CAN protocol enhancement

This article describes the enhanced CAN protocol called CAN-HG and the features of the IC circuitry from Canis that implement it.

CAN-HG has been designed to meet two important requirements: Increasing the bandwidth and guarding the network. There is a third requirement – and this requirement is the most important: the protocol enhancement must be completely compatible and interoperable with Classical CAN. It must run on existing wiring with existing nodes using legacy CAN controllers in legacy micro-controllers. Any approach that requires every node on a bus to be re-developed for new micro-controllers is infeasible: it must be possible to freely mix Classical CAN and CAN-HG on the same network segment.

Increasing the throughput

The bit time in CAN is set according to the electrical characteristics of the physical CAN network and the dominant factor is the propagation time across the bus: CAN arbitration requires that there is sufficient time for all nodes to signal a bit and for the signal to have reached all other nodes before the line is sampled. After arbitration has been decided this propagation time constraint no longer applies and shorter bit times could be adopted. But a new frame format, in which there is a switch to faster signaling causes compatibility problems: legacy CAN controllers cannot follow the new frame format and would typically generate error frames and drive the transmitter into bus-off state. CAN-HG solves this problem with the notion of a carrier frame.

A carrier frame is a CAN frame with a payload of 8 byte fixed to 30 00 00 00 00 00 00 00$_{16}$. After bit stuffing this results in the following bit pattern:

10000011100000100000100000100000100000100000010
00001000001000001000001000001000001

The underlined bits are the DLC field and those in bold are stuff bits. The others are the payload bits in the 8-byte CAN data field. The digital signal to the CAN transceiver (i.e. the TX pin from a CAN controller) is shown in Figure 1.

In a carrier frame are fourteen intervals of five dominant bits (i.e. 00000) followed by a recessive bit. The proposed CAN-HG protocol adds short-duration bits – called Fast Bits – within these intervals. The Fast Bits are placed so that all CAN controllers receiving the frame (including the transmitter) see only the original signal. This is ▷

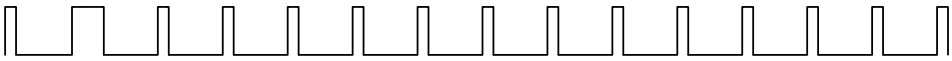

Figure 1: Carrier frame signals at the TX pin (Photo: Canis)

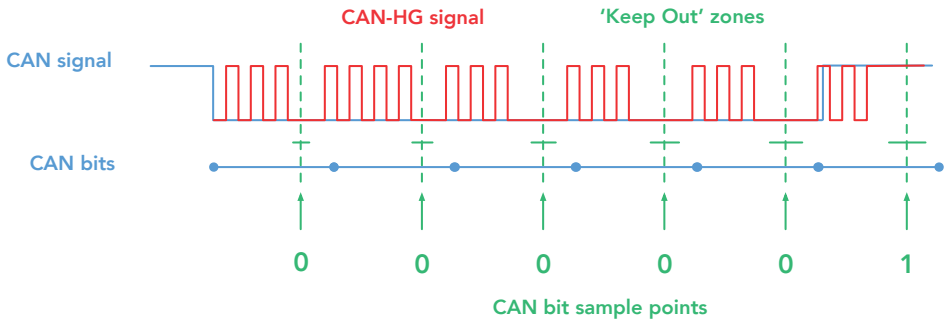

Figure 2: A 6-bit CAN interval beginning with a falling edge, recessive-to-dominant (Photo: Canis)

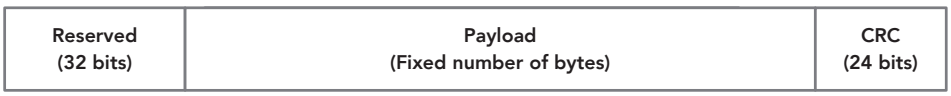

Figure 3: CAN-HG frame format with Reserved, Payload, and CRC field (Photo: Canis)

