

Appendix

Data-driven Mutation Testing: LuxSpace ADCS Case Study

This Appendix describe the procedures adopted to execute data-driven mutation testing on the LuxSpace ADCS case study system. Section 1 provide a detailed overview of the case study and the function targeted by data-driven mutation testing. Section 2 describes the fault models defined for the case study. Section 3 describes the integration of mutation probes into ADCS_IF_SW.

1. Overview of the case study

Data-driven mutation testing is applied to assess the quality of the test cases that exercise the ADCS software interface of the ESAIL system (hereafter, ADCS_IF_SW). In ESAIL, the ADCS_IF_SW is used to manage and collect data from hardware devices (e.g., sensors). Detailed specifications for the ADCS interface appear in the document *ESAIL-LXS-ICD-P-0184 ADCS IF SW External ICD*.

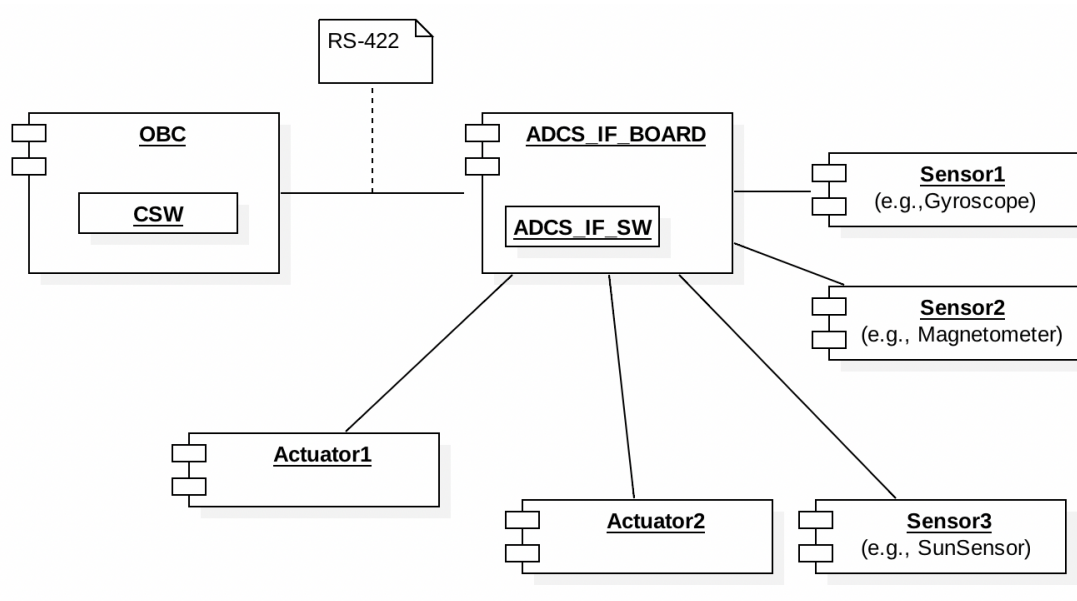


Figure 1: OBC-ADCS integration in ESAIL

Figure 1 provides an overview of the integration between ESAIL OBC and the ADCS board. ESAIL CSW (central software) runs on an onboard controller (OBC) with a Leon 3 microprocessor. The OBC is connected to ADCS interface boards (ADCS_IF_BOARD) through RS-422. The ADCS_IF_BOARD runs its own controller (ADCS_IF_SW). Each board processes data received from sensors and controls actuators. The ADCS_IF_SW is the

target of data-driven mutation testing and is the software layer where mutation probes are installed.

The `ADCS_IF_SW` implements functions used by the OBC to send data to devices (i.e., set their configuration) and functions that send devices data to the OBC.

The function of the `ADCS_IF_SW` that manages the communication between the ADCS and the OBC, i.e., *ObcRecvBlockCb*. The function is implemented in file *AdcsIf.c*.

The SVF simulator used for testing runs the OBC software but it simulates the behaviour of the `ADCS_IF_SW`. The `ADCS_IF_SW` is not executed inside the SVF but only simulated. The ESAIL system test suite contains test cases that exercise the integration between OBC and the `ADCS_IF_SW` but the `ADCS_IF_SW` is not actually run. The test suite that exercises the `ADCS_IF_SW` is one that should execute with hardware in the loop.

Since the functions that send data to the devices are tested with hardware in the loop, in the context of FAQAS, we will apply data-driven mutation testing only to verify the functions used by the ADCS to send data to the OBC.

Although in principle also messages from the OBC to the `ADCS_IF_SW` could be tested, the current test suite, which does not run the `ADCS_IF_SW` prevents it. Indeed, the simulator used in the current test suite makes assumptions about the messages received thus it would be very easy to break it by altering its input messages. To alter the messages sent from OBC to `ADCS_IF_SW` it would be necessary to (1) use a simulator that actually runs the `ADCS_IF_SW` or (2) target the test cases that include hardware in the loop.

Case (2) above, i.e., testing with hardware in the loop, is technically feasible because it is just a matter of deploying on the hardware a modified software that performs the mutation. However mutated packets may break some of the assumptions made when developing the software and thus break the hardware (e.g., altering the voltage of the board). For every mutation to be performed it might be necessary to ensure that the hardware is not going to be damaged. Such type of testing might thus be out of the budget for the current project and may require a dedicated project by itself.

The implementation of function *ObcRecvBlockCb* is shown in Section 1.1. It mainly consists of a switch command (line 138) that generates a response for the OBC after invoking a *data generation method* selected according to the request received on the data link. For example, Line 146 invokes method *GetIfStatus*, which prepares a response packet containing the information about the ADCS status.

Each *data generation method* receive as input an object of type *std::vector* (i.e., the object *newBlock*) that will be used to store the data to be sent to the OBC. The vector *newBlock* acts as a buffer; it contains elements of type *UInt8*, the length of the vector matches the size of the response message indicated in *ESAIL-LXS-ICD-P-0184* (*one element per byte*). Table 1 reports, for each feature targeted by data-driven mutation testing, the page in *ESAIL-LXS-ICD-P-0184* that describes the data format, the `ADCS_IF_SW` function that fill the content of message, and the size of the response message (i.e., the length of *std::vector*).

Table 2: Features targeted by data-driven mutation testing and message size

ADCS Feature	Page	ADCS_IF_SW function	Message size (bytes)
ADCS IF Status	19	GetIfStatus	6
ADCS IF HK	22	GetIfHk	37
GYTM - Gyroscope TM	34	GetGyroTm	21
MMTX - Magnetometer TX	41	GetMgtmTm	2
Sun Sensor TM	42	GetSsTm	48
SSTP - Sun Sensor Temperature	45	GetSsTemp	12
Reaction Wheel TX	50	GetRwTm	2
SpaceCraft HK	60	GetIfSchk	18
Magnetorquer Set PWM RSP	57	GetMgtqTm	39

Each invocation of a *data generation method* generates a response (i.e., the vector *newBlock*) that may either contain the desired result or an error code. The response generated in the first case is referred to as *nominal response message*, the response generated in the second case is an *error response message*. The response message is sent to the OBC through the invocation of function `SendResponse` (Lines 298 and 312). When an error code is generated, the data generation method returns *CR_Failure*. The response code is read by function *ObcRecvBlockCb* to determine if it is necessary to trim the buffer before sending back to OBC; this behaviour is handled by the parameter *true* passed to `SendResponse` (Line 312).

