CAN-HG overview

Augmenting Classic CAN for Performance and Security

Dr. Ken Tindell, CTO Canis Automotive Labs

Document number	1905
Version	04
Issue date	2020-12-14

1 Overview

1.1 Motivation

CAN is a successful and widely adopted protocol, offering robust real-time communications with guaranteed latencies and atomic multicast. But there are limitations. Many applications today require much more bandwidth than the maximum 1Mbit/sec that CAN offers, requiring networks to be partitioned into multiple buses with gateways copying messages between buses. There is no security in CAN so that a single node contaminated with malware can subvert the bus.

CAN needs augmenting to bring it up to date:

- It needs to run much faster
- It needs security

The CAN-HG protocol augments existing CAN with ***H***igher speed data and provides bus ***G***uarding support to stop spoofing and denial-of-service attacks. Most importantly, CAN-HG interoperates completely with existing CAN systems:

- It operates with the existing CAN physical layer – both the wiring and the transceivers.
- It is completely compatible with standard CAN frames – it augments existing CAN frames with additional data at a bit rate of up to 10Mbit/sec.
- Existing CAN controller hardware is not disturbed by CAN frames augmented with CAN-HG data (the data is simply invisible to controllers).

1.2 CAN-HG augmenting classic CAN

CAN-HG augments classic CAN to meet the challenges described above by adding *fast bits* inside of a CAN bit. These data bits are much faster than the bits of the underlying CAN protocol:

- A CAN bit is typically 2000ns in duration (i.e. 500kbit/s) and there is much empty time between CAN bit sample points.
- The CAN physical layer can sustain a bit time of 100ns (i.e. 10Mbit/sec) if special signal processing measures are taken.

Fast bits can be added to a CAN frame by taking advantage of a feature of the CAN specification.

Copyright © 2020 Canis Automotive Labs Ltd.

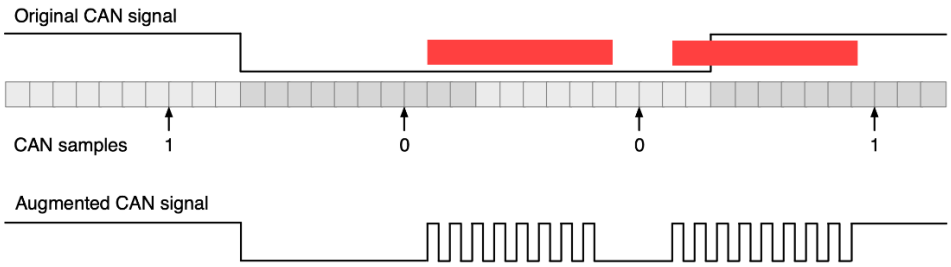

Figure 1: Adding fast bits into a CAN signal

CAN requires that the bus signal be ignored after a bit is sampled as a 0 until the next sample point. Figure 1 shows an example of placing CAN-HG fast bits across four CAN bits of 1001. In the top trace, the red area shows where fast bits can be added: after a sample point where the CAN bit has been sampled as a 0, and before the next sample point. The bottom trace shows an example of the augmented CAN signal.

The CAN physical layer introduces bit asymmetry and noise due to reflections from impedance mismatches and at 10Mbit/sec these effects are significant. CAN-HG adopts a digital filtering system combined with a dynamic bit decoding scheme (CAN-HG has been tested on various topologies including 24m long FLRY-A cables with several unterminated stubs).

The placement of fast bits requires calculations that take account of the physical layer characteristics, clock accuracy, and so on. CAN-HG avoids the need for these calculations to be made in setting up a system and instead defines standard *profiles* that specify the complete network-wide setup of a CAN-HG system including the CAN characteristics. A system is configured simply by selecting the profile.

1.3 CAN-HG frames

A CAN-HG frame is a set of fast bits embedded inside a CAN frame. A CAN-HG frame consists of two parts:

- A CAN-HG *header* of 32 bits. All CAN frames can be augmented with a header.
- An optional CAN-HG *body* of up to 928 bits (for CAN running at 500kbit/sec). A body carries up to 104 bytes of user payload data and to achieve this the augmented CAN frame needs to be an 8-byte CAN frame (such a CAN frame is called a 'carrier frame').

