Establishing feasibility of isolating higher-layer protocols for CAN into reconfigurable hardware

Aniket Bhattacharya, John Deere Technology Center – INDIA

The protocol stack supporting CAN (which encapsulates the overlying layers in the OSI 7 – layer abstraction model) is commonly implemented in software possibly within an OS or a task scheduler framework. Implementation of the protocol stack in software imposes memory overheads and constrains message handling when the bus is heavily loaded. Attempts to improve message handling capacity often culminate in esoteric, ROM – intensive filtering mechanisms.

This paper discusses implementation of the protocol stack in an FPGA/ CPLD based platform using examples based on J1939. This results in cleaner isolation of reusable IP for the protocol stack, preserves provisions to tailor it without being limited by resources on the micro, and obviates the need of a CAN interface on the micro.

Also, the paper identifies the challenges and tradeoffs involved in having this kind of a hybrid system with the user application executing off a micro, and the protocol stack realized in hardware, focusing on three key areas – time to market, rework in migrating from proven software based protocol stacks, and cost implications.

1 Introduction

Most often, in modern vehicle applications, the CAN protocol stack is implemented as software, and shares the same code space, data space and resources with the ECU application on a microcontroller/ microprocessor (hereinafter generically referred as micro) based platform. While this is a proven technique and has served the vehicle systems community well for quite a while, increasing complexity of vehicle systems are placing additional demands on the micro embedded in the ECU and putting a premium on the resources available to this micro. This, coupled with the need to handle and filter more CAN traffic, drives designers either to select more expensive, resource rich micros to re-use the software, or to redesign the software using complex, non – intuitive techniques to accommodate within the existing micro. The increased complexity of software is sharply accentuated when using embedded OS. The CAN protocol stack functionality, which should ideally be abstracted as a separate layer in the OS, becomes very tightly coupled within the OS kernel and application layer, rendering any debug exercise a very lengthy and time consuming process.

This paper seeks to address this issue by proposing a strategy, wherein the CAN interface and the CAN protocol stack

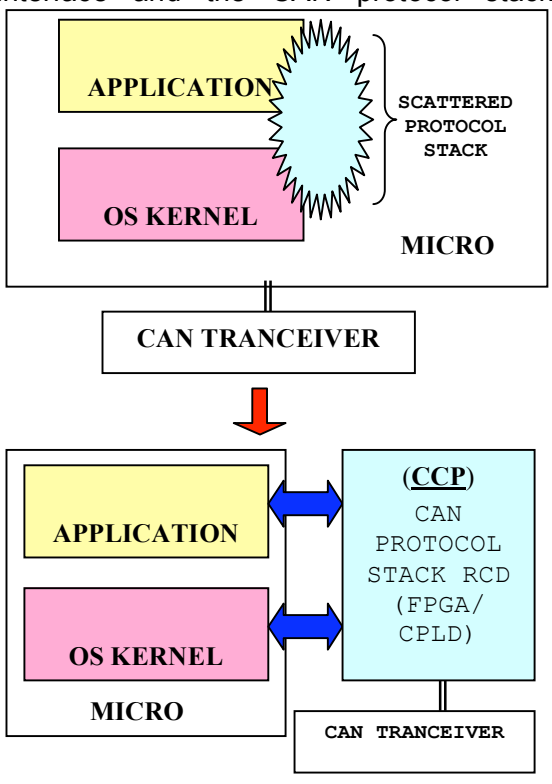

Figure 1: Migrating from the classical approach to the CAN co – processor approach.

functionality are realized on a separate "CAN co - processor" based on Reconfigurable hardware[1] (RCD) thereby relegating all the protocol functionality to the "CAN co - processor" (CCP). This is contrasted with the conventional approach described above in the figure 1.

Since, synthesizable IP for CAN interfaces targeted at RCDs are commonly available, this paper does not delve into the design of the CAN interface. Also, as far as possible, there has been an attempt to suggest a design independent of the particular flavor of the CAN protocol used. No protocol specific details have therefore been discussed. However, part of the design partitioning and data flow has been explained using J1939 as a representative protocol in view of its versatility across applications. The paper does not make any specific recommendations with regard to the micro and the RCD that needs to be used, since these will be dictated by the application requirements.

2 Requirements

The different aspects of the requirement for the "CCP strategy" introduced in the foregoing description are captured as follows.