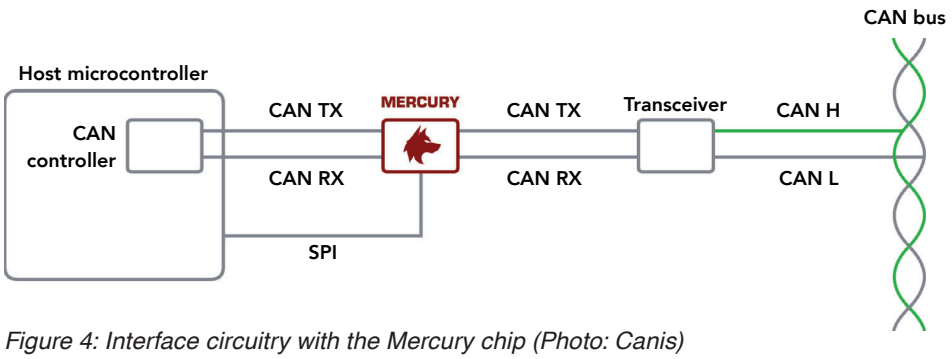

Figure 4: Interface circuitry with the Mercury chip (Photo: Canis)

illustrated with a simplified example in Figure 2. It shows a 6-bit interval beginning with a falling edge from a recessive bit (i.e. 1) and ending at the end of the final recessive bit, with the vertical arrows showing the sample points for each bit. The fast bits overwrite the original CAN signal but return to the original signal value around the sample point.

In ordinary circumstances it would not be possible to put arbitrary edges within a CAN bit: a falling edge initiates the re-synchronization process, which adjusts the sample point and this would then cause later bits to be misread, leading to a CAN error frame being raised and a failed frame transmission. But with CAN-HG the format of the carrier frame and the placement of fast bits are designed to exploit a feature of the re-synchronization: the first falling edge at the beginning of the interval is a re-synchronization point. All connected CAN controllers will perform sample point adjustments to offset from when this edge is detected. But further falling edges within the same bit (up to the sample point) will not result in another re-synchronization: this is prohibited by the Classical CAN protocol. Furthermore, falling edges after the first sample point and in subsequent bits in the interval (up until the sample point of the last bit in the interval) will also not result in a re-synchronization: the Classical CAN protocol prohibits this, if the previously sampled bit is dominant.

The first Fast Bit in an interval is placed a suitable time after the falling edge marking the start of the first bit in the interval. The delay is long enough to give all controllers time to have detected the edge because the CAN controller state machine polls for the falling edge with its time quantum clock. Fast Bits are not located within a Keep Out zone around the sampling point to ensure that all connected CAN controllers see the original signal. The zone is large enough to encapsulate the earliest possible time any CAN controller could sample the bit to the latest possible time. This must account for jitter due to the time quantum polling period: the re-synchronization point – and therefore the sampling points – will vary relative to the falling edge at the start of the interval. It must also account for the different nominal sample points of each CAN controller: they typically will be clocked at different frequencies and have a different number of time quanta per bit.

The zone must also allow for the effects of clock drift in the oscillators of each CAN controller. CAN-HG allows each of the six Keep Out zones to be of a different duration to account for clock drift: for the first zone the clocks can have drifted only a small amount but by the last zone the clocks may have drifted by a significant amount. This is to allow more Fast Bits to be placed in the interval.

2.4 Encoding fast bits

Fast Bits are encoded with a simple NRZ asynchronous serial communication scheme. The falling edge at the start of the interval is taken as the end of a stop bit and used as a synchronization point and the first Fast Bit is located a fixed time offset from this point. The last Fast Bit before each of the first five Keep Out zones is designated a stop bit (and hence is always a logic 1) and the falling edge of this bit is used to resynchronize the fast bit timing.

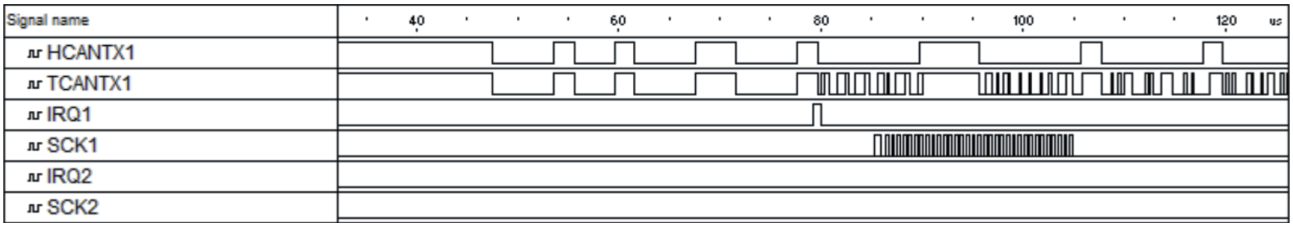

Figure 5: Logic analyzer trace of the first part of a CAN carrier frame (Photo: Canis)