The CAN header is placed into a CAN frame after the sample point of CAN frame bit 'r0'. The header contains:

- An 8-bit physical source address (designed to map directly on to J1939 and CANOpen addresses)
- A 15-bit CRC[1] that is applied over the header bit and the CAN frame ID.

[1] The header CRC has a Hamming Distance of 6

Copyright © 2020 Canis Automotive Labs Ltd.

- Calibration bits (that allow the receive to sync with the sending bits).
- Flags (one flag indicates that a body follows the header).

The full CAN-HG frame is shown Figure 2:

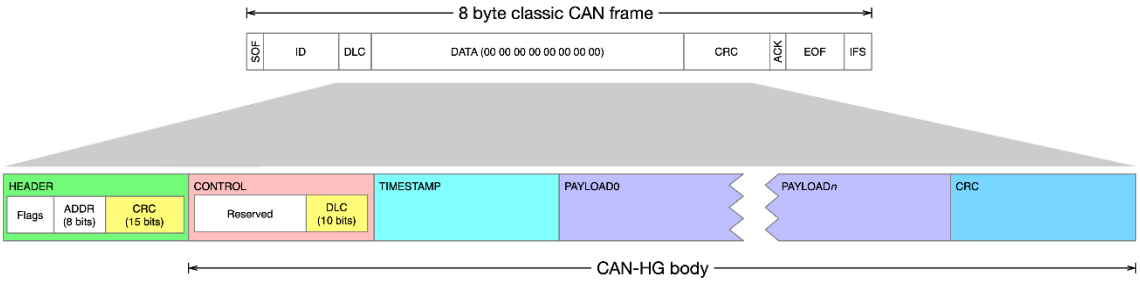

Figure 2: Anatomy of a CAN-HG frame embedded inside a classic CAN frame

A CAN-HG body consists of the following fields:

- 32-bit Control field
- 32-bit Timestamp field
- Payload (multiple 32-bit words)
- 32-bit CRC

The number of user payload words is fixed for a given profile: CAN-HG frames exist only inside CAN frames and the size of the CAN carrier frame is fixed. To make it easier to write driver software, the Control field contains a 10-bit DLC value that the software sets to indicate how many bytes within the user payload are valid (the reason the DLC is 10 bits is that when the CAN bit rate is lower, the number of CAN-HG payload words that can fit inside a carrier frame goes up, and CAN-HG payloads of up to 1024 bytes are possible).

The Timestamp field is the time when the falling edge of the CAN SOF bit happened measured by the *sender's* clock. A receiver can compare its own SOF timestamp with the sender's to get a measure of global time; this can be used to synchronize events across a bus (e.g. when sensors are read, or actuators commanded).

The Payload field contains user data. In Profile 0 (where the CAN baud rate is 500kbit/sec) this is 104 bytes.

The CRC field[2] protects the CAN-HG frame, both header and body and includes the CAN frame ID. It provides a Hamming Distance of 6 (necessary for high-integrity applications).

[2] The CRC comes from CMU's 'CRC Polynomial Zoo'

Copyright © 2020 Canis Automotive Labs Ltd.

2 CAN-HG security

2.1 Background

Approaches to CAN security have traditionally taken two forms:

- A security gateway. This patrols what goes between a 'trusted' and 'untrusted' CAN bus by receiving messages on one side and either discarding or re-transmitting them on the other side.
- Encryption. A layer is defined above CAN frames and extra information is added to include a *message authentication code* that the receiver uses to verify that the message came from the sender and has not been tampered with.

These have drawbacks. For example, a security gateway does not prevent carefully crafted messages from exploiting a bug in the firmware of a node on the trusted side and hijacking it. Encryption requires more bandwidth to transmit messages and cannot protect against a node being hijacked and used to forge messages with the keys contained in the node. And neither approach protects the bus from denial-of-service attacks[3].

CAN-HG takes a different approach and operates at a level below the CAN frame by using fast bits to communicate security information.