- The micro should have a fast and seamless communication mechanism with the RCD embedding the CCP.
- The CCP should effectively offload all protocol functionality from the micro.
- Any operations that need authorization/ privileged access should be structured such that the authorization granting is retained by the micro.
- The memory subsystem design should be cognizant of the fact that RCDs are not memory rich resources.

Predicated on these primary requirements, a set of intermediate requirements can be derived. These are summarized below:

- The memory should be shared between the micro and the CCP.
- The shared memory should be mapped into the data memory of the micro, to ensure ease and speed of access. (This warrants the use of a micro featuring Von – Neumann architecture)
- The shared memory space could be zoned into several spaces with read and write privileges mutually exclusive between the micro and the CCP. This minimizes the possibility of simultaneous access, thereby minimizing the injection of wait states when semaphores are invoked.
- Despite processing a request independent of micro involvement, the CCP should be capable of notifying the micro of any change of status warranting the micro's intervention. The same mechanism should allow the micro to respond to requests arriving from other nodes seeking authorization before initiating some sequence.

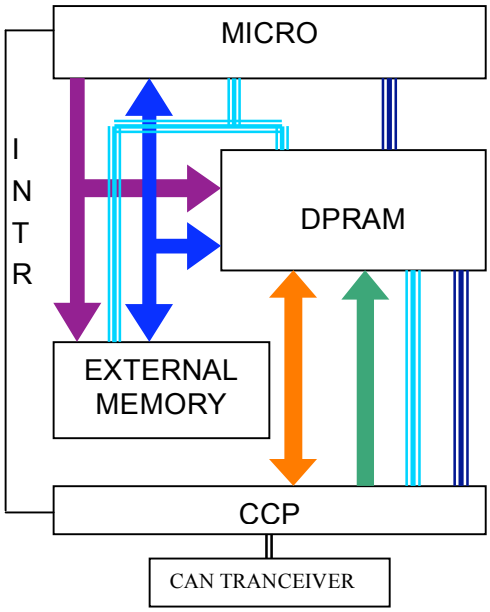

Figure 2: Generic Hardware schematic

Legend: The CCP and the micro have separate dedicated access to the DPRAM. (Addresses shown in violet and green, while data bus shown in orange and blue) Semaphore lines are bundled together in dark blue, while control signals are shown in turquoise. The external memory and the DPRAM are both mapped into the micro memory.

The final set of requirements needs to be arrived at, after considering application and protocol specific caveats. The

[1] 1 RCD is used as a generic term encompassing all kinds of programmable logic devices such as PLDs, CPLDs, and FPGAs.

following illustration (figure 2) depicts a generic hardware that would be consistent with the above requirements.

Notice that a Dual port RAM (DPRAM) has been used providing for an interface mechanism. Use of the DPRAM also allows for a shared memory between the micro and the CCP which could be mapped into the data memory of the micro to streamline access from the micro. This arrangement also addresses the RCD constraint of limited internal memory. An interrupt line from the CCP to the micro provides for a method to communicate an event in the CCP that merits the micro's attention.

3 Memory subsystem Design

Figure 3 shows the spaces that the DPRAM could be zoned into, along with the corresponding access privileges. The emboldened labels refer to the spaces that could be standard across protocols. The other labels denote spaces specific to a J1939 protocol implementation. This paper does not mandate the memory ranges, or organization of these spaces within the addressable memory map.

Command Space: This space in the DPRAM is treated as a write stack by the micro and can be read by the CCP. Whenever any operation is requested by the micro, it pushes a corresponding "command" into this space qualifying that command with necessary parameters. The CCP reads this command, interprets the context of the command, and uses the parameters encapsulated in the command format to initiate the desired operation. This paper does not stipulate any command format, since the format needs to be finalized after arriving at the supported command set.

Data Space: These spaces in the DPRAM contain the data that needs to be transmitted via CAN, or serve as place holders for receipt of data. The particular data space to be used needs to be specified by the command, along with the offset within the particular data space, so that the CCP can construct the starting address of the location in the DPRAM from where it needs to fetch data for subsequent transmission, or to determine the destination for the received data. Thus the access privileges (read/ write) though mutually exclusive, are context specific.

Request Space: The CCP treats this space as a write stack, while read access is retained by the micro. The request space contains details about a request for a transaction that has been received via CAN from another node on the network. The CCP writes the request into the request space seeking authorization to service the received request. The authorization is conveyed via a command from the micro.