1.1 Function ObcRecvBlockCb

```

89 // --OPENING ELEMENT--AdcsIf::ObcRecvBlockCb--
90 /// Function that is called when a block of data is received from the data link layer.
91 /// @param block The received data block.
92 void AdcsIf::ObcRecvBlockCb(const std::vector<Smp::UInt8>& block)
93 {
94     // MARKER: OPERATION BODY: START
95     Trace(4, "Received command: 0x%s", OhbCommon::ByteUtils::BinToHex(block));
96
97     if(!CheckRxEnabled())
98     {
99         return;
100     }
101
102     std::vector<Smp::UInt8> newBlock(block);
103
104     Smp::UInt8 cmdId = block[0];
105     Smp::UInt8 subcmdId = block[1];
106
107     if(forcedResponse && (forcedResponseCmdId == cmdId || forcedResponseCmdId < 0)
108        && (forcedResponseSubcmdId == subcmdId || forcedResponseSubcmdId < 0))
109     {
110         newBlock.resize(2);
111
112         if(forcedResponseErrorCode >= 0)
113         {
114             // Generate forced error response
115             Trace(2, "Generating forced response message with error code 0x%02X", forcedResponseErrorCode);
116             newBlock.push_back(forcedResponseErrorCode);
117             SendResponse(newBlock, true);
118         }
119         else
120         {
121             // Generate forced valid response
122             Trace(2, "Generating forced response message with data 0x%s", forcedResponseData.c_str());
123             for(unsigned int i = 0; i < forcedResponseData.length(); i += 2)
124             {
125                 std::string byteString = forcedResponseData.substr(i, 2);

```

```

126         newBlock.push_back(strtol(byteString.c_str(), NULL, 16));
127     }
128     SendResponse(newBlock, false);
129 }
130 return;
131 }
132
133 bool processed = true;
134 CommandResult cr = CR_Failure;
135
136 if(Status->ADRD || ((cmdId == 1) && (subcmdId < 3)))
137 {
138     switch(cmdId)
139     {
140     case 1:
141     {
142         switch(subcmdId)
143         {
144         case 0:
145         {
146             cr = GetIfStatus(newBlock);
147         }
148         break;
149         case 1:
150         {
151             cr = GetIfHk(newBlock);
152         }
153         break;
154         case 2:
155         {
156             cr = SetIfPower(newBlock);
157         }
158         break;
159         case 3:
160         {
161             cr = SetUnitStatus(newBlock);
162         }
163         break;
164         case 4:
165         {
166             cr = SetConfiguration(newBlock);
167         }
168         break;
169         case 5:
170         {
171             cr = LclRetrigger(newBlock);
172         }
173         break;