2.2 Message integrity

Message integrity means that all receivers should only act on messages that were created by the legitimate sender. For CAN this means that the frame should come from the node implied by the ID of the frame. CAN-HG supports message integrity with hardware: a type of device called a *Bus Guardian* is added to a node between the CAN controller RX/TX lines and the CAN transceiver:

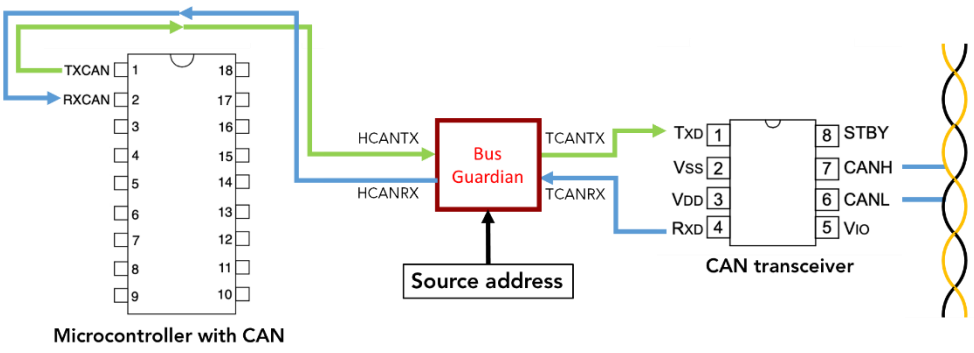

Figure 3: A standard CAN microcontroller with a CAN-HG Bus Guardian

[3] For a detailed analysis of various CAN security approaches see Canis Automotive Labs document 1901 "CAN Bus Security: Attacks on CAN and their mitigations".

Copyright © 2020 Canis Automotive Labs Ltd.

Figure 4 below shows a standard CAN frame, with the 'CAN RX' line showing the digital output of a CAN transceiver and 'CAN H' and 'CAN L' showing the voltages on the twisted pair CAN bus. The CAN ID is highlighted in green.

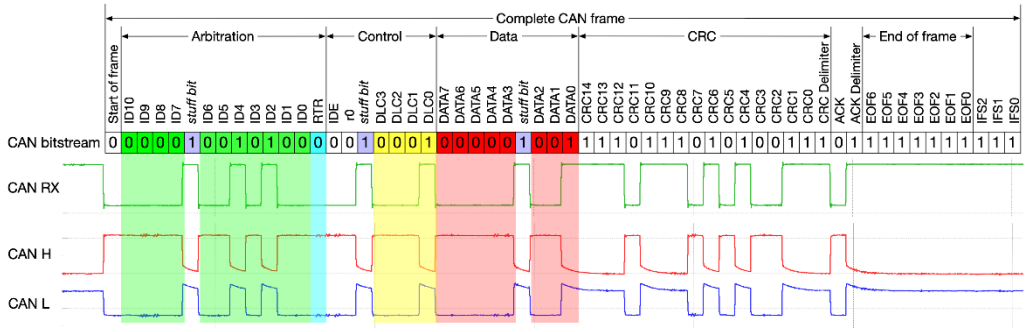

Figure 4: A standard CAN frame (has an 11-bit ID of 0x14 and a 1-byte payload)

The Bus Guardian augments a CAN frame by adding CAN-HG headers to CAN frames as they are transmitted from the existing CAN controller hardware. The header contains the source address of the sending node.

Figure 5 shows the output of a logic analyzer protocol decoder for CAN-HG.