Status Space: To notify the micro about a change of status vis-à-vis a command, the CCP pushes status data into this space which it treats as a write stack. Thus, whenever, the CCP starts servicing a particular command, it updates the status space accordingly. Likewise, when the particular command has been completely serviced, or had to be aborted due to some reason, the CCP again writes this status to the status space. The micro can read the status space to determine the status of the particular command.

Interrupt Cause Space/ Register: This space/ register substitutes an interrupt controller. Write rights for this space/ register belong to the CCP, while read rights are retained by the micro. Whenever, the CCP needs to generate an interrupt for the micro, it first writes the cause qualifiers for that interrupt into this space/ register. The cause qualifiers would typically include one field for the interrupt category (request/ status) and another field containing the offset within the request/ status space where further information pertaining to that interrupt has been written. When the micro reads this space/ register consequent to the interrupt, it uses these contents to dereference the correct space and construct an address to read the updated status/ request details.

Active DTCs space[2]: The micro uses this space to write and order diagnostic trouble codes (DTCs) into. The first address in this space contains a code to indicate whether the DTcs are active. The CCP periodically reads the first address in this space and

[2] This space is specific to J1939 protocol stack implementation in the CCP.

determines whether DTCs in this space need to be read, and if so how many. It accordingly reads the subsequent addresses in this space, and broadcasts the DTCs over CAN. (as part of a "DM1" message in J1939)

Previously active DTCs space[2]: The micro uses this space to transfer DTCs from the Active DTCs space when they cease to be active. Thus write privileges to this space are reserved by the micro. The CCP seeks read access to this space when it gets a "Request DM2" from another node on CAN. If and after the micro sanctions this access, the CCP gains read privileges to this space, and responds to the "Request DM2" with the data in this space.

Node names space[2]: This is a special space in that the CCP can gain both read and write access, depending on the command issued by the micro. The micro has read privileges; however it refrains from accessing the space while the CCP has been commanded to use the space to avoid the possibility of a simultaneous access. This space is used to maintain a table of names of the different nodes on the particular CAN. It is periodically (at discretion of the micro) populated by the CCP to update the set of nodes on the network. The first address in this space has the self-name of the node and is read only for both the CCP as well as the micro in normal execution mode.

Node addresses space[2]: This too is a special space since the CCP can gain both read and write access, depending on the command issued by the micro. Whenever the micro wishes to send a destination specific message, it issues a command giving the name of the node it wishes to send the message to. This is indicated by the offset in the name space. The CCP uses this offset to read the corresponding address from the node address space. This address is used to assemble the message transmitted. This space is also populated by the CCP periodically to update the addresses of all the nodes on the network. This space along with the "Node Names space" is crucial to the "address claim procedure" in J1939. The first address in this space has the self-address of the node, which is mirrored within the CCP. This forms part of the source address that is sent out in every message originating from the particular node. In case of a change of source address, the CCP updates the first address of this space, and reflects this change in its internal register as well.

4 Internal architecture

Figure 3 shows the internal architecture of the CCP. This is suggestive only and broadly identifies the blocks and their functionalities. The intent is to provide a template to base the CCP design from.

The heart of the CCP is the "CCP master state machine" diagrammed in figure 4. This is supported by the following blocks:

- Command Handler block. (CHB)
- Protocol Implementation block. (PIB)
- Status generator block. (SGB)
- Interrupt generator block. (IGB)
- Request handler block. (RHB)

The "CCP master state machine", in addition to coordinating all the activities of the CCP operation, also embeds the interface with the DPRAM. This block generates the read and write strobe signals and is responsible for semaphore handling should a simultaneous access situation arise. The state machine unit in this block performs the following activities:

- Routes the address generated by the correct support block (including itself) to the address bus on the DPRAM.
- Generates trigger signals for triggering the state machines in the support blocks.

To elucidate the operation of the CCP, a couple of examples are used.

Suppose micro A wants to send a request for a particular data set (PGN in J1939) available with micro B. Micro A accordingly constructs a command, encapsulating the command code, designator of the data requested, offset of the name for micro B in the node name space, and an enable bit. This command is then pushed by the micro into the command space.