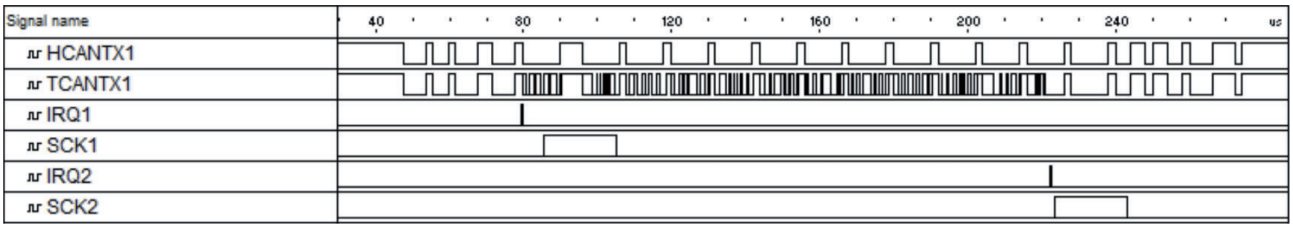

Figure 6: Logic analyzer trace of a received CAN-HG frame (Photo: Canis)

CAN-HG also supports bit stretching: it allows the duration of a Fast Bit to vary based on the time since the last stop bit falling edge so that bits closer to the sync point can take advantage of the lower clock drift and be shorter than ones further away. This can significantly increase the size of the overall frame payload.

The following Fast Bit parameter settings configure the CAN-HG network:

- The skip times from the falling edge of a stop bit (or the start of the interval) to a Fast Bit
- The number of subsequent Fast Bits (including a subsequent stop bit)
- The initial duration of the first Fast Bit after a stop bit
- The bit stretching factor to add to the duration of each subsequent Fast Bit
- The offset to Fast Bit sampling to compensate for the additional CAN transceiver dominant-recessive propagation delay
- The total number CAN-HG payload bytes (all CAN-HG frames within a network have the same fixed payload size)

A set of CAN-HG Fast Bit parameters can be checked for validity against a given set of network properties (oscillator accuracy, CAN sample point ranges, etc.). It is a simple search problem to find a set of parameters that pass the validity checks. Payloads of 120 bytes are achievable for common network properties.

A CAN-HG frame has the following format: All fields are an 8-bit multiple and are fixed in size. The Reserved field has a fixed length of 32 bit. It is reserved for future usage (it is intended that it will contain sequence number and timestamp information generated automatically by the CAN-HG hardware). The Payload field is of fixed but configurable size, depending on the parameters set for the network (as described above).

The CRC field has a length of 24 bit, with a polynomial of $5D6DCB_{16}$ and an initial value of $FEDCBA_{16}$ (the same as the Flexray CRC), which gives a Hamming Distance of 6 for frames up to 2024 bit. The CRC is calculated over the CAN-ID of the carrier frame and the Reserved and Payload fields of the CAN-HG frame. There is a long-standing problem where the CRC of CAN has a Hamming Distance of just 2 in certain pathological cases due to stuff bits. This is addressed in CAN-HG: the checking in CAN-HG not only checks the CRC-24 but also checks that the carrier frame is of the required fixed format. Early computational experiments suggest that a carrier frame consequently has an effective Hamming Distance of 7. A CAN-HG error is handled in the same way as for the CAN protocol: an error frame is generated that destroys the carrier frame and triggers the normal CAN error recovery process.

Guarding the network

There are two major properties of any secure messaging scheme:

- Authenticity: To act on the contents of a frame, the receiver must be sure the frame came from the genuine sender.
- Secrecy: The contents of the frame must be kept secret and shared only with the intended recipients.

Authenticity is important because it allows a system to be built in which each node can trust the contents of the frames and does not have to construct elaborate (and potentially faulty) protocols for checking the data. Secrecy is also useful, not just for protecting sensitive data (perhaps the firmware of a node during download) but also because it makes it harder to reverse engineer or tamper with a system.

A common way to obtain authenticity and secrecy is with cryptographic protocols. But there are significant costs to this: there is the complexity and overheads within each node of including cryptographic protocols (code space, RAM, CPU time), the overheads on the bus (extra data needs to be sent to authenticate frames and to prevent replay attacks and there are synchronization delays that add to latencies) and the challenge of distributing and storing secret keys in every node.

CAN-HG provides in hardware both frame authenticity (for CAN frames as well as CAN-HG frames) and frame secrecy.

