

# Transparent Distributed Messaging for Power Electronics Systems

Parool Mody and Stephen H. Edwards

Department of Computer Science  
McBryde Hall, Virginia Tech, Blacksburg, VA 24061

**Abstract** - This paper presents a distributed transparent messaging protocol for plug and play power electronics building blocks (PEBBs). The protocol provides for transparent messaging between processors across a network for any multi-controller application. The protocol works for an application with any number of controllers and processor allocation strategy, without any code change. The paper further includes an assessment of network parameters to determine system performance.

## I. INTRODUCTION

The protocols designed so far for power electronics systems are for single controller systems or multiple controller systems with fixed processor allocation. Single controller systems fail to use the advantages offered by distributed systems, which are improved efficiency and greater fault tolerance. Using multiple controller systems with fixed processor allocation severely restricts the flexibility and hence the usage of the system.

In this paper, we present a protocol for transparent inter-processor communication across a network thereby allowing transparent distribution of any multi-controller application. The protocol is designed such that it can run the same application without any kind of code change in virtually any kind of distributed configuration, where configuration is the number of controllers used in the system plus the processor allocation strategy used. The protocol works well even for single-controller applications and for pre-defined allocation of processors to controllers. The protocol, thus offers a lot of flexibility and ease of use in running a multi-controller application and evaluating its performance using different number of controllers and/or processor allocation strategy. The protocol also enables an application, with an automated processor allocation strategy, to transparently configure itself for any number of processor nodes without requiring any changes or recompilation.

The following section gives a brief introduction to the dataflow architecture. Section III provides the design and implementation of the messaging protocol. Section IV gives an analysis of the system performance. Section V presents a brief summary of the work to be done in future to improve protocol performance. Section VI concludes the paper.

---

Dataflow architecture is a software architecture used to design software for plug and play power electronics building blocks. It is a data-driven architecture consisting of a large number of program elements to support component-level design. The embedded system being designed consists of a number of Elementary Control Objects (ECOs), which are concurrently executing entities. The ECOs are connected to each other through data channels. The ECOs read data from the data channels, process the data and generate the output.

The ECOs are scheduled by their firing rules. Firing rule for an ECO indicates the input channels on which the ECO should wait for data before being fired. A read on a data channel can unblock an ECO waiting to write data into that data channel. Similarly, a write operation can fire an ECO waiting for input on that data channel. The execution of the ECOs is managed by the DARK Operating system [2].

The application programmer provides a DFG (Dataflow graphs) descriptor file, which contains information on the number of ECOs present in the system and the data channels that connect them. The application programmer also provides implementation of the ECOs.

The protocol makes use of the existing dataflow architecture to provide for transparent message passing between ECOs present on different controllers.

## III. DESIGN AND IMPLEMENTATION

Based on the number of controllers available, the system automatically allocates ECOs to different controllers. The controllers will be placed on a ring, which operate based on a protocol called PESNET [1]. The ECOs communicate asynchronously with each other by reading or writing data into the data channel. They are not aware of the number of controllers present in the system and hence of the distributed nature of the communication.

The protocol needs to ensure that communication between ECOs on different controllers is carried out transparently. This necessitates orderly arrival of messages and special handling of loss messages.