Meanwhile the CCP master state machine block (MSMB) is in the default state – the "CMD CCL" state – wherein it enables the CHB and routes the address generated by

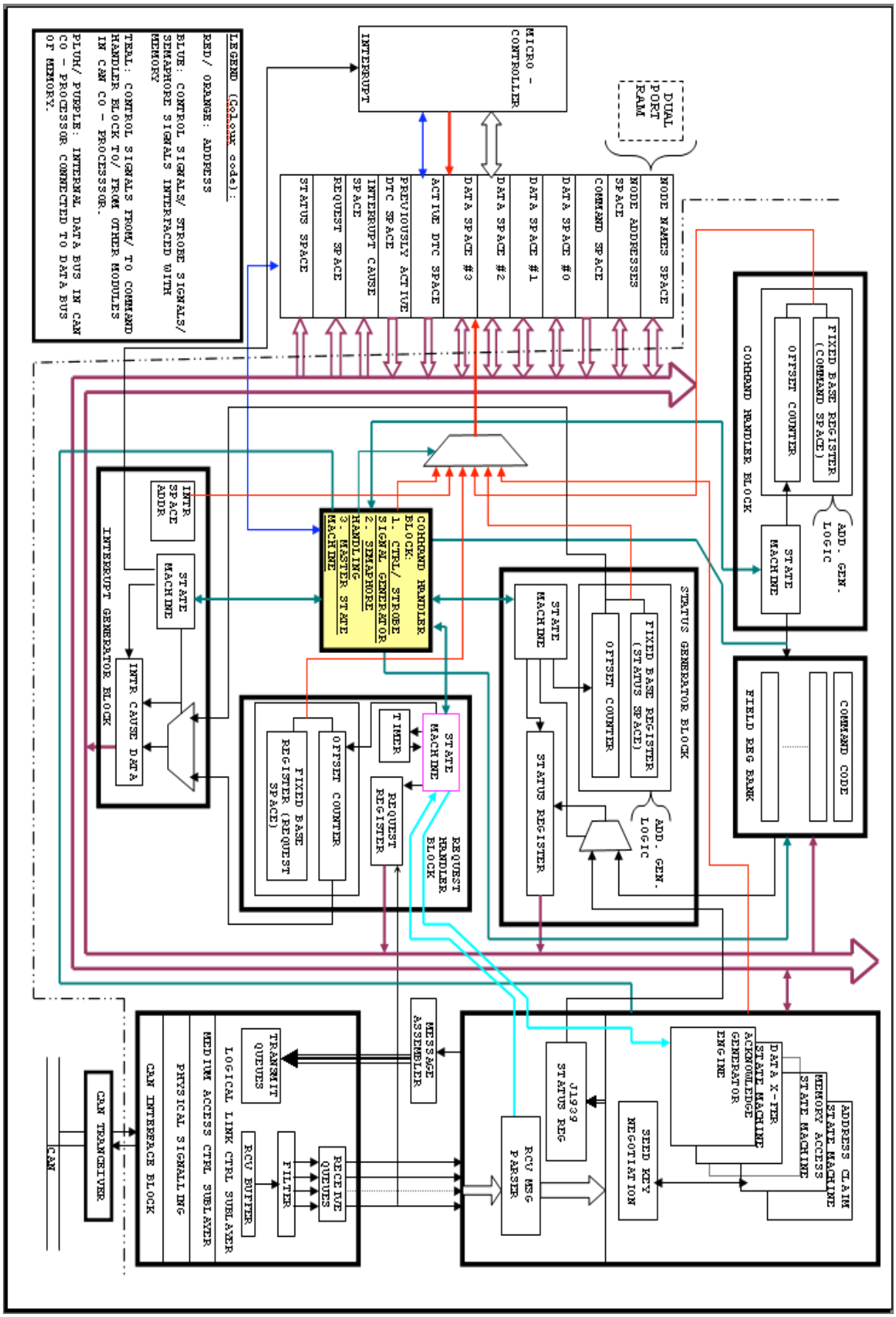

Figure 3: Suggested internal architecture of the CAN co – processor

the address logic module in the CHB to the DPRAM address bus used by the CCB. As the CHB cycles through the command space, it encounters the command written by micro A. Since the enable bit is set, the CHB splits the

command into its constituent fields and mirrors these fields into a bank of “field registers”. It then signals the MSMB, causing the main state to advance to “STS GEN1”. Here the MSMB triggers the SGB. The SGB gets the command designator and the offset of the command within the command space from the CHB, appends a state code for “servicing command” and pushes the status thus constructed, into the status space. The MSMB subsequently advances to the “INT GEN1 state”, wherein it enables the interrupt generator block (IGB). The IGB writes the interrupt cause to the interrupt cause space/ register which it creates by copying the offset in the status space to which the status was written by the SGB, from the address generation logic in the SGB. Further it adds a field to indicate that a status update warranted the interrupt. Finally, the IGB flags an interrupt to micro A, after which the MSMB state advances to “PTCL OPER”.