```

```

174         default:
175             Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
176             processed = false;
177         }
178     }
179     break;
180 case 4:
181     {
182         switch(subcmdId)
183         {
184             case 0:
185             {
186                 cr = GetGyroTm(newBlock);
187             }
188             break;
189             default:
190                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
191                 processed = false;
192             }
193         }
194     }
195     break;
196 case 5:
197     {
198         switch(subcmdId)
199         {
200             case 0:
201             {
202                 cr = GetMgtmTm(newBlock);
203             }
204             break;
205             default:
206                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
207                 processed = false;
208             }
209         }
210     }
211     break;
212 case 6:
213     {
214         switch(subcmdId)
215         {
216             case 0:
217             {
218                 cr = GetSsTm(newBlock);
219             }
220             break;
221             case 1:
222             {
223                 cr = GetSsTemp(newBlock);
224             }
225             break;
226             default:
227                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
228                 processed = false;
229             }
230         }
231     }
232     break;
233 case 7:
234     {
235         switch(subcmdId)
236         {
237             case 0:
238             {
239                 cr = GetSsTm(newBlock);
240             }
241             break;
242             case 1:
243             {
244                 cr = GetSsTemp(newBlock);
245             }
246             break;
247             default:
248                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
249                 processed = false;
250             }
251         }
252     }
253     break;
254 case 8:
255     {
256         switch(subcmdId)
257         {
258             case 0:
259             {
260                 cr = GetSsTm(newBlock);
261             }
262             break;
263             case 1:
264             {
265                 cr = GetSsTemp(newBlock);
266             }
267             break;
268             default:
269                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
270                 processed = false;
271             }
272         }
273     }
274     break;
275 case 9:
276     {
277         switch(subcmdId)
278         {
279             case 0:
280             {
281                 cr = GetSsTm(newBlock);
282             }
283             break;
284             case 1:
285             {
286                 cr = GetSsTemp(newBlock);
287             }
288             break;
289             default:
290                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
291                 processed = false;
292             }
293         }
294     }
295     break;
296 case 10:
297     {
298         switch(subcmdId)
299         {
300             case 0:
301             {
302                 cr = GetSsTm(newBlock);
303             }
304             break;
305             case 1:
306             {
307                 cr = GetSsTemp(newBlock);
308             }
309             break;
310             default:
311                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
312                 processed = false;
313             }
314         }
315     }
316     break;
317 case 11:
318     {
319         switch(subcmdId)
320         {
321             case 0:
322             {
323                 cr = GetSsTm(newBlock);
324             }
325             break;
326             case 1:
327             {
328                 cr = GetSsTemp(newBlock);
329             }
330             break;
331             default:
332                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
333                 processed = false;
334             }
335         }
336     }
337     break;
338 case 12:
339     {
340         switch(subcmdId)
341         {
342             case 0:
343             {
344                 cr = GetSsTm(newBlock);
345             }
346             break;
347             case 1:
348             {
349                 cr = GetSsTemp(newBlock);
350             }
351             break;
352             default:
353                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
354                 processed = false;
355             }
356         }
357     }
358     break;
359 case 13:
360     {
361         switch(subcmdId)
362         {
363             case 0:
364             {
365                 cr = GetSsTm(newBlock);
366             }
367             break;
368             case 1:
369             {
370                 cr = GetSsTemp(newBlock);
371             }
372             break;
373             default:
374                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
375                 processed = false;
376             }
377         }
378     }
379     break;
380 case 14:
381     {
382         switch(subcmdId)
383         {
384             case 0:
385             {
386                 cr = GetSsTm(newBlock);
387             }
388             break;
389             case 1:
390             {
391                 cr = GetSsTemp(newBlock);
392             }
393             break;
394             default:
395                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
396                 processed = false;
397             }
398         }
399     }
400     break;
401 case 15:
402     {
403         switch(subcmdId)
404         {
405             case 0:
406             {
407                 cr = GetSsTm(newBlock);
408             }
409             break;
410             case 1:
411             {
412                 cr = GetSsTemp(newBlock);
413             }
414             break;
415             default:
416                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
417                 processed = false;
418             }
419         }
420     }
421     break;
422 case 16:
423     {
424         switch(subcmdId)
425         {
426             case 0:
427             {
428                 cr = GetSsTm(newBlock);
429             }
430             break;
431             case 1:
432             {
433                 cr = GetSsTemp(newBlock);
434             }
435             break;
436             default:
437                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
438                 processed = false;
439             }
440         }
441     }
442     break;
443 case 17:
444     {
445         switch(subcmdId)
446         {
447             case 0:
448             {
449                 cr = GetSsTm(newBlock);
450             }
451             break;
452             case 1:
453             {
454                 cr = GetSsTemp(newBlock);
455             }
456             break;
457             default:
458                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
459                 processed = false;
460             }
461         }
462     }
463     break;
464 case 18:
465     {
466         switch(subcmdId)
467         {
468             case 0:
469             {
470                 cr = GetSsTm(newBlock);
471             }
472             break;
473             case 1:
474             {
475                 cr = GetSsTemp(newBlock);
476             }
477             break;
478             default:
479                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
480                 processed = false;
481             }
482         }
483     }
484     break;
485 case 19:
486     {
487         switch(subcmdId)
488         {
489             case 0:
490             {
491                 cr = GetSsTm(newBlock);
492             }
493             break;
494             case 1:
495             {
496                 cr = GetSsTemp(newBlock);
497             }
498             break;
499             default:
500                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
501                 processed = false;
502             }
503         }
504     }
505     break;
506 case 20:
507     {
508         switch(subcmdId)
509         {
510             case 0:
511             {
512                 cr = GetSsTm(newBlock);
513             }
514             break;
515             case 1:
516             {
517                 cr = GetSsTemp(newBlock);
518             }
519             break;
520             default:
521                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
522                 processed = false;
523             }
524         }
525     }
526     break;
527 case 21:
528     {
529         switch(subcmdId)
530         {
531             case 0:
532             {
533                 cr = GetSsTm(newBlock);
534             }
535             break;
536             case 1:
537             {
538                 cr = GetSsTemp(newBlock);
539             }
540             break;
541             default:
542                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
543                 processed = false;
544             }
545         }
546     }
547     break;
548 case 22:
549     {
550         switch(subcmdId)
551         {
552             case 0:
553             {
554                 cr = GetSsTm(newBlock);
555             }
556             break;
557             case 1:
558             {
559                 cr = GetSsTemp(newBlock);
560             }
561             break;
562             default:
563                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
564                 processed = false;
565             }
566         }
567     }
568     break;
569 case 23:
570     {
571         switch(subcmdId)
572         {
573             case 0:
574             {
575                 cr = GetSsTm(newBlock);
576             }
577             break;
578             case 1:
579             {
580                 cr = GetSsTemp(newBlock);
581             }
582             break;
583             default:
584                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
585                 processed = false;
586             }
587         }
588     }
589     break;
590 case 24:
591     {
592         switch(subcmdId)
593         {
594             case 0:
595             {
596                 cr = GetSsTm(newBlock);
597             }
598             break;
599             case 1:
600             {
601                 cr = GetSsTemp(newBlock);
602             }
603             break;
604             default:
605                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
606                 processed = false;
607             }
608         }
609     }
610     break;
611 case 25:
612     {
613         switch(subcmdId)
614         {
615             case 0:
616             {
617                 cr = GetSsTm(newBlock);
618             }
619             break;
620             case 1:
621             {
622                 cr = GetSsTemp(newBlock);
623             }
624             break;
625             default:
626                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
627                 processed = false;
628             }
629         }
630     }
631     break;
632 case 26:
633     {
634         switch(subcmdId)
635         {
636             case 0:
637             {
638                 cr = GetSsTm(newBlock);
639             }
640             break;
641             case 1:
642             {
643                 cr = GetSsTemp(newBlock);
644             }
645             break;
646             default:
647                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
648                 processed = false;
649             }
650         }
651     }
652     break;
653 case 27:
654     {
655         switch(subcmdId)
656         {
657             case 0:
658             {
659                 cr = GetSsTm(newBlock);
660             }
661             break;
662             case 1:
663             {
664                 cr = GetSsTemp(newBlock);
665             }
666             break;
667             default:
668                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
669                 processed = false;
670             }
671         }
672     }
673     break;
674 case 28:
675     {
676         switch(subcmdId)
677         {
678             case 0:
679             {
680                 cr = GetSsTm(newBlock);
681             }
682             break;
683             case 1:
684             {
685                 cr = GetSsTemp(newBlock);
686             }
687             break;
688             default:
689                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
690                 processed = false;
691             }
692         }
693     }
694     break;
695 case 29:
696     {
697         switch(subcmdId)
698         {
699             case 0:
700             {
701                 cr = GetSsTm(newBlock);
702             }
703             break;
704             case 1:
705             {
706                 cr = GetSsTemp(newBlock);
707             }
708             break;
709             default:
710                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
711                 processed = false;
712             }
713         }
714     }
715     break;
716 case 30:
717     {
718         switch(subcmdId)
719         {
720             case 0:
721             {
722                 cr = GetSsTm(newBlock);
723             }
724             break;
725             case 1:
726             {
727                 cr = GetSsTemp(newBlock);
728             }
729             break;
730             default:
731                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
732                 processed = false;
733             }
734         }
735     }
736     break;
737 case 31:
738     {
739         switch(subcmdId)
740         {
741             case 0:
742             {
743                 cr = GetSsTm(newBlock);
744             }
745             break;
746             case 1:
747             {
748                 cr = GetSsTemp(newBlock);
749             }
750             break;
751             default:
752                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
753                 processed = false;
754             }
755         }
756     }
757     break;
758 case 32:
759     {
760         switch(subcmdId)
761         {
762             case 0:
763             {
764                 cr = GetSsTm(newBlock);
765             }
766             break;
767             case 1:
768             {
769                 cr = GetSsTemp(newBlock);
770             }
771             break;
772             default:
773                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
774                 processed = false;
775             }
776         }
777     }
778     break;
779 case 33:
780     {
781         switch(subcmdId)
782         {
783             case 0:
784             {
785                 cr = GetSsTm(newBlock);
786             }
787             break;
788             case 1:
789             {
790                 cr = GetSsTemp(newBlock);
791             }
792             break;
793             default:
794                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
795                 processed = false;
796             }
797         }
798     }
799     break;
800 case 34:
801     {
802         switch(subcmdId)
803         {
804             case 0:
805             {
806                 cr = GetSsTm(newBlock);
807             }
808             break;
809             case 1:
810             {
811                 cr = GetSsTemp(newBlock);
812             }
813             break;
814             default:
815                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
816                 processed = false;
817             }
818         }
819     }
820     break;
821 case 35:
822     {
823         switch(subcmdId)
824         {
825             case 0:
826             {
827                 cr = GetSsTm(newBlock);
828             }
829             break;
830             case 1:
831             {
832                 cr = GetSsTemp(newBlock);
833             }
834             break;
835             default:
836                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
837                 processed = false;
838             }
839         }
840     }
841     break;
842 case 36:
843     {
844         switch(subcmdId)
845         {
846             case 0:
847             {
848                 cr = GetSsTm(newBlock);
849             }
850             break;
851             case 1:
852             {
853                 cr = GetSsTemp(newBlock);
854             }
855             break;
856             default:
857                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
858                 processed = false;
859             }
860         }
861     }
862     break;
863 case 37:
864     {
865         switch(subcmdId)
866         {
867             case 0:
868             {
869                 cr = GetSsTm(newBlock);
870             }
871             break;
872             case 1:
873             {
874                 cr = GetSsTemp(newBlock);
875             }
876             break;
877             default:
878                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
879                 processed = false;
880             }
881         }
882     }
883     break;
884 case 38:
885     {