CAN-HG provides frame authentication by means of anti-spoofing: preventing spoofed frames from reaching nodes. The anti-spoofing features apply at the CAN level: all CAN frames are protected, not just CAN-HG frames.

The core concept of CAN-HG bus guarding is the authorized frames list: it is a secure list inside the CAN-HG hardware that lists the CAN frames that the host node is authorized to send and the CAN-HG frames that the host node is authorized to receive.

An outgoing CAN frame from the host node is checked against the list and if it is not on the list of transmitted CAN frames then the frame is blocked from transmission. This ensures that if any node with CAN-HG hardware is hijacked it cannot send spoofed CAN frames on the bus.

An incoming CAN frame to the host node is checked against the list and if it is on the list of transmitted CAN frames then the CAN-HG hardware destroys the CAN frame. A frame is destroyed in the same way as for a faulty CAN-HG frame or carrier frame: an error frame is generated that destroys the frame and triggers the normal CAN error recovery process. This ensures that all nodes that normally receive this frame from the host node are protected against spoofing of that frame. The above means that CAN-HG provides protection to CAN nodes that don't include CAN-HG hardware.

CAN-HG supports secrecy for CAN-HG frames. The authorized frames list has entries for the identifiers of carrier frames containing CAN-HG payloads that the host node can receive. A CAN-HG frame payload is only decoded and passed to the host node if the incoming frame appears on this list. Unauthorized nodes – including those without CAN-HG hardware – will see only the carrier frame.

The first CAN-HG circuitry

The Mercury chip by Canis is a small stand-alone integrated circuit that is placed between the micro-controller with on-chip CAN module and the CAN transceiver. The first demonstrator system built by Canis uses the STM32F405 micro-controller by ST-Microelectronics, which comprises a bxCAN module. The CAN TX and CAN RX pins are routed into the Mercury chip, which modifies the bus signals and input those into the CAN transceiver. By default the CAN TX signal from the micro-controller is reflected to the transceiver, and the CAN RX signal from the transceiver is reflected to the micro-controller.

The Mercury chip is interfaced to the micro-controller via SPI as an SPI slave. As well as the four SPI lines (SS, SCK, MOSI, MISO) it includes an interrupt line to request servicing from the micro-controller. The SPI commands are structured so that most of the handling of SPI can be done by DMA in the micro-controller. The Mercury chip does not contain a complete CAN controller: all CAN frame handling is done by the CAN module in the micro-controller, including the generation of carrier frames. The Mercury chip contains a CAN receive state machine that is used to drive the CAN-HG protocol.

When the micro-controller sends a CAN frame this is detected by the Mercury chip and when the ID part has been received it is matched against the authorized frames list (by applying the mask and must-match values for each member of the list). If none of the entries match then the frame is being transmitted illegally and is destroyed: by pulling the TX line to the transceiver to dominant for six CAN bit times ▷

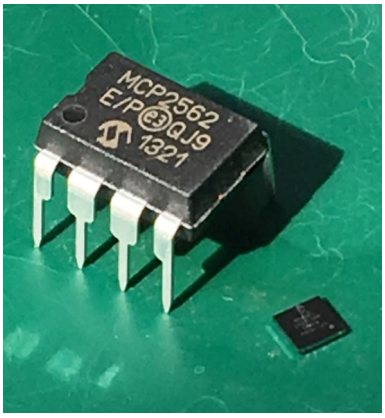

Figure 7: The Mercury chip is on the right; the chip on the left is a CAN transceiver (Photo: Canis)

(this is a CAN error flag and causes the frame to be abandoned and arbitration to be restarted in accordance with the Classical CAN protocol).

If there is a match in the authorized frames list and the frame is marked as carrier frame then the Mercury chip raises an interrupt to request that the micro-controller supplies the CAN-HG payload data for the carrier frame. The micro-controller queries over SPI the identity of the carrier frame and the Mercury chip returns a 6-bit index into the authorized frames list to indicate which frame matched. From this, the micro-controller can determine the CAN-HG payload and push it over SPI. The Mercury chip injects the payload data into the CAN-HG Fast Bits stream in the carrier frame as it continues to be transmitted by the micro-controller.

