Analysis and Improvements of FW download process

# Initial Status

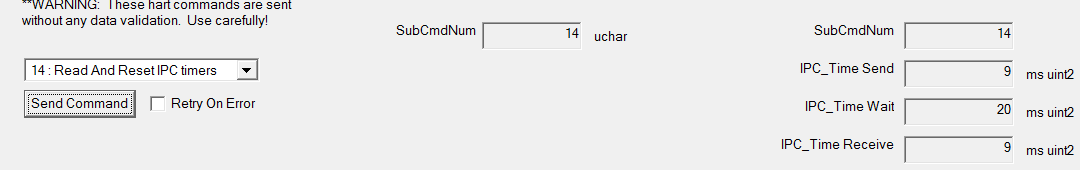
Instrumentation of FFP firmware (C61600) made it possible to measure

* Worst send/wait/receive times of IPC messaging (i.e., low level)
* Worst time between sends, worst time between receives and worst roundtrip time for the critical command 177. Note that roundtrip time is calculated from just before preparing the data payload to just after processing a response, so unlike wait time in IPC messaging, it accounts for processing time, including any task preepmtions.

All measurements are done with

* MAX\_RESPONSE\_DELAY (MRD) 4
* RB in AUTO
* APP in Failsafe (for clean measurements)
* No FBAP
* NI Configurator connected and updates RB and TB every 2 s
* SA connected over FF to read statistic sporadically

Typical timing is shown in Fig. 1. Note that an error can be as large as 1-3 ms.



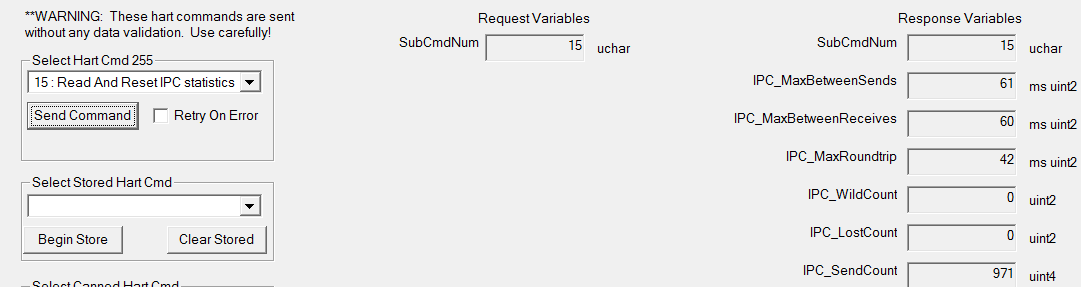
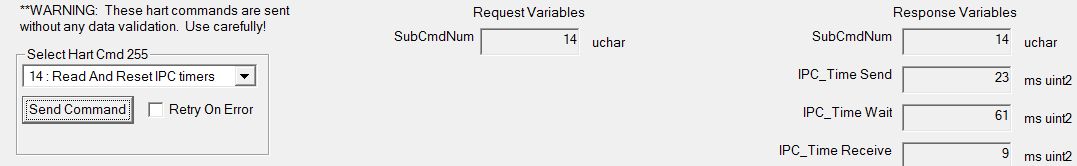


Fig. 1 Typical steady-state timing measured in C61601

During firmware download, the timing changes as shown in Fig. 2.



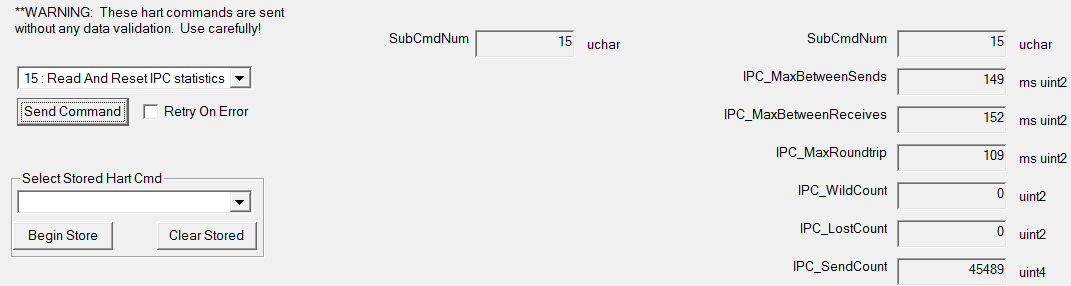


Fig 2. Firmware download timing measured in C66101

Download itself takes about 12:05 min.