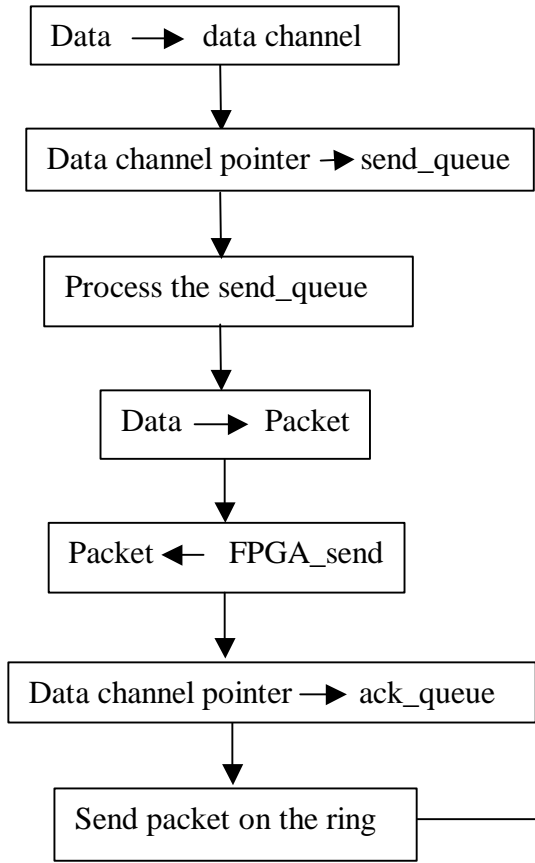


Fig. 1. Packet send protocol

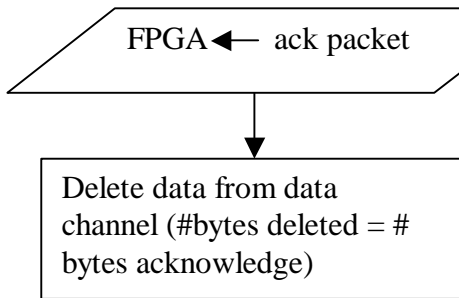


Fig. 3 Acknowledgement received protocol

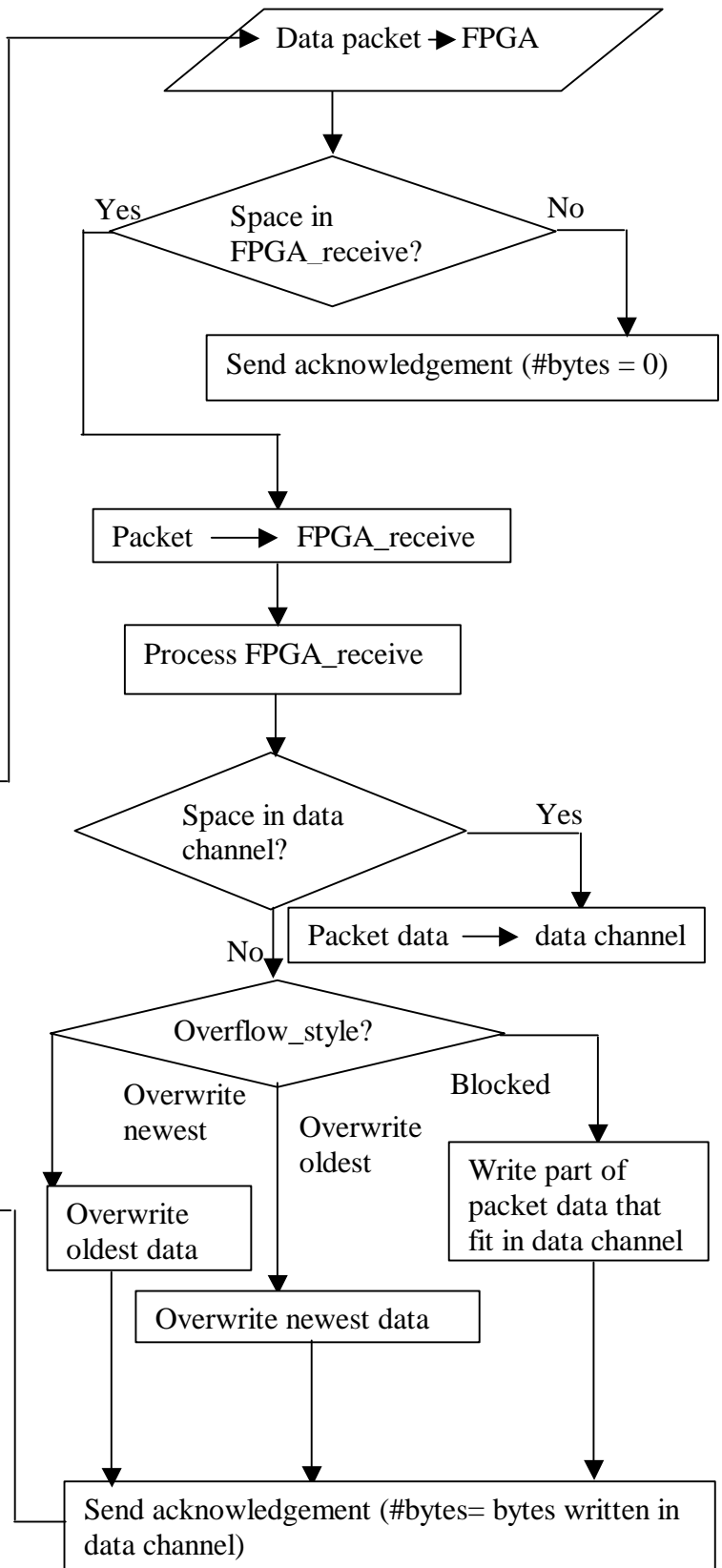


Fig. 2. Packet Receive protocol

A. Design

Fig. 1 describes the protocol for sending a data packet from the source ECO to a destination ECO, where the ECOs are on different controllers. The ECOs on a single controller communicate with each other by reading or writing data from data channels. Distributed communication also occurs through data channels. In case of distributed communication, the source ECO writes data to a distributed data channel. The OS processes the data in the data channel, packs it into packet and passes the packet onto the FPGA. Note that the OS does not yet remove the data from the channel. The packet is then sent onto the ring by the FPGA.