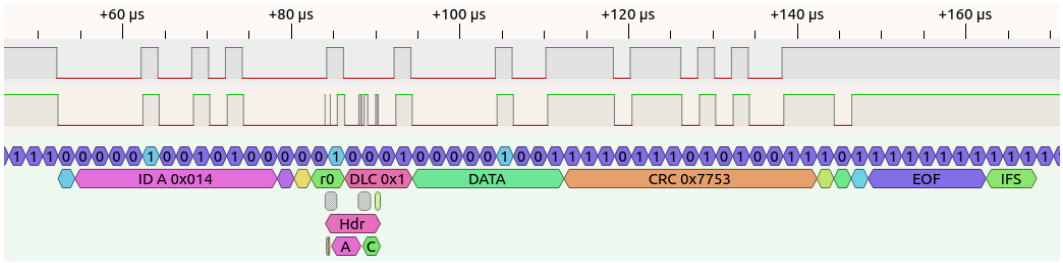

Figure 5: Bus Guardian adding a CAN-HG header inside a CAN frame

The top trace is the CAN TX signal from a CAN controller. The second trace is what is seen on the CAN bus after the Bus Guardian adds fast bits. CAN bits and CAN fields are shown and then underneath are the CAN-HG fast bits and decoded CAN-HG fields.

The fast bits appear as 'glitches' at this scale (a fast bit here is about twenty times shorter than a CAN bit). Zooming in to the CAN-HG header area shows the fast bits in more detail:

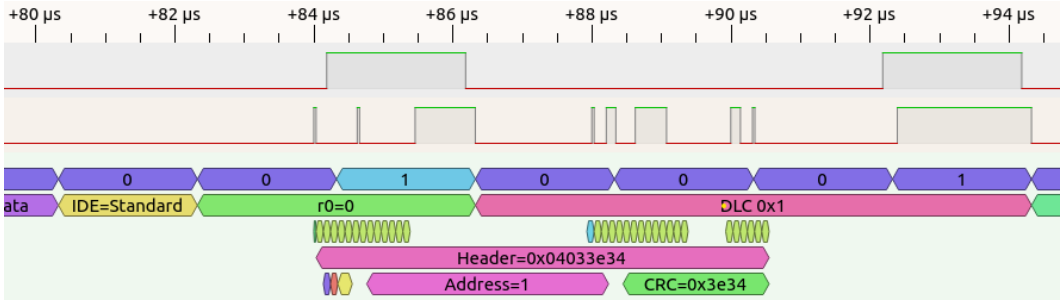

Figure 6: A closer look at the CAN-HG header fast bits

Copyright © 2020 Canis Automotive Labs Ltd.

The individual fast bits can be seen and the protocol decoder shows how they are used to encode flags, a source address and a CRC.

A CAN bus is protected by a central security node that contains CAN-HG Intrusion Detect System (IDS) hardware (Figure 7 and Figure 8).