The logic analyzer trace in Figure 3 illustrates this for a carrier CAN frame with an 11-bit ID of 123_{16}. The signal HCANTX1 is the CAN TX signal from the micro-controller, TCANTX1 is the signal from the Mercury chip to the CAN transceiver, IRQ1 is the interrupt line from the Mercury chip to the micro-controller and SCK1 is the SPI clock from the micro-controller (the SPI master). The trace shows how the interrupt is generated after the ID field and how the Mercury chip sends the Fast Bits representing the Reserved field while the micro-controller engages in an SPI transaction to push payload data (the SCK1 signal shows SPI activity).

Incoming CAN frames are handled as normal by the on-chip CAN module in the micro-controller: the CAN RX signal is passed through the Mercury chip from the CAN transceiver. The ID of the CAN frame is checked against the authorized frames list and if the frame is being received, but matches against a transmitted frame in the list, then the frame is destroyed by the Mercury chip. An incoming frame that matches against an entry in the list that marks it as a carrier frame triggers the CAN-HG frame reception process: the Fast Bits are decoded and the Reserved field is stored, the payload is placed in an internal buffer and the CAN-HG CRC checked. If the CRC does not match or the incoming frame is not a well-formed carrier frame (i.e. is not 8 byte or does not have a payload of $30\ 00\ 00\ 00\ 00\ 00\ 00\ 00_{16}$), then the frame is assumed to be corrupted and is destroyed. If a CAN-HG frame is received correctly then an interrupt is raised and the micro-controller extracts the payload over SPI. The CAN-HG payload is provisional: it must be tied back to the reception of the carrier frame. If the payload is received before the carrier frame completes it is possible that an error subsequently occurs before the carrier frame is received and the frame is destroyed and retransmitted according to the CAN protocol. The micro-controller CAN-HG driver marks the payload as received only when the corresponding carrier frame has been received.

The logic analyzer trace in Figure 4 illustrates the reception of a CAN-HG frame. The signal IRQ2 is the interrupt line at the receiver and SCK2 is the SPI master clock at the receiver. The trace shows how the CAN-HG payload is received before the carrier frame. With the specific network parameters in the example the CAN-HG frame fits into the first thirteen carrier frame intervals and the last interval is not used. The carrier frame is received some time after the CAN-HG frame is received and its payload uploaded to the micro-controller over SPI.

The shown Revision A version of the Mercury chip comes in a 36-pin BGA package.

The Mercury chip supports an authorized frames list of 64 entries with match/don't care masking over CAN IDs. The SPI interface can be clocked at up to 20 MHz. The CAN-HG payload size is fixed at 32 byte. Future revisions to the silicon are planned, including performance and security enhancements. For high-volume applications the Mercury functionality could be integrated into a CAN transceiver package or a micro-controller.

Deployment

CAN-HG was designed to make deployment easy and CAN and CAN-HG traffic can be freely mixed on the bus and so the deployment of CAN-HG can be focused on where it matters most. This means there is no need to update the hardware and software across all nodes on a network. Depending on the threat model for a network only a subset of nodes needs CAN-HG hardware: only frames that need protection from spoofing need hardware to enforce it. And the performance benefits of CAN-HG can be applied first in only the nodes with the highest bandwidth demands. An existing network design needs only minimal changes to obtain security and performance benefits of CAN-HG.

Mercury was designed to make the adoption of CAN-HG even easier: it does not require changes to micro-controller hardware or CAN controller software. A node can continue to use the same silicon and the same software drivers and continue to exploit the specific features of a specific CAN controller (for example, making use of hardware support for Time-Triggered CAN). If a node requires just the security features of CAN-HG, then there are no software changes required: a configured Mercury will destroy spoofed frames without any intervention by the micro-controller. And the higher performance of CAN-HG can be obtained just by adding Mercury SPI drivers. ◀

Author

Dr. Ken Tindell
Canis Automotive Labs
info@canislabs.com
www.canislabs.com

€140 Isolated CAN to USB

Introducing ValueCAN 4-1

- Exceptionally priced CAN to USB tool with no compromises
- 1 CAN channel, up to 1 Mb performance
- Pass-through support via J2534 / RP1210 / DLL
- Standalone mode
- Fully isolated
- IP65 rated
- Aluminum case
- Small size, easy to keep with your laptop
- CAN FD and multi-channel versions also available

Find out more: www.intrepidcs.com/vcan4

IntrepidCS

INTREPID CONTROL SYSTEMS GMBH

www.intrepidcs.com

+49 (0)721 1803083 -1 icsgermany@intrepidcs.com

USA Germany UK Japan Korea China India Australia