On receipt of the interrupt, the micro inspects the interrupt cause space/ register to determine the cause of the interrupt. It uses the offset in conjunction with the additional field in the interrupt cause space/ register to read the appropriate address in the status/ request space to deduce further information about what led to the interrupt. In this case, the micro A will read the status space to learn that the command it pushed to the command space is currently being serviced.

Meanwhile, the CCP has arrived into the “PTCL OPER” state. In this state, the MSMB provides DPRAM interfacing services to the PIB. For instance, the message assembler supporting the PIB, needs the source address and the destination address while constructing the request message to be sent out. (In J1939) The destination address is available in the node addresses space at the same offset location as the name of the destination device in the node names space. This offset is available in one of the field registers since it had been sent as part of the command. The PIB constructs the address accordingly and reads the node address space to fetch the destination address to pass on to the message assembler. Likewise the PIB also reads the first address in the node address space (offset = 0) to get the source address to pass on to the message assembler.

In the “PTCL OPER” state, the MSMB generates the appropriate sequence of strobe signals in order to handle suitably qualified DPRAM access requests from the protocol block.

The completion of command processing is signified by the message assembler writing the last (of the series of) message(s) to be transmitted into the appropriate registers in the underlying Logical Link Control (LLC) layer. The LLC, along with the Medium Access Control (MAC) and Physical Signaling(PLS) layers beneath it from the CAN interface, which is tasked with dispatching the message over the CAN bus.

When the PIB is done with processing the command sent out by micro A, it writes a status into an internal register, and signals the MSMB to advance to the “STS GEN2” state.

The “STS GEN2” state is similar to the “STS GEN1” state except for a field in the status register of the SGB that remains unpopulated in the “STS GEN1” state. This field captures the contents from the status register in the PIB. Also, the field for the state code is now updated to “serviced command”. The Address logic module in the SGB does not increment the address while in the “STS GEN2” state. The contents of the status register in the SGB are thus written to the same address in the status space.

From the “STS GEN2” state, the MSMB moves to the “INT GEN2” state which is identical in the sequence of operations carried out, to the “INT GEN1” state. When micro A receives an interrupt, it dereferences the particular command in the command space that has been serviced by extracting the command offset from the updated status. It evaluates the field for the state code and decides whether to clear the command enable bit to prevent against accidental re-execution. Eventually, the MSMB returns to the “CMD CCL” state.

When the CAN interface in the CCP associated with Micro B receives the

request, the LLC passes on the request to the RHB in the CCP, after filtering incoming messages. The RHB module is similar to the other support blocks in the CCP. However additional functionalities necessitate an embedded timer unit, and the capability to share the acknowledgement generation engine (AGE) in the PIB. This guarantees a response even if authorization is not received from micro B. As soon as a fresh request is received, the timer starts counting down. If the latency involved in the micro initiating the response exceeds the timeout period for the timer, the RHB, in collusion with AGE sends the appropriate acknowledgement on the CAN bus, and inhibits the command coming in from the micro as response.

If the MSMB is in the “CMD CCL” state and it receives a signal from the RHB indicating that a request is pending, MSMB advances to the “REQ HAN” state. In this state the address logic module in the RHB pushes the request along with the requisite qualifiers into the request space, as the MSMB advances into the “INT GEN 3” state. The “INT GEN 3” state is similar to the “INT GEN 1” state; however the IGB fetches the offset from the address generator module in the RHB and not from the SGB. Further, the additional field alluded to earlier, now contains a label for the RHB, not the SGB. After flagging the interrupt, the IGB signals the MSMB to return to the default “CMD CYCLE” state.

Meanwhile, micro B, receives the interrupt, and using the procedure detailed earlier, reads the correct address in the request space, and deduces that a request from micro A for a particular set of data has caused the interrupt. In case the requested data is available and can be shared, micro A arranges the data contiguously in a data space, and constructs the command for transmission of data, including a field for the particular data space, a field for the offset within that space, and a field for the number of bytes to be transmitted. It then pushes this command into the command space.