886         switch(subcmdId)
887         {
888             case 0:
889             {
890                 cr = GetSsTm(newBlock);
891             }
892             break;
893             case 1:
894             {
895                 cr = GetSsTemp(newBlock);
896             }
897             break;
898             default:
899                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
900                 processed = false;
901             }
902         }
903     }
904     break;
905 case 39:
906     {
907         switch(subcmdId)
908         {
909             case 0:
910             {
911                 cr = GetSsTm(newBlock);
912             }
913             break;
914             case 1:
915             {
916                 cr = GetSsTemp(newBlock);
917             }
918             break;
919             default:
920                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
921                 processed = false;
922             }
923         }
924     }
925     break;
926 case 40:
927     {
928         switch(subcmdId)
929         {
930             case 0:
931             {
932                 cr = GetSsTm(newBlock);
933             }
934             break;
935             case 1:
936             {
937                 cr = GetSsTemp(newBlock);
938             }
939             break;
940             default:
941                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
942                 processed = false;
943             }
944         }
945     }
946     break;
947 case 41:
948     {
949         switch(subcmdId)
950         {
951             case 0:
952             {
953                 cr = GetSsTm(newBlock);
954             }
955             break;
956             case 1:
957             {
958                 cr = GetSsTemp(newBlock);
959             }
960             break;
961             default:
962                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
963                 processed = false;
964             }
965         }
966     }
967     break;
968 case 42:
969     {
970         switch(subcmdId)
971         {
972             case 0:
973             {
974                 cr = GetSsTm(newBlock);
975             }
976             break;
977             case 1:
978             {
979                 cr = GetSsTemp(newBlock);
980             }
981             break;
982             default:
983                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
984                 processed = false;
985             }
986         }
987     }
988     break;
989 case 43:
990     {
991         switch(subcmdId)
992         {
993             case 0:
994             {
995                 cr = GetSsTm(newBlock);
996             }
997             break;
998             case 1:
999             {
1000                 cr = GetSsTemp(newBlock);
1001             }
1002             break;
1003             default:
1004                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1005                 processed = false;
1006             }
1007         }
1008     }
1009     break;
1010 case 44:
1011     {
1012         switch(subcmdId)
1013         {
1014             case 0:
1015             {
1016                 cr = GetSsTm(newBlock);
1017             }
1018             break;
1019             case 1:
1020             {
1021                 cr = GetSsTemp(newBlock);
1022             }
1023             break;
1024             default:
1025                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1026                 processed = false;
1027             }
1028         }
1029     }
1030     break;
1031 case 45:
1032     {
1033         switch(subcmdId)
1034         {
1035             case 0:
1036             {
1037                 cr = GetSsTm(newBlock);
1038             }
1039             break;
1040             case 1:
1041             {
1042                 cr = GetSsTemp(newBlock);
1043             }
1044             break;
1045             default:
1046                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1047                 processed = false;
1048             }
1049         }
1050     }
1051     break;
1052 case 46:
1053     {
1054         switch(subcmdId)
1055         {
1056             case 0:
1057             {
1058                 cr = GetSsTm(newBlock);
1059             }
1060             break;
1061             case 1:
1062             {
1063                 cr = GetSsTemp(newBlock);
1064             }
1065             break;
1066             default:
1067                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1068                 processed = false;
1069             }
1070         }
1071     }
1072     break;
1073 case 47:
1074     {
1075         switch(subcmdId)
1076         {
1077             case 0:
1078             {
1079                 cr = GetSsTm(newBlock);
1080             }
1081             break;
1082             case 1:
1083             {
1084                 cr = GetSsTemp(newBlock);
1085             }
1086             break;
1087             default:
1088                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1089                 processed = false;
1090             }
1091         }
1092     }
1093     break;
1094 case 48:
1095     {
1096         switch(subcmdId)
1097         {
1098             case 0:
1099             {
1100                 cr = GetSsTm(newBlock);
1101             }
1102             break;
1103             case 1:
1104             {
1105                 cr = GetSsTemp(newBlock);
1106             }
1107             break;
1108             default:
1109                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1110                 processed = false;
1111             }
1112         }
1113     }
1114     break;
1115 case 49:
1116     {
1117         switch(subcmdId)
1118         {
1119             case 0:
1120             {
1121                 cr = GetSsTm(newBlock);
1122             }
1123             break;
1124             case 1:
1125             {
1126                 cr = GetSsTemp(newBlock);
1127             }
1128             break;
1129             default:
1130                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1131                 processed = false;
1132             }
1133         }
1134     }
1135     break;
1136 case 50:
1137     {
1138         switch(subcmdId)
1139         {
1140             case 0:
1141             {
1142                 cr = GetSsTm(newBlock);
1143             }
1144             break;
1145             case 1:
1146             {
1147                 cr = GetSsTemp(newBlock);
1148             }
1149             break;
1150             default:
1151                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1152                 processed = false;
1153             }
1154         }
1155     }
1156     break;
1157 case 51:
1158     {
1159         switch(subcmdId)
1160         {
1161             case 0:
1162             {
1163                 cr = GetSsTm(newBlock);
1164             }
1165             break;
1166             case 1:
1167             {
1168                 cr = GetSsTemp(newBlock);
1169             }
1170             break;
1171             default:
1172                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1173                 processed = false;
1174             }
1175         }
1176     }
1177     break;
1178 case 52:
1179     {
1180         switch(subcmdId)
1181         {
1182             case 0:
1183             {
1184                 cr = GetSsTm(newBlock);
1185             }
1186             break;
1187             case 1:
1188             {
1189                 cr = GetSsTemp(newBlock);
1190             }
1191             break;
1192             default:
1193                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1194                 processed = false;
1195             }
1196         }
1197     }
1198     break;
1199 case 53:
1200     {
1201         switch(subcmdId)
1202         {
1203             case 0:
1204             {
1205                 cr = GetSsTm(newBlock);
1206             }
1207             break;
1208             case 1:
1209             {
1210                 cr = GetSsTemp(newBlock);
1211             }
1212             break;
1213             default:
1214                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1215                 processed = false;
1216             }
1217         }
1218     }
1219     break;
1220 case 54:
1221     {
1222         switch(subcmdId)
1223         {
1224             case 0:
1225             {
1226                 cr = GetSsTm(newBlock);
1227             }
1228             break;
1229             case 1:
1230             {
1231                 cr = GetSsTemp(newBlock);
1232             }
1233             break;
1234             default:
1235                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1236                 processed = false;
1237             }
1238         }
1239     }
1240     break;
1241 case 55:
1242     {
1243         switch(subcmdId)
1244         {
1245             case 0:
1246             {
1247                 cr = GetSsTm(newBlock);
1248             }
1249             break;
1250             case 1:
1251             {
1252                 cr = GetSsTemp(newBlock);
1253             }
1254             break;
1255             default:
1256                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1257                 processed = false;
1258             }
1259         }
1260     }
1261     break;
1262 case 56:
1263     {
1264         switch(subcmdId)
1265         {
1266             case 0:
1267             {
1268                 cr = GetSsTm(newBlock);
1269             }
1270             break;
1271             case 1:
1272             {
1273                 cr = GetSsTemp(newBlock);
1274             }
1275             break;
1276             default:
1277                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1278                 processed = false;
1279             }
1280         }
1281     }
1282     break;
1283 case 57:
1284     {
1285         switch(subcmdId)
1286         {
1287             case 0:
1288             {
1289                 cr = GetSsTm(newBlock);
1290             }
1291             break;
1292             case 1:
1293             {
1294                 cr = GetSsTemp(newBlock);
1295             }
1296             break;
1297             default:
1298                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1299                 processed = false;
1300             }
1301         }
1302     }
1303     break;
1304 case 58:
1305     {
1306         switch(subcmdId)
1307         {
1308             case 0:
1309             {
1310                 cr = GetSsTm(newBlock);
1311             }
1312             break;
1313             case 1:
1314             {
1315                 cr = GetSsTemp(newBlock);
1316             }
1317             break;
1318             default:
1319                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1320                 processed = false;
1321             }
1322         }
1323     }
1324     break;
1325 case 59:
1326     {
1327         switch(subcmdId)
1328         {
1329             case 0:
1330             {
1331                 cr = GetSsTm(newBlock);
1332             }
1333             break;
1334             case 1:
1335             {
1336                 cr = GetSsTemp(newBlock);
1337             }
1338             break;
1339             default:
1340                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1341                 processed = false;
1342             }
1343         }
1344     }
1345     break;
1346 case 60:
1347     {
1348         switch(subcmdId)
1349         {
1350             case 0:
1351             {
1352                 cr = GetSsTm(newBlock);
1353             }
1354             break;
1355             case 1:
1356             {
1357                 cr = GetSsTemp(newBlock);
1358             }
1359             break;
1360             default:
1361                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1362                 processed = false;
1363             }
1364         }
1365     }
1366     break;
1367 case 61:
1368     {
1369         switch(subcmdId)
1370         {
1371             case 0:
1372             {
1373                 cr = GetSsTm(newBlock);
1374             }
1375             break;
1376             case 1:
1377             {
1378                 cr = GetSsTemp(newBlock);
1379             }
1380             break;
1381             default:
1382                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1383                 processed = false;
1384             }
1385         }
1386     }
1387     break;
1388 case 62:
1389     {
1390         switch(subcmdId)
1391         {
1392             case 0:
1393             {
1394                 cr = GetSsTm(newBlock);
1395             }
1396             break;
1397             case 1:
1398             {
1399                 cr = GetSsTemp(newBlock);
1400             }
1401             break;
1402             default:
1403                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1404                 processed = false;
1405             }
1406         }
1407     }
1408     break;
1409 case 63:
1410     {
1411         switch(subcmdId)
1412         {
1413             case 0:
1414             {
1415                 cr = GetSsTm(newBlock);
1416             }
1417             break;
1418             case 1:
1419             {
1420                 cr = GetSsTemp(newBlock);
1421             }
1422             break;
1423             default:
1424                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1425                 processed = false;
1426             }
1427         }
1428     }
1429     break;
1430 case 64:
1431     {
1432         switch(subcmdId)
1433         {
1434             case 0:
1435             {
1436                 cr = GetSsTm(newBlock);
1437             }
1438             break;
1439             case 1:
1440             {
1441                 cr = GetSsTemp(newBlock);
1442             }
1443             break;
1444             default:
1445                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1446                 processed = false;
1447             }
1448         }
1449     }
1450     break;
1451 case 65:
1452     {
1453         switch(subcmdId)
1454         {
1455             case 0:
1456             {
1457                 cr = GetSsTm(newBlock);
1458             }
1459             break;
1460             case 1:
1461             {
1462                 cr = GetSsTemp(newBlock);
1463             }
1464             break;
1465             default:
1466                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1467                 processed = false;
1468             }
1469         }
1470     }
1471     break;
1472 case 66:
1473     {
1474         switch(subcmdId)
1475         {
1476             case 0:
1477             {
1478                 cr = GetSsTm(newBlock);
1479             }
1480             break;
1481             case 1:
1482             {
1483                 cr = GetSsTemp(newBlock);
1484             }
1485             break;
1486             default:
1487                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1488                 processed = false;
1489             }
1490         }
1491     }
1492     break;
1493 case 67:
1494     {
1495         switch(subcmdId)
1496         {
1497             case 0:
1498             {
1499                 cr = GetSsTm(newBlock);
1500             }
1501             break;
1502             case 1:
1503             {
1504                 cr = GetSsTemp(newBlock);
1505             }
1506             break;
1507             default:
1508                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
1509                 processed = false;
1510             }
1511         }
1512     }
1513     break;
1514 case 68:
1515     {
1516         switch(subcmdId)
```

```

224         default:
225             Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
226             processed = false;
227         }
228     }
229     break;
230     case 7:
231     {
232         switch(subcmdId)
233         {
234             case 0:
235             {
236                 cr = GetRwTm(newBlock);
237             }
238             break;
239             default:
240                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
241                 processed = false;
242             }
243         }
244     }
245     break;
246     case 8:
247     {
248         switch(subcmdId)
249         {
250             case 0:
251             {
252                 cr = SetMgtqPwm(newBlock);
253                 if(cr == CR_Success)
254                 {
255                     if(newBlock[2] == 0x55)
256                     {
257                         // Bypass Magnetometer response
258                         newBlock.resize(2);
259                         cr = GetMgtqTm(newBlock);
260                     }
261                     else
262                     {
263                         cr = BuildMgtmDataRequestCmd(newBlock);
264                         cr = GetMgtmTm(newBlock);
265                     }
266                 }
267             }
268             break;
269             default:
270                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
271                 processed = false;
272             }
273         }
274     }