## Analysis

Comparing Fig. 2 with Fig. 1, we can observe that IPC Time Send is increased (by 14 ms). There are at least two reasons for that:

1. Longer IPC messages are sent
2. IPC HART task may be preempted between taking a timestamp and starting the transmission.

IPC Time wait is increased by 41 ms, which is explained by writing to flash with IPC command 245.

Combined expense of 41+14=55 ms is how much we can account for in IPC timing, which would bring us to worst timing  
Between Sends=149-55 = 94 ms  
Between Receives=152-55 = 98 ms  
Roundtrip=109-55 = 109-55 = 44 ms

The adjusted roundtrip would be in line with the steady state and the 33-37 ms increase in sends and receives remains unaccounted for, except for task preemptions.

## Discussion

Firmware download over ISP takes about 7:15 min. The time is important not only for the future user experience, but for the testers as well. With longer FF download, they would prefer ISP download, and FF download might not get sufficiently exercised.

More importantly, there are adverse effects of firmware download on real-time control. The times between sends, between receives, and roundtrip time increased more than two-fold. The setpoint will not be delivered as crisply, and for a fast macro cycle, feedback will be delayed, which may lead to limit cycling. This is contrary to the assumption that firmware download is transparent to the control process.

Therefore, it is desirable to improve both IPC timing and overall download time.

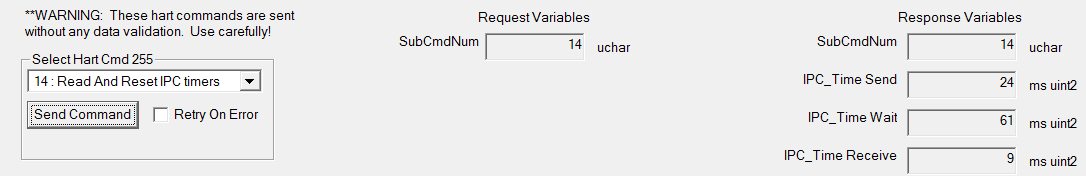
An obvious approach is to write FFP and APP flash asynchronously with FF download messaging, and return the last known result for the response code. However, writing to flash can’t fall too far behind because the response to the last message (SWDL\_LOAD\_LAST\_SEGM) must come on time. So, we want to fall behind by at most one message.

# Optimization Steps

## Interleaving messaging and writes in FFP

In this step (C61601), I implemented immediate response to FF message, and actual work is done in the process context.

The overall download time is 7:55 min. IPC statistics are shown in Fig.3



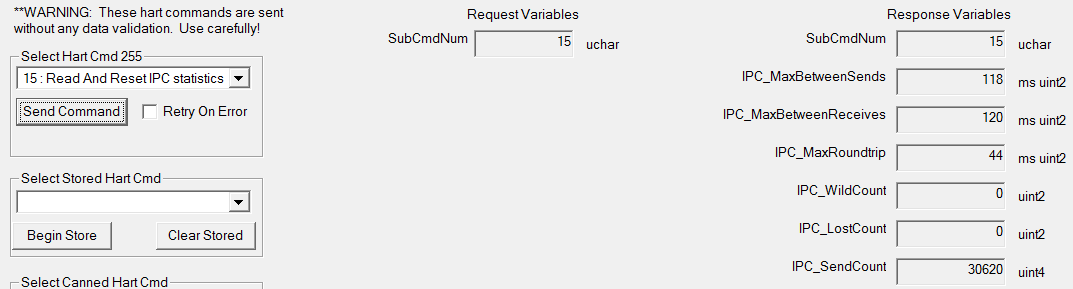


Fig. 3. Interleaving in FFP

Remarkably, IPC max roundtrip is brought to the level of steady state, as it would be expected always.

