Embedding it better...

µTasker Technical Note

µTasker – Ethernet Reception

Table of Contents

1. Introduction

This technical note discusses the strategies used to receive Ethernet frames in the µTasker project.

2. Ethernet Frame Reception

All devices supporting Ethernet use an Ethernet controller that autonomously handles the reception of individual frames, including saving the frame content to local memory. This needs to be the case due to the speed of reception so that no bytes of data in the frame can be lost under normal conditions. Some devices use dedicated memory, possibly realised in form of a FIFO, and others may use DMA to save the content directly to shared memory with low impact to the processor's operation.

When a valid frame has been fully received it is necessary for the application (in this case it may be TCP/IP protocol stack software) to handle its content. This is invariably triggered by an interrupt on the Ethernet frame reception.

3. Scheduling Strategies

In most cases it is not a good idea to handle the Ethernet receptions directly in the reception routine for a number of reasons:

1) The TCP/IP software may already be handling a previous reception frame and so it would not be appropriate for a newer reception frame to be handled before the previous one has been completely treated. *Single interrupt types would however block further interruptions of the same type before the first is completed*.
2) Depending on the protocol involved and its treatment and responses it can take quite a lot of processor time to handle the protocols involved and it is undesirable to block other interrupt during this time.
3) When there is a high Ethernet load it may not be desirable to allow interrupt priority processing since it would tend to steal processor time. A method of controlling the processing rate may be more appropriate.

Therefore the basic strategy is to provoke the scheduling of the Ethernet task when an interrupt arrives. The treatment of the received Ethernet frame is therefore delayed until the scheduler activates the Ethernet task, during which time the particular Ethernet buffer is possessed by the processor (rather than by the Ethernet controller) and is therefore blocked for further reception. Ethernet controllers usually have a number of Ethernet buffers which can be used (often configurable by the user) and additional frames can be received into these free ones until all become blocked, in which case – *if it ever happens* – the reception will be discarded by the Ethernet controller. When a reception frame is lost due to there not being any further available buffers there may be some indication of the fact (via an interrupt [if enabled] or an event counter).

Once an Ethernet frame has been fully processed it is freed by the Ethernet task which calls the routine `fnFreeBuffer()`. A freed buffer then belongs to the Ethernet controller, which can use it for subsequent reception frame buffering.

3.1 Default Scheduling

The default scheduling strategy (when no options enabled) is for the Ethernet reception input handler to send an interrupt event to the Ethernet task. This event is typically `EMAC_RX_INTERRUPT` but there may be different events depending on which buffer the frame has been received to (eg. processors with a limit number of fixed buffers).

The interrupt event is put to the input queue of the Ethernet task, which will then be woken due to this. The Ethernet task reads its input buffer and extracts an event for each Ethernet frame that is waiting.

This technique also allows other Ethernet related events (such as event reporting certain errors) to be sent to the task and processed there instead of `EMAC_RX_INTERRUPT`.

It is however not imperative to have an interrupt event for each waiting buffer since a single `EMAC_RX_INTERRUPT` event causes as many waiting frames to be processed as there presently are.

The Ethernet task has its input queue dimensioned to be able to accept approximately the maximum number of interrupt event that are possible. The length of the queue [in the task table] is also chosen to be an exact multiple of the length of interrupt events so that a dropped interrupt event (due to lack of space in the queue) would not cause a partial message to result; eg. `(HEADER_LENGTH * 12)`

3.2 `ETHERNET_RELEASE_AFTER_EVERY_FRAME` Scheduling

When the option `ETHERNET_RELEASE_AFTER_EVERY_FRAME` is enabled the Ethernet task doesn't use an input queue but instead each Ethernet reception interrupt simple causes the Ethernet task to be scheduled. The Ethernet task therefore assumes that there is a reception buffer to be read and will try to do this.

In addition, it will only handle one single waiting frame before returning from the task. When it leaves the task it re-schedules itself once (assuming that it did have a reception frame to handle) which ensures that it then immediately handles further waiting buffers in case the Ethernet reception had scheduled the task a multiple number of times, whereby it is to be noted that the number of times that it was scheduled is not known in this case since there is not an interrupt event for each one.

As soon as the task doesn't find a further waiting Ethernet reception buffer it returns and will be scheduled again only when there is a further reception frame interrupt.

The advantage of this technique over the default strategy is that only one single reception buffer is treated at a time and, in between each processing, other tasks can be scheduled.

This technique doesn't allow other Ethernet interrupt events to be sent to the Ethernet task since the task no longer has an input queue and expects only reacts to input buffer ready events. This is however not a general problem since such events tend to be used for event counting and can usually be handled directly in the interrupt routine or even ignored.

3.3 ETHERNET_RELEASE_AFTER_EVERY_FRAME Multiple-Pass Scheduling

This option is based on the `ETHERNET_RELEASE_AFTER_EVERY_FRAME` option (which also needs to be enabled) but allows the Ethernet task to handle more than one Ethernet frame in each pass. The value `ETHERNET_RELEASE_LIMIT` can be set to define the maximum number of Ethernet reception frames that are allowed to be handled before the task returns in the same manner as described in the previous section.

If there are not the maximum number of frames actually waiting the task will still return before the defined maximum number are handled; the value is a limit in order to avoid the task from processing too many frame in a very busy network without allowing other tasks in the system to be scheduled.

4. Conclusion

This technical note has described the default method of handling Ethernet reception frames by the Ethernet task plus additional scheduling options that are available to tune the behaviour to best match the intended environment.

Modifications:

V1.00 25.08.2011 - Initial version