```

```

274     case 9:
275     {
276         switch(subcmdId)
277         {
278             case 0:
279             {
280                 cr = GetIfScHk(newBlock);
281             }
282             break;
283             default:
284                 Log(Smp::Services::LMK_Warning, "Sub-command %u not implemented", subcmdId);
285                 processed = false;
286             }
287         }
288         break;
289         default:
290             Log(Smp::Services::LMK_Warning, "Command %u not implemented", cmdId);
291             processed = false;
292         }
293     }
294     switch(cr)
295     {
296     case CR_Success:
297     {
298         SendResponse(newBlock, false);
299     }
300     break;
301     case CR_InProgress:
302     {
303         Trace(5, "Operation in progress");
304     }
305     break;
306     case CR_Failure:
307     {
308         if(!processed)
309         {
310             newBlock.push_back(0x56);
311         }
312         SendResponse(newBlock, true);
313     }
314     break;
315     default:
316         Log(Smp::Services::LMK_Error, "Command result %u not supported", cr);
317     }
318     // MARKER: OPERATION BODY: END
319 }
320 // --CLOSING ELEMENT--AdcsIf::ObcRecvBlockCb--
321

```

2. Fault Model

In the case of ADCS_IF_SW we have defined a total of 18 fault models, two for each feature listed in Table 1. For each feature, one fault model captures the fault that might affect a nominal response message, one fault model captures the faults that might affect an error response message. In our experiments, however, we considered only the fault features known to be exercised by the SVF test suite, which are:

- IfStatus
- IfHK
- GYTM
- GYTMFailure
- SunSensorTM
- SunSensorTMFailure
- SSTP
- SSTPFailure
- SpaceCraftHK
- MagnetorquerSetPWMRSP

In the following sections we describe the fault models by providing for each byte of the response message (column *Byte*), the relevant bits (column *Bit*), a description of the information that is supposed to be transmitted by the byte (column *Description*), the type of data written on the byte (column *Type*), the fault classes that might affect the byte (column *Fault class*). We do not report the span of the item since it can be deduced from the table; indeed, descriptions that span over multiple rows correspond to data types that, to be loaded, require the reading of multiple data items. Concerning data types, the type DOUBLE is used for data items that internally to ESAIL are represented using the type `Smp::Float64`. On the channel, `Smp::Float64` is transmitted as `<PTC=3, PCF=6> Unsigned Integer 10bits`, which in the code is represented with `Smp::Int16`.

For each fault class, we indicate the value of the parameters required to configure the corresponding mutation operator (see Table 2.1 of D2). We use the keyword `@MIB` to indicate that the parameter value should be derived from the MIB database for ESAIL, more precisely from the file OBC.dat. In the database, the min and max range value for the nominal cases are reported. For example, Figure 2 shows a portion of the OBC.dat from which we can determine that MIN and the MAX values for AIFN031U are 3 and 3.6, respectively. The delta (i.e., parameter D) coincides with the lowest positive number that can be represented with the number of decimals appearing in the range (e.g., 0.1 for AIFN031U and 0.01 for AIFN031U). For some of the data items in the table we report also the corresponding identifier in OBC.dat. Missing identifiers will be reported in the coming months while refining the approach; indeed, decisions on the data items to be addressed by the approach may change after the first preliminary tests.

AIFN030U	1	H	24	33.53	AAA_OL80	1
AIFN031U	1	H	3	3.6	AAA_OL80	1
AIFN032U	1	H	3	3.6	AAA_OL80	1

Figure 2: Portion of OBC.dat

In column Fault class, the label NONE indicates that we are not interested into performing data-driven mutation testing for that specific byte. In general, we do not target with data-driven mutation those data items that do not concern features covered by the test suite. These are typically data items that do not cause a crash of the on board software or data items used only for self-testing of the board.