The combined expense is now measured to be 57 ms over steady state, and adjusted worst times are  
Between Sends=120-57 = 63 ms  
Between Receives=118-57 = 61 ms  
That is entirely in line with steady state.

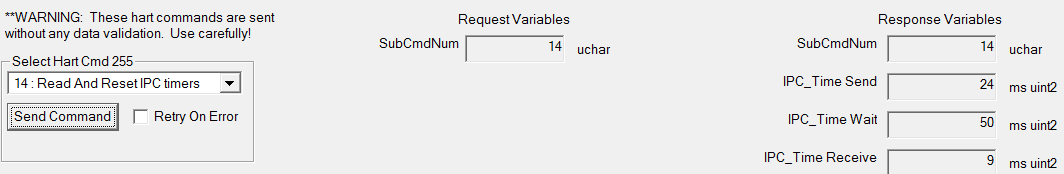
The next savings could come from reducing IPC time wait: We can’t do much with IPC time send because we do send long messages.

## Interleaving messaging and writes in APP

### Preliminary step

This easy step (C61629), APP’s IPC task sends an immediate response and continues to writing flash if there is something to write

The statistics are shown in Fig. 4.



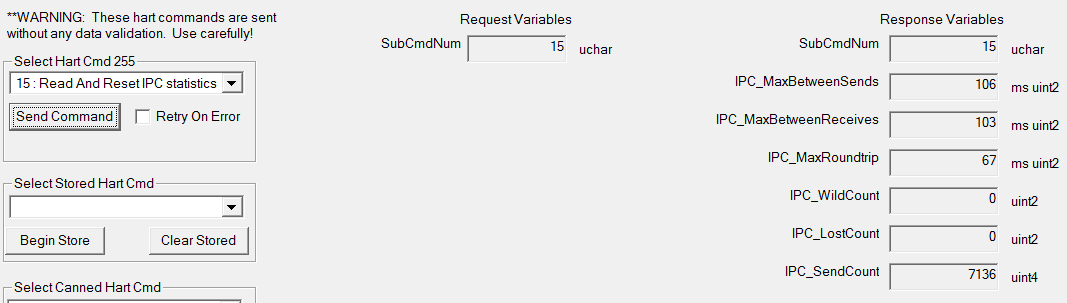


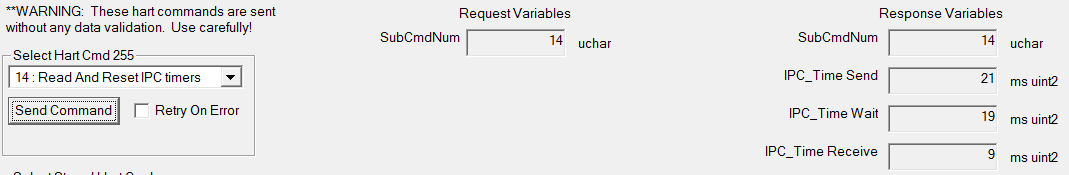
Fig. 4. Lightweight interleaving in APP

We observe that IPC wait time is reduced somewhat but IPC roundtrip time got worse. This is explained by deferred write in APP in the same context as IPC data exchange processing thus delaying the latter.

### A better strategy in APP

In this step (C61674), I implemented immediate response to IPC command 245, and actual work is done in a newly created task with priority below IPC and above process.

The statistics are shown in Fig.5.