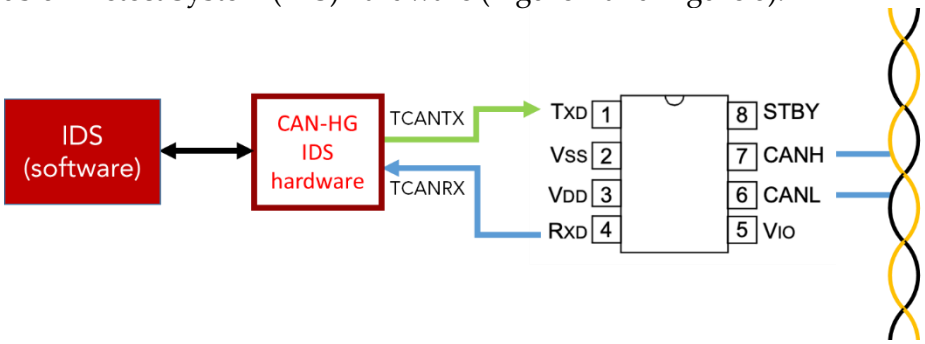

Figure 7: The central security node with CAN-HG IDS hardware

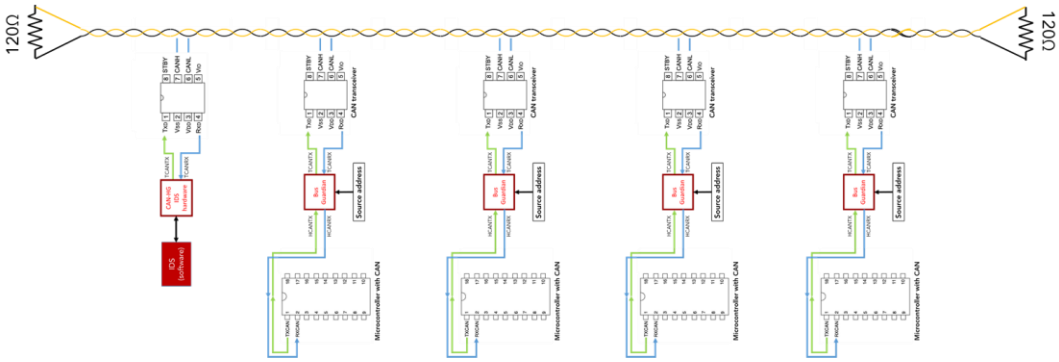

Figure 8: An example system of a central security node protecting a CAN bus of four other nodes

The basic operation of the central IDS is as follows:

- The CAN-HG IDS hardware decodes the CAN-HG header before the rest of the CAN frame is fully received and passes this to IDS software.
- The IDS software runs (typically in an interrupt handler) and examines the CAN ID of the partially received frame and the CAN-HG header.
- If the source address in the header does not match the expected source address of the CAN ID, then the IDS software determines this frame is a spoof.
- The IDS software destroys the frame by instructing the CAN-HG IDS hardware to raise an error (i.e. transmit an error flag of 6 dominant bits) to destroy the CAN frame.

The CAN protocol defines three major stages of frame transmission.

- The first phase is arbitration. Multiple transmitters with a valid frame start to transmit the CAN ID. A transmitter will drop out of arbitration after transmitting a recessive bit but reading a dominant bit.

Copyright © 2020 Canis Automotive Labs Ltd.

- The next phase is transmitting the DLC, data and CRC. After arbitration is finished there is a single transmitter remaining4. and this sends the remaining bits of the CAN frame.
- The last phase is message acceptance. At the end of the EOF field, receivers will accept the bit sequence as valid and pass the frame to the application software.

If an error occurs at any point during transmission, then the CAN protocol stops receiving the frame and all CAN nodes resynchronize. Forcing an error on to the bus before the last phase prevents a CAN frame from being received. This part of the CAN protocol means that no other node on the bus will see the spoof frame because it is stopped before it gets to the end of the EOF field of the frame.

IDS software that actively *protects* the bus is called an *Intrusion Detection Prevention System* (IDPS). Because CAN bits are relatively long (each bit is 2 microseconds at 0.5Mbit/sec) there are many microseconds of 'thinking time' before the end of the EOF field is reached and by which time the IDS software must have reacted. In even low-powered microcontrollers the IDS can execute thousands of CPU instructions in this time.

This IDPS approach provides authentication of the message directly in hardware: the spoof is detected because of where it comes from, and the address identifying that is injected by the Bus Guardian hardware at the source node (the Bus Guardian is not configurable, and the source address is fixed when the node is manufactured).

The Bus Guardian provides some additional security measures beyond injecting a CAN-HG header. For example, if the Bus Guardian sees its own source address being used on the bus then it automatically destroys the frame (since it must be a spoof). This prevents a device directly attached to the CAN bus from forging a CAN-HG header.

2.3 Bus availability

Message integrity is a necessary but not sufficient property of a secure system. A system must also be available: an attack should not be able to stop the normal operation of the system. For CAN this means that an attack needs to be stopped from disrupting the bus with a denial-of-service (DoS) attack.

There are lots of ways CAN is vulnerable to a DoS attack, from simply flooding the bus with high priority messages to carefully timed attacks on the CAN protocol itself. For example, the well-known Bus-Off Attack can force a node off the bus by injecting errors into its frames until the error confinement part of the CAN protocol is triggered and the CAN controller is taken offline. Attacks on the CAN protocol can take place when malware in the node re-purposes the CAN RX/TX pins of the

[4] CAN remote frames are an exception to this rule.

Copyright © 2020 Canis Automotive Labs Ltd.

controller into general purpose I/O (GPIO) pins under software control and 'bit bangs' CAN signals that can be subtly altered to mount an attack[5].

CAN-HG has specific support for ensuring bus availability:

- The Bus Guardian checks for CAN protocol attacks from its host and temporarily ceases passing through the CAN controller's signal to the CAN transceiver, preventing further attacks for a time.
- The central IDPS can broadcast a 'cease' command on CAN that causes the Bus Guardian to block the attacking host CAN controller's signals until further notice.