Columns Byte, Bit, and Description match the columns of corresponding tables in *ESAIL-LXS-ICD-P-0184*.

2.1 ADCS IF Status

Byte	Bit	Description	Type	Fault class
1	2..0	Reset Source Provides information about last reset. The bit is cleared after the first read of the status 0 = No reset 1 = Power-on Reset 2 = External Reset (released by JTAG adapter) 3 = Watchdog Reset 4 = Brown-out Reset 5 = JTAG AVR Reset (logic reset by JTAG) 6 = Not used 7 = Not used	BIN	BF(MIN=3;MAX=3) BF(MIN=4;MAX=4) BF(MIN=5;MAX=7)
	3	ADCS IF ready This bit is set when ADCS is ready to read/write to units. In the boot of the ADCS IF shall be a time to initialize all modules and units. After initialization of the ADCS IF, modules and units, shall go to a ready state. While ADCS IF is not ready, the available commands are: <ul style="list-style-type: none"> ASST ASHK ASCT 		
	4	OBC communication error This bit is set if a communication error between OBC and ADCS IF occurred in the last command. The bit is cleared after the first reading of the status 0 = No error 1 = Communication error		
	7..5	Unit communication error This bit is set if a communication error between ADCS IF and ADCS unit occurred. The bit is cleared after the first read of the status 0 = No error 1 = Communication error		
2	7..0	Unit in error Provides a list of units in error. 0 = No error 1 = Unit error Each bit is assigned to one unit: Bit 0 = Gyroscope unit Bit 1 = Reaction Wheel	BIN	BF(MIN=0;MAX=4)

		Bit 2 = Magnetorquer Bit 3 = Magnetometer Bit 4 = Sun Sensor		
3	7..0	Watchdog Reset Counter Watchdog Reset counter value. Increment in every watchdog reset. Value is stored in non-volatile memory To clear watchdog reset counter, shall be used the ASCF command.	INT	None: ESAIL OBC does not deal with anomalous values of reset counters. Thus we do not expect ESAIL test suite to fail in case of a high reset counter..
4	7..0	Overall Reset Counter Overall reset counter value. Increment in every device reset. Value is stored in non-volatile memory To clear overall reset counter, shall be used the ASCF command.	INT	None: same as above.
5	1..0	Gyroscope enable Enable/Disable status of nominal or redundant bus transceiver. 0 = Disabled both transceivers 1 = Enabled nominal transceiver only 2 = Enabled redundant transceiver only 3 = not existing (reserved for future needs)	BIN	BF(MIN=0;MAX=2) BF(MIN=2;MAX=4) BF(MIN=5;MAX=7)
	4..2	Reaction Wheel enable Enabled/Disabled status of bus transceiver. 0 = Disabled transceiver 1 = Enabled transceiver 7..2 = not existing (reserved for future needs)		
	7..5	3 axis Magnetorquer enable General Enable/Disable status of the Magnetorquer Driver for all three axis. 0 = Disabled 1 = Enabled Bit assignement: Bit 0 = Enabled/Disabled Driver Bit 1 = 0 not used (reserved for future needs) Bit 2 = 0 not used (reserved for future needs)		
6	1..0	Magnetometer enable Enable/Disable status of nominal or redundant bus transceiver. 0 = Disabled both transceivers 1 = Enabled nominal transceiver only 2 = Enabled redundant transceiver only 3 = not existing (reserved for future needs)	BIN	BF(MIN=0;MAX=1) BF(MIN=2;MAX=7)
	7..2	Sun Sensor board ADC enable Enabled/Disabled Sun Sensor board ADC, see Note 3) 0 = Disabled 1 = Enabled Each bit is assigned to one ADC: Bit 0 = Enabled/Disabled ADC2		

		Bit 1 = Enabled/Disabled ADC3 Bit 2 = Enabled/Disabled ADC4 Bit 3 = Enabled/Disabled ADC5 Bit 4 = Enabled/Disabled ADC6 Bit 5 = Enabled/Disabled ADC7		
--	--	---	--	--

2.2 ASHK - ADCS IF HK

Byte	Bit	Description	Type	Fault class
1	7..0	VCC1N		NONE
2	7..0	OBC Nominal transceiver circuit voltage		
3	7..0	VCC1R		NONE
4	7..0	OBC Redundant transceiver circuit voltage		NONE
5	7..0	VCC2		NONE
6	7..0	Gyroscope transceiver/UART circuit voltage		NONE
7	7..0	VCC3		NONE
8	7..0	Magnetometer transceiver/UART circuit voltage		NONE
9	7..0	VCC4		NONE
10	7..0	Reaction Wheel transceiver/UART circuit voltage		NONE
11	7..0	VCCa		NONE
12	7..0	Internal power supply (5.5V), measured with ADC0		NONE
13	7..0	VCCb		
14	7..0	Internal power supply (5.5V), measured with ADC1	DOUBLE	VAT(T=3.6;D=0.1) ID: AIFN031U
15	7..0	VBUS Unit input bus voltage	DOUBLE	VAT(T= 33.53;D=0.01) VBT(T=24;D=1) OBSW336U ID: AIFN030U (24-33.53) ASHK_VBUS - Unit In Bus Volt
16	7..0			
17	7..0	VCC5		NONE
18	7..0	Supply voltage for ADC2, ADC3, ADC4 and VCCB1. Sun-sensor PCB		NONE
19	7..0	VCC6		NONE
20	7..0	Supply voltage for ADC5, ADC6, ADC7 and VCCB2. Sun-sensor PCB		NONE
21	7..0	VCC5_IN		NONE
22	7..0	LDO input voltage for ADC2, ADC3, ADC4 and VCCB1. Sun-sensor PCB		NONE
23	7..0	VCC6_IN		NONE
24	7..0	LDO input voltage for ADC5, ADC6, ADC7 and VCCB2. Sun-sensor PCB		NONE
25	7..0	VCC_SW1 SSB internal switched power supply, measured by ADC3		VAT(T=6;D=1) ID: AIFN035U (5-6) ASHK_VCC_SW1 - SSB Sup Volt 1

26	7..0	Remark: the voltage VCC_SW is measured 2 times with two different ADC. This allows to compare the results and conclude for a drift in the ADC's.		
27	7..0	VCC_SW2 SSB internal switched power supply, measured by ADC6		NONE
28	7..0	Remark: the voltage VCC_SW is measured 2 times with two different ADC. This allows to compare the results and conclude for a drift in the ADC's.		NONE
29	7..0	T_PCB_TEMP1 Main Board PCB Temperature, sensor 1 Temperature of VCC DC/DC regulator. Remark: 1/2 is measured on the same place, it's to compare the values to discover a measurement failure	DOUBLE	VOR(MIN=-20; MAX=50;D=1) AIFR037T?
30	7..0			ID: AIFN037T (-20-50) ASHK_TMP1 - Main Brd Temp 1
31	7..0	T_PCB_TEMP2 Main Board PCB Temperature, sensor 2 Temperature of VCC DC/DC regulator. Remark: 1/2 is measured on the same place, it's to compare the values to discover a measurement failure	DOUBLE	VOR(MIN=-20; MAX=50;D=1) AIFR038T?
32	7..0			ID: AIFN038T (-20-50) ASHK_TMP2 - Main Brd Temp 2
33	7..0	T_PCB_TEMP3a Sun Sensor Board PCB Temperature, sensor 3a. Temperature of VCC5 LDO regulator. Remark: 3a/b is measured on the same place, it's to compare the values to discover a measurement failure	DOUBLE	VOR(MIN=-20; MAX=50;D=1) AIFR039T
34	7..0			ID: AIFN039T (-20-50) ASHK_TMP3a - SSB PCB Temp 3a
35	7..0	T_PCB_TEMP3b Sun Sensor Board PCB Temperature, sensor 3b. Temperature of VCC5 LDO regulator. Remark: 3a/b is measured on the same place, it's to compare the values to discover a measurement failure	DOUBLE	VOR(MIN=-20; MAX=50;D=1) AIFR040T
36	7..0			ID: AIFN040T (-20-50) ASHK_TMP3b - SSB PCB Temp 3b
37	7..0	T_PCB_TEMP4 Sun Sensor Board PCB Temperature, sensor 4. Temperature of VCC6 LDO regulator.	DOUBLE	VOR(MIN=-20; MAX=50;D=1) AIFR041T ID: AIFN041T (-20-50) ASHK_TMP4 - SSB PCB Temp 4