Fig. 2 describe the protocol on the receiver side. When the FPGA receives a data packet, it stores the packet, to be later processed by the OS. The OS extracts data from the data packet and writes it to the data channel identified by the packet. It then prepares an acknowledgement packet acknowledging the number of data bytes written to the data channel. The acknowledgement packet is then passed to the FPGA. The FPGA sends the packet over the ring.

Fig. 3 explains the protocol when an acknowledgement packet is received. The OS deletes the data from the data channel based on the number of bytes acknowledged.

B. Data Structures

The protocol makes use of two circular buffers – the send\_queue and the ack\_queue. A send\_queue entry points to a distributed data channel that has data to send. An ack\_queue entry points to a distributed data channel that is awaiting an acknowledgement for the packet sent. Note that the send\_queue and the ack\_queue together will contain not more than one entry corresponding to each distributed data channel. The size of the send\_queue and ack\_queue is equal to the number of distributed data channels in the system.

The status of a data channel is indicated by the value stored in the alloc\_type field of the data channel. The alloc\_type field of a distributed data channel can have one of the three values

1. WAITING\_TO\_SEND when the data channel contains data to be sent across the ring
2. SENT when a packet has been sent and an acknowledgement is awaited
3. EMPTY when the data channel is empty and is not waiting for an acknowledgement.

The alloc\_type field of a normal data channel will always have value NULL.

The FPGA uses two fixed size buffers - FPGA\_send and FPGA\_receive. The FPGA\_send stores packets to be sent on the ring while the FPGA\_receive stores packets received from the ring. Data structures for the protocol are as shown in Fig. 4. Note that the size of the send\_queue and ack\_queue is equal to the number of distributed data channels.

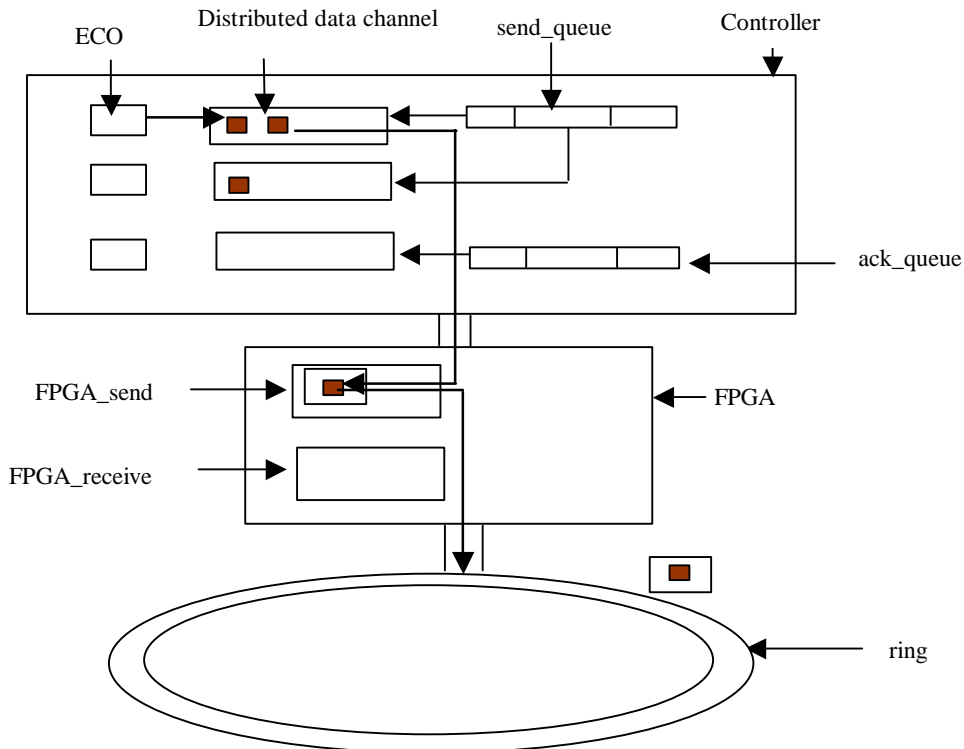


Fig. 4. Data Structures

### C. Implementation

The sender's side protocol is described by the dotted lines in Fig. 4. When an ECO writes data into a distributed data channel with `alloc_type` field as `EMPTY`, a pointer to the data channel is stored in a `send_queue` entry. The `alloc_type` field of the data channel is changed to `WAITING_TO_SEND`.