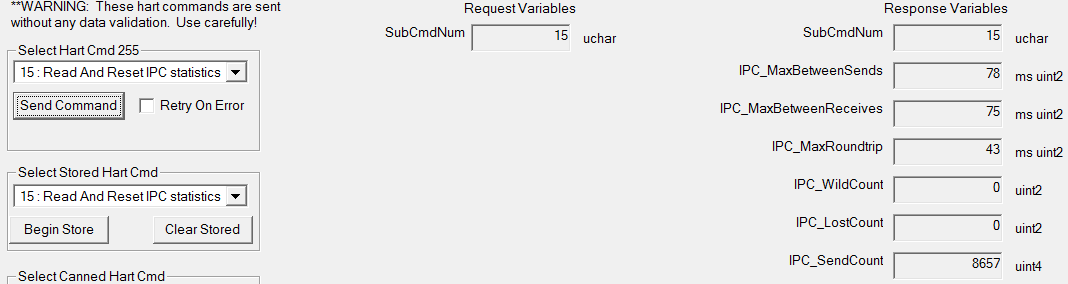


Fig.5. Interleaving in APP in a separate task

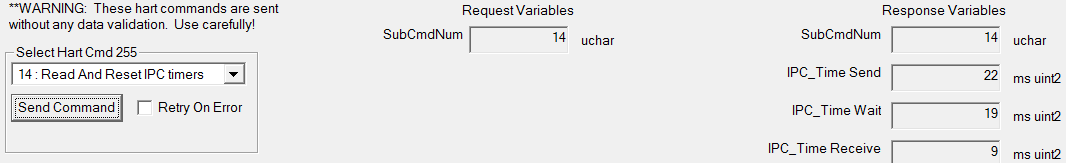
With ~10 ms penalty in IPC timing (attributed to long flash write command), the IPC roundtrip time is entirely in line with steady state, and adjusted times between sends and between receives (68 ms and 65 ms) are almost inline and the difference can be attributed to longer preemptions by FDC task during firmware download

## Further optimization in FFP

Inspired by improvement brought by APP proxy task, I tried to move FFP deferred processing to a separate proxy task (Shelveset entitled “further fwdl improvements (unfinished)”).

This is an attempt to boost the priority and speed of deferred processing of FW download in FFP. It is a slightly faster overall download but worse than the original (C61674) w.r.t. IPC period.

The Fig. 6 shows overall results slightly worse than FFP deferring to Process task. This is surprising but could be possibly explained by increased RTOS overhead of additional events processing



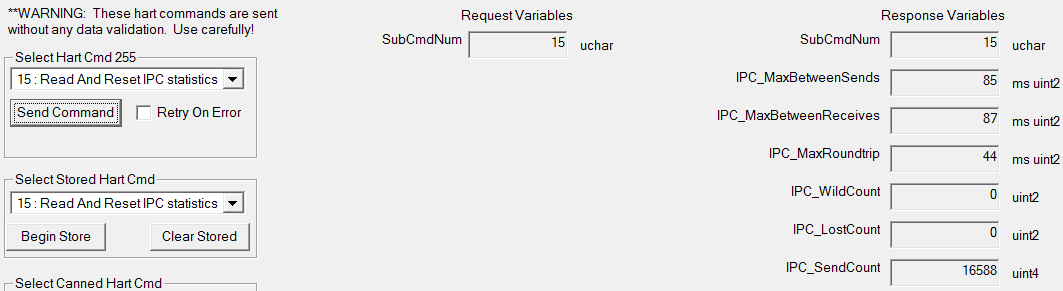


Fig. 6. Timing with FFP deferred processing in Appl task.

# Conclusions

Further improvements could include tinkering with IPC HART task priority and splitting that task into scheduler and processor. The reason is that what remains of “HART Master” in FFP is a scheduler which is or can be made very fast, and a relatively slow data preparation and parsing for periodic commands.

Perhaps that would be a dead-end strategy vs. a better solution would be to have a separate channel (USART4) for IPC.

## Stability

At MRD=4, I saw one failure of the original inline download.

In any case, there are HART-over-FF commands provided for switching between the original inline and the new interleaved (default) methods separately for FFP (255.17 read, 255.18 write) and APP (129.17 read, 130.17 write).

## Other

As a cherry on top, I implemented HART-over-FF command 255.251 to interrogate percent completion. Could be plugged in DTM!