2.3 GYTM - Gyroscope TM

Byte	Bit	Description	Type	Fault class
------	-----	-------------	------	-------------

1	7..0	Unit identifier Identification of the unit that addresses the message 0 = Nominal 1 = Redundant	INT	BF(MIN=0,MAX=0)
21..2	7..0	Gyroscope Telemetry All telemetry data from Gyroscope. Message is the same sent from Gyroscope unit without adding/removing data	HEX	NONE: the type of data transmitted appear to be too much complicated to be mutated in such a way of triggering a test failure. Could be targeted in the future.

Byte	Bit	Description		
1	7..0	Error type .	HEX	IV(VALUE=0x51) IV(VALUE=0x52) IV(VALUE=0x53) IV(VALUE=0x54) IV(VALUE=0x56)

2.5 Sun Sensor TM

Byte	Bit	Description	Type	Fault class
1	7..0	Photodiode Q1 current ADC #3	DOUBLE	VAT(T=2.6;D=0.1)
2	7..0			ID: AIFN044I (0-2.6) SSTM_PXQ1 - pX Q1 Curr
3	7..0	Photodiode Q2 current ADC #3	DOUBLE	VAT(T=2.6;D=0.1)
4	7..0			ID: AIFN045I
5	7..0	Photodiode Q3 current ADC #3	DOUBLE	VAT(T=2.6;D=0.1)
6	7..0			ID: AIFN046I
7	7..0	Photodiode Q4 current ADC #3	DOUBLE	VAT(T=2.6;D=0.1)
8	7..0			ID: AIFN047I
9	7..0	Photodiode Q1 current ADC #2	DOUBLE	VAT(T=2.6;D=0.1)
10	7..0			ID: AIFN048I
11	7..0	Photodiode Q2 current ADC #2	DOUBLE	VAT(T=2.6;D=0.1)
12	7..0			ID: AIFN049I
13	7..0	Photodiode Q3 current ADC #2	DOUBLE	VAT(T=2.6;D=0.1)
14	7..0			ID: AIFN050I
15	7..0	Photodiode Q4 current ADC #2	DOUBLE	VAT(T=2.6;D=0.1)
16	7..0			ID: AIFN051I
17	7..0	Photodiode Q1 current ADC #6	DOUBLE	VAT(T=2.6;D=0.1)
18	7..0			ID: AIFN052I
19	7..0	Photodiode Q2 current ADC #6	DOUBLE	VAT(T=2.6;D=0.1)
20	7..0			ID: AIFN053I
21	7..0	Photodiode Q3 current ADC #6	DOUBLE	VAT(T=2.6;D=0.1) ID: AIFN054I

22	7..0			
23	7..0	Photodiode Q4 current ADC #6	DOUBLE	VAT(T=2.6;D=0.1) ID: AIFN055I
24	7..0			
25	7..0	Photodiode Q1 current ADC #5	DOUBLE	VAT(T=2.6;D=0.1) ID: AIFN056I
26	7..0			
27	7..0	Photodiode Q2 current ADC #5	DOUBLE	VAT(T=2.6;D=0.1) ID: AIFN057I
28	7..0			
29	7..0	Photodiode Q3 current ADC #5	DOUBLE	VAT(T=2.6;D=0.1) ID: AIFN058I
30	7..0			
31	7..0	Photodiode Q4 current ADC #5	DOUBLE	VAT(T=2.6;D=0.1) ID: AIFN059I
32	7..0			
33	7..0	Photodiode Q1 current ADC #4	DOUBLE	VAT(T=2.6;D=0.1) ID: AIFN060I
34	7..0			
35	7..0	Photodiode Q2 current ADC #4	DOUBLE	VAT(T=2.6;D=0.1) ID: AIFN061I
36	7..0			
37	7..0	Photodiode Q3 current ADC #4	DOUBLE	VAT(T=2.6;D=0.1) ID: AIFN062I
38	7..0			
39	7..0	Photodiode Q4 current ADC #4	DOUBLE	VAT(T=2.6;D=0.1) ID: AIFN063I
40	7..0			
41	7..0	Photodiode Q1 current ADC #7	DOUBLE	VAT(T=2.6;D=0.1) ID: AIFN064I
42	7..0			
43	7..0	Photodiode Q2 current ADC #7	DOUBLE	VAT(T=2.6;D=0.1) ID: AIFN065I
44	7..0			
45	7..0		DOUBLE	VAT(T=2.6;D=0.1)

		Photodiode Q3 current ADC #7		ID: AIFN066I
46	7..0			
47	7..0	Photodiode Q4 current ADC #7	DOUBLE	VAT(T=2.6;D=0.1) ID: AIFN067I
48	7..0			

Byte	Bit	Description	Type	Fault class
1	7..0	Error type .	HEX	IV(VALUE=0x51) IV(VALUE=0x54) IV(VALUE=0x56)

2.6 SSTP - Sun Sensor Temperature

Byte	Bit	Description	Type	Fault class
1	7..0	Temperature reading from ADC #3	DOUBLE	VOR(MIN=-70;MAX=100;D=1) ID: AIFN068T (-70-100) SSTP_TPXP - Temperature Xp
2	7..0			
3	7..0	Temperature reading from ADC #2	DOUBLE	VOR(MIN=-70;MAX=100;D=1) ID: AIFN069T
4	7..0			
5	7..0	Temperature reading from ADC #6	DOUBLE	VOR(MIN=-70;MAX=100;D=1) ID: AIFN070T
6	7..0			
7	7..0	Temperature reading from ADC #5	DOUBLE	VOR(MIN=-70;MAX=100;D=1) ID: AIFN071T
8	7..0			
9	7..0	Temperature reading from ADC #4	DOUBLE	VOR(MIN=-70;MAX=100;D=1) ID: AIFN072T
10	7..0			
11	7..0	Temperature reading from ADC #7	DOUBLE	VOR(MIN=-70;MAX=100;D=1) ID: AIFN073T
12	7..0			

Byte	Bit	Description	Type	Fault class
1	7..0	Error type .	HEX	IV(VALUE=0x51) IV(VALUE=0x54) IV(VALUE=0x56)