When the operating system scheduler is called, it checks for entries in the `send_queue`. If the `send_queue` is not empty and there is space in the `FPGA_send` buffer, the scheduler reads data from the data channel pointed to by the `send_queue` entry writes it into a packet and then stores the packet in the `FPGA_send` buffer. A pointer to the data channel is removed from the `send_queue` and added in the `ack_queue`. The `alloc_type` field of data channel is changed to `SENT`. Note that the data is not yet removed from the data channel.

Fig. 5 shows the packet structure. The packet contains addresses of the source and destination controllers. The `datachannel_id` field is used to uniquely identify the data channel and thereby the ECOs associated with the data channel.

```
typedef struct
{
    Net_Address from_address    : 8;
    Net_Address to_address     : 8;
    char datachannel_id        : 16;
    packet_command command     : 4;
    unsigned int number_of_bytes: 4;
    char data [9];
}
Digested_Packet;
```

Fig. 5. Packet structure

The command field is used to indicate the packet type. The command field can have one of the two values -

1. `data_packet`
2. `ack_packet`

For a data packet, the `number_of_bytes` field indicates the number of data bytes contained in the packet. The `number_of_bytes` field for an acknowledgement packet indicates the number of data bytes acknowledged by the receiver. 4 bits have been allocated for the command field to provide for future improvements.

When the FPGA gets an empty token on the ring, it grabs the token and passes the data packet onto the ring.

The FPGA on the receiver side removes the packet from the ring and stores it in the `FPGA_receive` buffer, if there is space in the buffer. If the `FPGA_receive` buffer is full, the FPGA sends out an acknowledgement packet acknowledging zero bytes of data.

The operating system scheduler checks the `FPGA_receive` buffer for any incoming packets. If the receive buffer is not empty, it reads the packet and checks if there is space in the data channel identified by the packet. If there is space for the entire data packet, then the data packet is written into the data channel. Otherwise, the scheduler overwrites the oldest data element or the newest data element or writes part of the data that fit into the available space in the data channel. These actions are based on value of the `Overflow_style` field of the data channel. In all cases, an acknowledgement packet is sent back acknowledging the number of bytes written into the data channel.

When an acknowledgement packet is received, the number of bytes acknowledged determines the number of bytes to be deleted from the data channel. The pointer to the data channel in the `ack_queue` is then removed.

### D. Fault Tolerance

In order for the messaging to be transparent, the design needs to ensure that packets arrive in order. Orderly arrival of packets is ensured by requiring every packet to be acknowledged and the next packet for a given data channel be sent only after an acknowledgement is received for the previous packet.

When a packet is sent over the network, a copy of the data is stored on the sender's side and is deleted only after an acknowledgement for the data is received. If there is not enough space for the entire packet data in the data channel on the receiver side, it is possible that only part of the data gets written into the data channel and hence only part of the data gets acknowledged. In that case, only those data bytes that are acknowledged are deleted and an attempt is made to re-send the unacknowledged data bytes.

For packets lost due to node or ring failure, the protocol relies on the underlying PESNet protocol to ensure fault tolerance. The PESNet protocol makes use of a dual, counter-rotating fiber optic rings to improve the fault tolerance of a network. In case of a node or a link failure, the bi-directional ring allows the message can backtrack. Thus, the PESNet protocol ensures that there won't be any missing packets irrespective of a node or a link failure.

#### IV. ANALYTICAL PERFORMANCE ASSESSMENT

An analysis of the system performance is performed based on factors such as network speed, number of nodes on the ring and saturation of the network.

Let us consider a network of  $N$  nodes. Let  $P$  be the size of a packet in bits and  $R$  be the transmission rate of the network. Then, the time required for a complete cycle of a packet that is the time between the transmission of a packet by a sender, its processing at the receiver and the return of an acknowledgement packet from the receiver back to the sender is given as

$$T_{\text{cycle}} = T_{\text{process}} + T_{\text{send}} + T_{\text{ack}}$$

where,

$T_{\text{cycle}}$  = Time to transmit packet from sender to receiver

$T_{\text{process}}$  = Time to process the packet at the receiver

$T_{\text{ack}}$  = Time to transmit the acknowledgement from the receiver back to the sender

The time to transmit a packet from sender to receiver depends on the time to send a packet over a single network link, the number of hops between sender and receiver and the saturation of the network. Hence,

$$T_{\text{send}} = T_{\text{sat\_delay}} + \text{Distance} \times T_{\text{packet}}$$

where,

$T_{\text{packet}}$  = time to send a packet over a single link,

$T_{\text{sat\_delay}}$  = Delay due to network saturation for a single packet and

Distance = # of hops between sender and receiver.

