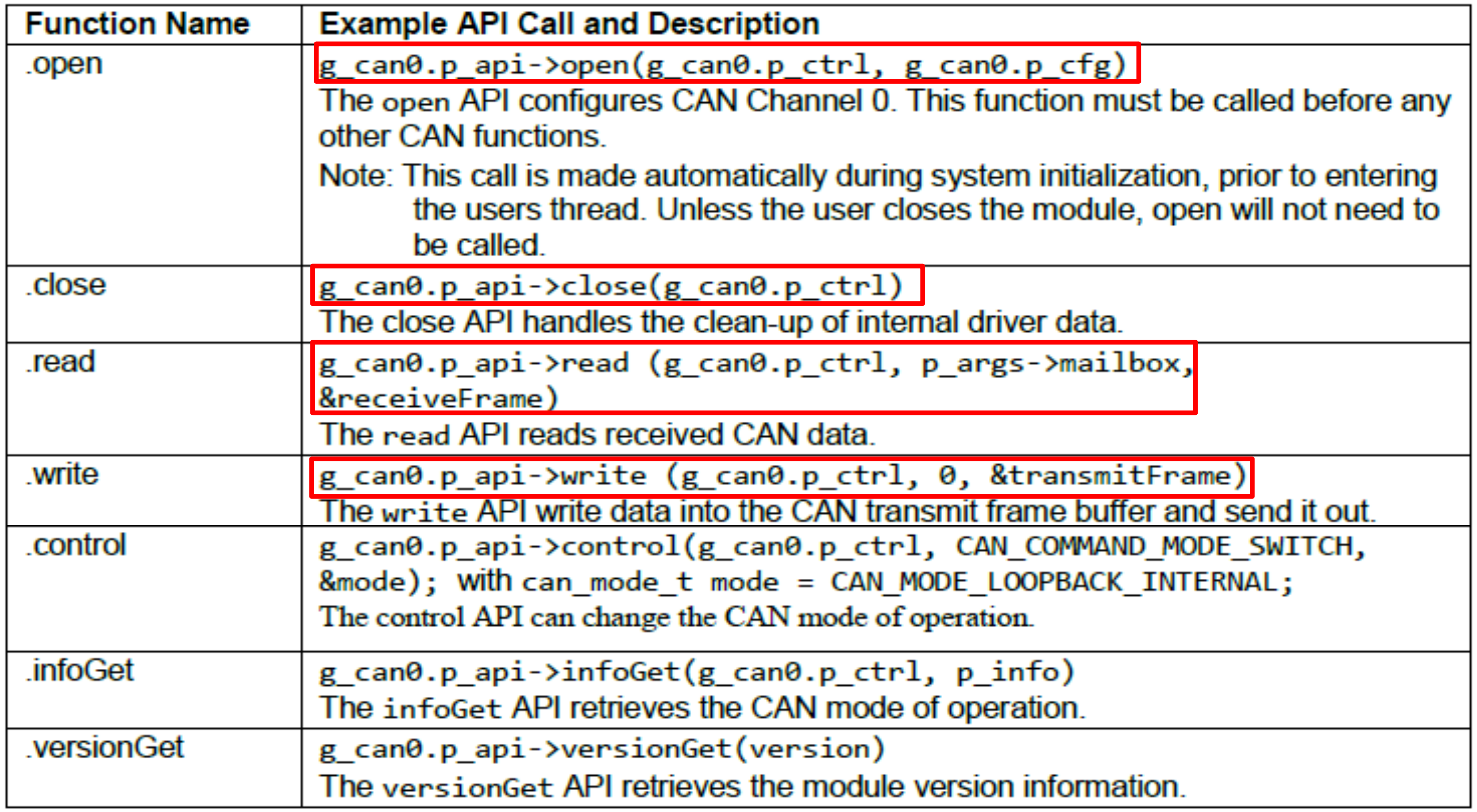
(https://www.renesas.com/en-us/software/D6001427.html)



Source: Renesas Synergy CAN HAL Driver Module Guide
r11an0065eu0101-synergy-can-hal-mod-guide

9.4 – SOFTWARE STACK – CASE STUDY

Example of CAN API for Renesas Synergy microcontroller hardware



Function Name	Example API Call and Description
.open	`g_can0.p_api->open(g_can0.p_ctrl, g_can0.p_cfg)` The `open` API configures CAN Channel 0. This function must be called before any other CAN functions. Note: This call is made automatically during system initialization, prior to entering the users thread. Unless the user closes the module, open will not need to be called.
.close	`g_can0.p_api->close(g_can0.p_ctrl)` The close API handles the clean-up of internal driver data.
.read	`g_can0.p_api->read (g_can0.p_ctrl, p_args->mailbox, &receiveFrame)` The `read` API reads received CAN data.
.write	`g_can0.p_api->write (g_can0.p_ctrl, 0, &transmitFrame)` The `write` API write data into the CAN transmit frame buffer and send it out.
.control	`g_can0.p_api->control(g_can0.p_ctrl, CAN_COMMAND_MODE_SWITCH, &mode);` with `can_mode_t mode = CAN_MODE_LOOPBACK_INTERNAL;` The control API can change the CAN mode of operation.
.infoGet	`g_can0.p_api->infoGet(g_can0.p_ctrl, p_info)` The `infoGet` API retrieves the CAN mode of operation.
.versionGet	`g_can0.p_api->versionGet(version)` The `versionGet` API retrieves the module version information.

Basic API functions

Source: Renesas Synergy CAN HAL Driver Module Guide r11an0065eu0101-synergy-can-hal-mod-guide

Renesas.com