2.8 SpaceCraft HK

Byte	Bit	Description	Type	Fault class
1	7..0	TMTC_SW1 Identifies the switching position of the TMTC switch 1: the voltage is ~1.1V for position A and 2.2V for position B. 0V or 3.3V will indicate a short or an interruption.	DOUBLE	VAT(T=3.3;D=0.1) VBT(T=0;D=0.1) ID: AIFN086X
2	7..0			
3	7..0	TMTC_SW2 Identifies the switching position of the TMTC switch 2: the voltage is ~1.1V for position A and 2.2V for position B. 0V or 3.3V will indicate a short or an interruption.	DOUBLE	VOR(MIN=0.5;MAX=2.75;D=0.01) ID: AIFN087X
4	7..0			
5	7..0	SC_TEMP1 Temperature SC-TEMP1 of a sensor in the S/C structure	DOUBLE	VOR(MIN=0;MAX=50;D=1) ID: AIFN088T (0-50) SCHK_SCT1 - OBC Thermistor 1
6	7..0			
7	7..0	SC_TEMP2 Temperature SC-TEMP2 of a sensor in the S/C structure	DOUBLE	VOR(MIN=0;MAX=50;D=1) ID: AIFN089T
8	7..0			
9	7..0	SC_TEMP3 Temperature SC-TEMP3 of a sensor in the S/C structure	DOUBLE	VOR(MIN=0;MAX=50;D=1) ID: AIFN090T
10	7..0			
11	7..0	SC_TEMP4 Temperature SC-TEMP4 of a sensor in the S/C structure	DOUBLE	VOR(MIN=0;MAX=50;D=1) ID: AIFN091T
12	7..0			
13	7..0	SC_TEMP5 Temperature SC-TEMP5 of a sensor in the S/C structure	DOUBLE	VOR(MIN=9.9253;MAX=29.9979;D=0.0001) ID: AIFN092T
14	7..0			

15	7..0	SC_TEMP6 Temperature SC-TEMP6 of a sensor in the S/C structure	DOUBLE	VOR(MIN =9.9253; MAX= 29.9979; D=0.0001)
16	7..0			ID: AIFN093T

Byte	Bit	Description		
1	7..0	Error type .	HEX	IV(VALUE=0x56)

2.9 Magnetorquer Set PWM RSP

Byte	Bit	Description	Type	Fault class
1	7..0	Unit identifier Magnetometer Identification of the Magnetometer unit that addresses the message 0 = Nominal 1 = Redundant	BIN	BF(MIN=0;MAX=0)
2	7..0	Magnetometer Data request reply Byte1 Sync(LSB) (Note 1)		NONE: Not to address with the approach because the effect of a mutation is not predictable (the trasferred data is complex, e.g., signal).
3	7..0	Magnetometer Data request reply Byte2 Sync(MSB) (Note 1)		NONE: same as above.
4	7..0	Magnetometer Data request reply Byte3 RAdr (Note 1)		NONE: same as above.
5	7..0	Magnetometer Data request reply Byte4 Sadr (Note 1)		NONE: same as above.
6	7..0	Magnetometer Data request reply Byte5 ReplyMsg (Note 1)		NONE: same as above.
7	7..0	Magnetometer Data request reply Byte6 Bx Low (Note 1)		NONE: same as above.
8	7..0	Magnetometer Data request reply Byte7 Bx Middle		NONE: same as above.
9	7..0	Magnetometer Data request reply Byte8 CS error + Average + pos Clip X + neg Clip X + BX High (Note 1)		NONE: same as above.
10	7..0	Magnetometer Data request reply Byte9 By Low (Note 1)		NONE: same as above.
11	7..0	Magnetometer Data request reply Byte10 By Middle (Note 1)		NONE: same as above.
12	7..0	Magnetometer Data request reply Byte11 spare + pos Clip Y + neg Clip Y + BY High (Note 1)		NONE: same as above.
13	7..0	Magnetometer Data request reply Byte12 Bz Low (Note 1)		NONE: same as above.
14	7..0	Magnetometer Data request reply Byte13 Bz Middle (Note 1)		NONE: same as above.
15	7..0	Magnetometer Data request reply Byte14 spare + pos Clip Z + neg Clip Z + BZ High (Note 1)		NONE: same as above.
16	7..0	Magnetometer Data request reply Byte15 CS (Note 1)		NONE: same as above.
17	7..0	Magnetorquer nX Current - on Current MTXA_N when powered	DOUBLE	VOR(MIN=0.14,MAX=0.21,D=0.01)

18	7..0			MT nX Curr - on
19	7..0	Magnetorquer nX Current - off Current MTXA_N when unpowered	DOUBLE	VOR(MIN=0,MAX=0.2,D=0.1)
20	7..0			AIFR075I MT nX Curr - off
21	7..0	Magnetorquer pX Current - on Current MTXA_P when powered	DOUBLE	VOR(MIN=0.14,MAX=0.21,D=0.01)
22	7..0			AIFR076I MT pX Curr - on
23	7..0	Magnetorquer pX Current - off Current MTXA_P when unpowered	DOUBLE	VOR(MIN=0,MAX=0.2,D=0.1)
24	7..0			AIFR077I MT pX Curr - off
25	7..0	Magnetorquer nY Current - on Current MTYA_N when powered	DOUBLE	VOR(MIN=0.14,MAX=0.21,D=0.01)
26	7..0			AIFR078I MT nY Curr - on
27	7..0	Magnetorquer nY Current - off Current MTYA_N when unpowered	DOUBLE	VOR(MIN=0,MAX=0.2,D=0.1)
28	7..0			AIFR079I MT nY Curr - off
29	7..0	Magnetorquer pY Current - on Current MTYA_P when powered	DOUBLE	VOR(MIN=0.14,MAX=0.21,D=0.01)
30	7..0			AIFR080I MT pY Curr - on
31	7..0	Magnetorquer pY Current - off Current MTYA_P when unpowered	DOUBLE	VOR(MIN=0,MAX=0.2,D=0.1)
32	7..0			AIFR081I MT pY Curr - off
33	7..0	Magnetorquer nZ Current - on Current MTZA_N when powered	DOUBLE	VOR(MIN=0.14,MAX=0.21,D=0.01)
34	7..0			AIFR082I MT nZ Curr - on
35	7..0	Magnetorquer nZ Current - off Current MTZA_N when unpowered	DOUBLE	VOR(MIN=0,MAX=0.2,D=0.1)
36	7..0			AIFR083I MT nZ Curr - off
37	7..0	Magnetorquer pZ Current - on Current MTZA_P when powered	DOUBLE	VOR(MIN=0.14,MAX=0.21,D=0.01)
38	7..0			AIFR084I

				MT pZ Curr - on
39	7..0	Magnetorquer pZ Current - off Current MTZA_P when unpowered	DOUBLE	VOR(MIN=0,MAX= 0.2,D=0.1) AIFR085I MT pZ Curr - off

3. Mutation Probes

Mutation probes are manually integrated into the source code of function *ObcRecvBlockCb*. Figure 3 shows an example of how we integrate mutation probes. All the probes are integrated following the same pattern; more precisely, for each data generation function we manually insert two invocations to the FAQAS mutation probe API, one to perform mutation of the nominal response message (Line 154, in Figure 3) the other one to mutate an error response message (Line 149). The choice of the data model to pass to the FAQAS mutation probe API is based on the value of *cr*, the variable that captures the return status of the specific data generation function invoked (function *GetIfHk* in Figure 3).

The function `_FAQAS_mutate` takes as input the fault model to be used to drive the mutation. Fault models are automatically generated from template files matching to the tables reported in Section 2.

```
144:     case 0:
145:     {
146:         cr = GetIfStatus(newBlock);
147:         if ( cr == CR_Failure ){
148:             FaultModel *dm = _FAQAS_GetIfStatus_FM_Error ()
149:             _FAQAS_mutate( newBlock, dm );
150:             _FAQAS_delete_DM( dm )
151:         } else {
152:             FaultModel *dm = _FAQAS_GetIfStatus_FM ()
153:             _FAQAS_mutate( newBlock, dm);
154:             _FAQAS_delete_DM( dm )
155:         }
156:     }
```

Figure 3: Mutation probe for *GetIfStatus*