When this command is read by the CCP for micro B, it processes the command using the procedure in the foregoing description. (Depending on the number of bytes to be transmitted, the J1939 block might opt to use the transport protocol to split the transmission into multiple messages.)

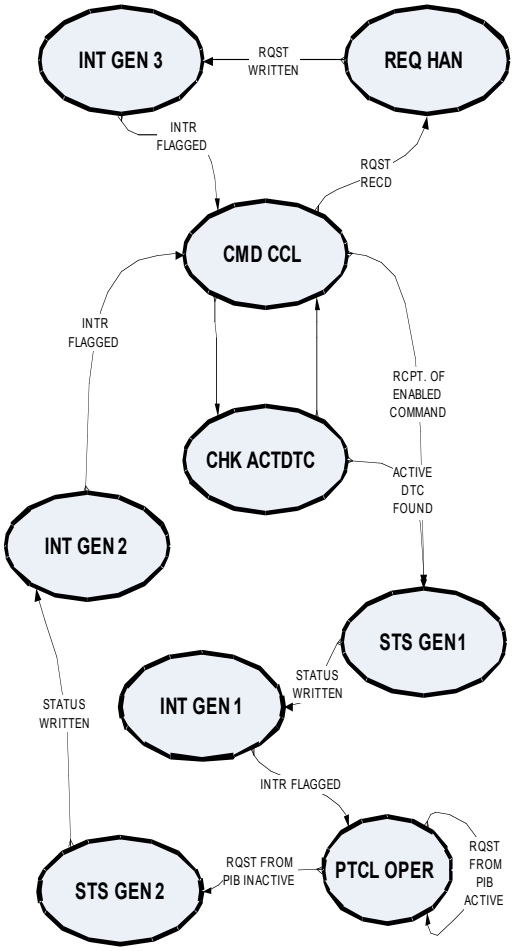

Figure 4: Master State Machine

Qualifying note to figure 4: The MSMB cycles between “CMD CCL” and “CHK ACTDTC” states by default. In the “CHK ACTDTC” state, the MSMB reads the first address in the Active DTC space to judge whether any active DTCs exist. If so, it populates the bank of field registers accordingly and moves to the “STS GEN1” state, wherein the status data register is populated with a proprietary data string predefined within the MSMB. If no active DTCs exist, the MSMB returns to the “CMD CYCLE” state.

5 Highlights

- The definition of the command allows for an enable bit which permits deletion of commands prior to processing.
- A separate status space allows for the microcontroller to interrogate the CCP without actually interrupting the CCP operation.
- The protocol block includes an interpreter to determine the context of the command and use the information in the field registers accordingly and invoke the appropriate constituent block in the PIB. Figure 3 shows some of the constituents of the PIB for the J1939 protocol.
- Modularizing the design will enable the individual blocks to be developed as objects which can subsequently be instantiated to realize the final design.
- The PIB is implemented as a separate block, so that different blocks corresponding to different protocols can be plugged in.
- Incorporation of a scan chain into the PIB will allow the contents of the state vector registers in the constituent sub – blocks of the PIB to be sampled and examined at run time. This will give a window into the internal working of the PIB to an extent that could not be possible in the traditional micro based approach.
- Since the CCP is implemented in hardware, the PIB operates concurrently and processes the command independent of the state transitions in the MSMB.

6 Conclusion

As with migration to any new approach, the CCP strategy will entail some tradeoffs. Adoption of this strategy will therefore need to be justified in the light of the following challenges.

- The protocol complexity shall dictate the selection of platform for the RCD. This will translate to an additional cost component for the hardware.
- Although this can be offset to some extent by selecting a lower cost micro, (not as resource rich and devoid of CAN interface) a change in hardware will be required. This change would have a trickle down effect, translating to higher initial cost.
- Adoption of the CCP approach will necessitate changes in the software structure. Depending on the protocol complexity and comfort level of software developers, additional time needs to be budgeted. This will adversely affect time to market in the short term.
- Given the ramifications of 1, 2, and 3, migration to the CCP strategy can presumably be justified in case of high volumes of production.
- However, this strategy can be adopted in new developments, where hardware designs can be tailored accordingly. This strategy affords a lot of flexibility to the designer in terms of lesser constraints in the selection of micro. Since no specific demands are placed on the application software, the designer also gets more leeway in the selection of an embedded OS, if that is deemed necessary.

It is believed that adoption of this approach will prove to be beneficial in the long run. The robustness of this approach should allow for easier debugging and faster design cycle times.