The cease command is sent in a CAN frame with the highest priority CAN ID (i.e. an 11-bit ID of 0). This ensures that the command cannot be blocked by a bus flood and will be received by a Bus Guardian. The cease command itself has special protections beyond the CAN-HG message integrity measures: the Bus Guardian detects an attempt to spoof CAN ID 0 and temporarily takes the host off the bus for long enough for an IDPS cease command to be sent to confirm a permanent block.

2.4 Confidentiality

Confidentiality is the third 'triad' of security: keeping the contents of communication secret. In embedded systems this is less important: most messaging is concerned with non-confidential real-time sensor and actuator data. Consequently CAN-HG does not have specific support defined for confidentiality. Instead, a higher layer can use the large CAN-HG payload to contain the extra information required for encryption (Initialization Vectors, counters, etc.).

Canis Automotive Labs has developed a CAN encryption layer called 'CryptoCAN' that uses AES in CFB mode to encrypt the payloads of CAN frames, with Message Authentication Codes (MACs) using the NIST CMAC algorithm. This layer can be applied to the payload field of a CAN-HG frame body. There are currently reserved bits in the CAN-HG Control field and some of these could be used to describe the encryption scheme in an encrypted payload.

2.5 Security policies

CAN-HG security provides security *mechanisms* in hardware, such as:

- Providing events with metadata throughout a CAN frame
- Destroying a frame
- Handling IDPS commands

The security *policies* are implemented in the IDPS software that meets overall application-specific system requirements.

---

[5] The CANHack toolkit is an open source bit-banging library for CAN; see https://github.com/kentindell/canhack

Copyright © 2020 Canis Automotive Labs Ltd.

3 Implementation

3.1 CAN-HG engine hardware "`hgmac`"

The core CAN-HG protocol is implemented in a Verilog hardware IP block called `hgmac`. This contains a CAN protocol engine plus the logic to augment an external CAN signal with CAN-HG data. It synthesises to just under 5000 gates.

Other hardware modules are placed around the core engine to create specific devices, as discussed below.

3.2 Bus Guardian

The Bus Guardian hardware IP consists of the core CAN-HG engine plus two further modules:

- A host CAN signal monitor and blocker
- A handler for received IDPS commands

The Bus Guardian can be implemented in a standard-alone device (typically a small low-cost FPGA) or integrated into a System-on-Chip (SoC) device, or even integrated directly into a CAN transceiver (the Bus Guardian does not use any non-volatile memory such as EEPROM or flash and the number of gates is low enough to fit with large geometry processes).

Canis Automotive Labs has developed a specific FPGA solution called the Mercury Bus Guardian, a tiny multi-chip module designed to ease early low-volume adoption of CAN-HG (Figure 9).

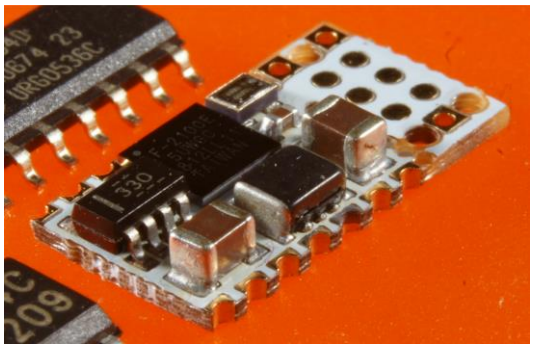

Figure 9: The Mercury Bus Guardian multi-chip module

Copyright © 2020 Canis Automotive Labs Ltd.

3.3 IDPS controller

IDPS controller consists of the core CAN-HG engine plus a set of registers and buffer memory to allow IDPS software running on a host CPU to be driven by CAN events during a frame being sent on the bus and obtain metadata on the CAN frame (e.g. the source address in a header). In addition, the IDPS controller permits the transmission of CAN carrier frames with CAN-HG bodies carrying a full CAN-HG payload. The trace in Figure 10 shows a CAN carrier frame augmented by CAN-HG fast bits. The CAN bit rate is 0.5Mbit/sec and the augmented data is at 10Mbit/sec.