The network can be considered as divided into slots of data packets. A packet can be sent over the network only at the start of a packet slot. Since the time to process a packet at the receiver is very small, it can be safely assumed that the processing time is equal to length of a single packet slot and hence equal to the time to transmit a packet over a single link.

$$T_{\text{process}} = T_{\text{packet}}$$

The time to transmit an acknowledgement packet can be given as

$$T_{\text{ack}} = T_{\text{sat\_delay}} + (N - \text{Distance}) \times T_{\text{packet}}$$

Hence, the total cycle time can be written as

$$T_{\text{cycle}} = 2 \times T_{\text{sat\_delay}} + (N + 1) \times T_{\text{packet}}$$

If  $s$  is the saturation index of the network with value between 0 and 1, then the saturation delay can be given as

$$T_{\text{sat\_delay}} = 1 / 2 \times s / (1-s) \times T_{\text{packet}}$$

If  $T_{\text{delay}}$  is the delay introduced by each node in the network, then

$$T_{\text{packet}} = P / R + T_{\text{delay}}$$

Hence, the time for a complete cycle can be given as

$$T_{\text{cycle}} = (N + 1 + s/(1-s)) \times T_{\text{packet}}$$

To provide a basis for concrete discussion, we consider an example application with two controllers and 6 phase legs. The controllers are switching at a frequency of 20KHz. As there are 2 controllers and 6 phase legs, the number of nodes,  $N$  in the network is 8. Let us assume that the network saturation is 25% that is value of  $s$  is 0.25. As 8 data bits get transmitted as 10 bits due to 4B/5B encoding by the transceivers, a packet of size 16 bytes will give a value of  $P$  as 160 bits. If the network speed  $R$  is 100Mbytes per second, and the delay introduced by each node,  $T_{\text{delay}}$  is 3 nanoseconds, then the total cycle time will be around 15 microseconds.

Thus, it will be possible to perform 3 such cycles in a single switching period of 50 microseconds.

#### V. FUTURE WORK

The protocol is still in its experimental stage, with its performance on the controller board yet to be determined. Although, the controller board is ready, the implementation of the PESNet protocol on the controller board is not yet done.

Testing on the controller board will involve determining the time for a complete cycle of a packet on the controller board and comparing it with the empirically evaluated value. Testing will also involve comparing the actual number of cycles that can be completed in one switching cycle and the empirical estimate of the same. Testing will provide valuable feedback on steps to be taken to improve performance.

Further research on PESNet is on developing a set of communication protocols called the PESNet Interface (PESNI) that specify the physical, media-access and network layers of the OSI reference model. The PESNI will consist of 3 specifications, physical layer protocol (PLP), link layer protocol (LLP) and network layer protocol (NTP). These will serve to standardize the protocol, provide improved fault tolerance and will also reduce the round trip time by the use of routing methods. It would be interesting to measure the performance of our messaging protocol on the improved version of PESNet.

In the future, work will be done on implementing the protocol for controllers physically stacked together. The

performance of the protocol will then be evaluated. As there won't be any delay over the ring, the performance is expected to be much better than the one observed over the ring.

## VI. CONCLUSION

In this paper, we have made an attempt to facilitate the use of variable number of controllers in power electronics systems. The protocol allows for systems to have either static or a dynamic allocation of processors to controllers. Testing on the controller board will provide insight into the efficiency of the protocol and further steps to be taken to improve the performance.

## ACKNOWLEDGEMENT

This work was supported primarily by the Office of Naval Research under Award Number N00015-01-1-0954 and secondarily by the ERC Program of the National Science Foundation under Award Number EEC-9731677.

## REFERENCES

- [1] "Protocol Design of Dual Ring PESNet (DRPESNet)" CPES 2002 Power Electronics Seminar and NSF/Industry Annual Review, April 2002.
- [2] "DARK: Designing a high performance micro-kernel for power electronics controllers" CPES 2002 Power Electronics Seminar and NSF/Industry Annual Review, April 2002.