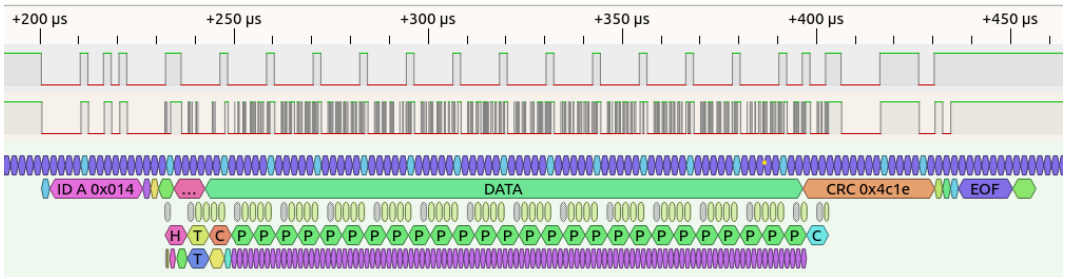

Figure 10: An example CAN carrier frame augmented by CAN-HG

The top line shows the CAN TX signal produced internally and the second line shows the output from the IDPS controller. The fields marked "P" are the 32-bit payload words and underneath the payload words are shown the individual payload bytes.

Large CAN-HG payloads inside CAN carrier frames are particularly useful for specific applications where high-speed transfers are important, such as high-bandwidth sensors or bulk transfer of data to and from an external connection.

The IDPS controller is accessed via AXI4 bus or SPI slave interfaces. There are three main deployment options:

- A hardware IP block on a hybrid FPGA with an integrated microcontroller subsystem and connected to the microcontroller via AXI bus.
- A hardware IP block on an SoC and connected to the rest of the system via an AXI bus.
- The Mercury IDPS, a standalone small FPGA providing an SPI interface to the host CPU.

In all cases, the IDPS functionality is carried out with firmware running on the host CPU that uses the IDPS controller to obtain data and carry out CAN-HG security instructions.

Copyright © 2020 Canis Automotive Labs Ltd.

3.4 CAN-HG controller

The CAN-HG controller is built around the CAN-HG engine and adds a register and buffering interface. The controller is designed to be an augmented CAN controller and contains a receive frame FIFO (with CAN ID filtering masks) and a transmit frame buffer. Outgoing CAN frames are augmented with CAN-HG headers and carrier frames can be sent with CAN-HG bodies containing large payloads.

The CAN-HG controller also contains an IDPS command handler that (like the Bus Guardian) accepts 'cease' commands from an IDPS and will disable the transmission of the CAN controller.

Three hardware transmit buffers are included to avoid the problem of priority inversion[6] when transmitting frames. These are internally arbitrated by CAN ID and the winner selected when bus arbitration starts. The interface to the buffers from the host MCU has been designed to simplify driver software (particularly AUTOSAR CAN drivers), with the interface to the driver using a novel 'tag' mechanism: an 8-bit value is supplied by the driver when writing a CAN frame to the controller (this tag has meaning to the driver as a frame indicator – it might be an array index or a pointer offset) and all controller events are communicated back to the driver using this tag.

There main deployment options are the same as for the IDPS controller:

- A hardware IP block on a hybrid FPGA with an integrated microcontroller subsystem and connected to the microcontroller via AXI bus.
- A hardware IP block on an SoC and connected to the rest of the system via an AXI bus.
- The Mercury CAN-HG controller, a standalone small FPGA providing an SPI interface to the host CPU.

SPI is a relatively slow chip-to-chip protocol for communications data that is as fast as 10Mbit/sec and alternatives for the Mercury CAN-HG interfacing are being explored.

[6] Priority inversion is a major problem for real-time systems and can lead to urgent messages being delayed for an arbitrarily long period of time, causing all kinds of timing faults. See https://kentindell.github.io/2020/06/29/can-priority-inversion/ for a description and demonstration of priority inversion on CAN.

Copyright © 2020 Canis Automotive Labs Ltd.