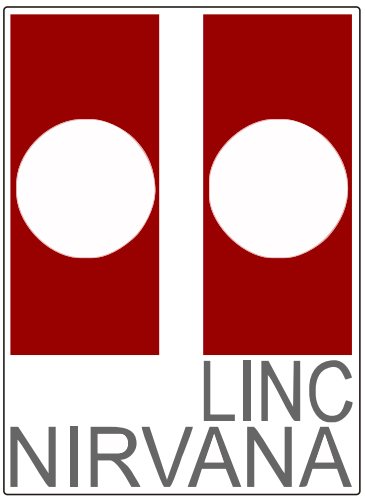
LINC-NIRVANA

The **L**BT **IN**terferometric **C**amera and
Near-**I**nfra**R**ed / **V**isible **A**daptive
i**N**terferometer for **A**stronomy

A collaborative project of the MPIA Heidelberg, INAF-Arcetri,
Universität zu Köln, and MPIfR Bonn
http://www.mpia.de/LINC



LINC-NIRVANA

—

Generic Infrared Software – ROE Pattern Constructor

Doc. No.	LN-MPIA-MAN-ICS-008
Short Title	GEIRS Pattern Constructor
Issue	6.097
Date	April 7, 2022

	Name	Date	Signature
Prepared	Richard J. Mathar	April 7, 2022	.....
Approved	Peter Bizenberger		.....
Released	Martin Kürster		.....

Change Record

Issue	Date	Sect.	Reason/Initiation/Documents/Remarks
0.041	2013-02-11	all	created
0.119	2013-04-29		Version submitted to the ADP
6.097	April 7, 2022	all	GEIRS SVN version trunk-r799M-65

Contents

List of Figures

1 OVERVIEW

1.1 Design

The Generic Infrared Software (GEIRS) is a software layer written in C++, ANSI-C and Java, which

- assembles parameter lists and commands received from its own graphical interface or other supervisor software,
- translates these into the firmware language ('patterns') of the readout electronics (ROE)
- initializes the readout cycles
- and accumulates the frames received from the ADC's of the electronics as FITS files on the local disks or X11 images displayed in the engineering GUI.

This document summarizes the functionality within GEIRS that scans a textual description of readout patterns and emits the codes (lines) that are converted by the firmware to the sequence of clocks. We coin the phrase *pattern constructor* to avoid a name clash with *pattern generator*, which is the bottom level (the pattern RAM program) of the firmware actually running the detector pins.

A design choice of the pattern construction is to split functionality into the scanner and the textual description, assuming that the scanner is a kind of virtual representation of the FPGA's capabilities and updated infrequently and co-jointly with that part of the firmware, whereas the patterns are independently editable by the electronic engineers investigating the characteristics and performance of the individual chips.

That lead to a simple type of programming (interpreter) language for patterns, the grammar of which we describe in this manual. (The noun *patterns* is perhaps misleading here, as the framework is the current interface to Version 3.1 of MPIA's readout electronics for Hawaii-2, Hawaii-2RG and Hawaii-4RG infrared detectors with two FPGA's, not any generic driver for a state machinery of arbitrary detector chips.)

1.2 Interfaces

The document complements the documents on the camera control software [1], ROE [2], readout patterns [3], and RoCon [4]. To obtain a full list of the commands understood by the FPGA, please contact the representative electronic engineer of the MPIA.

1.3 Acronyms

ADC analog-to-digit conversion

ADU analog-to-digital unit

AIP Leibniz-Institut für Astrophysik Potsdam https://www.aip.de

ANSI American National Standards Institute http://www.ansi.org

ASCII American Standard Code for Information Interchange https://en.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange

CARMENES Calar Alto High-Resolution Search for M Dwarfs with Exoearths with Near-infrared and Optical Echelle Spectrographs carmenes.caha.es

DMA Direct Memory Access

FITS Flexible Image Transport System http://fits.gsfc.nasa.gov

FPGA Field programmable gate array

GEIRS Generic Infrared Software

GUI Graphical User Interface

IP Internet Protocol

LINC LBT Interferometric Camera http://www.mpia-hd.mpg.de/LINC/

LINC-NIRVANA LBT Interferometric Camera and Near-Infrared / Visible Adaptive Interferometer for Astronomy

LN liquid nitrogen

LN LINC-NIRVANA

LUCI LBT NIR spectroscopic Utility with Camera and Integral-Field Unit for Extragalactic Research http://www.mpe.mpg.de/ir/lucifer

MPIA Max-Planck Institut für Astronomie, Heidelberg https://www.mpia.de

MPIfR Max-Planck Institut für Radioastronomie, Bonn http://www.mpifr-bonn.mpg.de

NIRVANA Near-Infrared / Visible Adaptive Interferometer for Astronomy

NTE NOT Transit Explorer https://nte.nbi.ku.dk/

PANIC Panoramic Near-Infrared Camera https://panic.iaa.es

PCIe Peripheral Component Interconnect Express https://en.wikipedia.org/wiki/PCI_Express

PLX PLX Technology, http://www.broadcom.com/products/pcie-switches-bridges/software-dev-kit

RAM Random Access Memory

RoCon Readout Controller

ROE Readout Electronics

SVN Subversion http://subversion.apache.org

1.4 References

References

[1] C. Storz, LINC-NIRVANA - Infrared Camera Control Software, lN-MPIA-FDR-ICS-005 (6 Jun. 2005).

[2] B. Grimm, U. Mall, LINC-NIRVANA - Readout Electronics for the Science Detector, LN-MPIA-FDR-ELEC-001 (21 Jan. 2005).

[3] V. Naranjo, LINC-NIRVANA - IR Detector Control Pattern, LN-MPIA-DES-ELEC-007 (5 Apr. 2008).

[4] J. R. Ramos, ROCON REad-out Controller Board (Nov. 2009).
URL webdavs://sk1/geirs/roe3MPIA/Roconv3-Draft.pdf

[5] W. Gaessler, LINC-NIRVANA - Software Style Guide, LN-MPIA-MAN-ICS-001 (02 Aug. 2004).

[6] U. Mall, C. Storz, CARMENES - NIR channel – Readout electronics and software, FDR-04C2A. E: in section 2.6.2 the factor 0.5 of the voltage divider is wrong. The actual value for the CARMENES racks is 0.699. (30 Jan. 2013).

[7] C. Storz, V. Naranjo, U. Mall, J. R. Ramos, P. Bizenberger, J. Panduro, Standard modes of MPIA's current H2/H2RG-readout systems, in: 2012 Astronomial Telescopes and Instrumentation, Vol. 8453 of Proc. SPIE, Int. Soc. Optical Engineering, 2012, p. 2E. doi:10.1117/12.927170.

[8] R. J. Mathar, LINC-NIRVANA - Generic Infrared Software, Graphical User Manual, LN-MPIA-MAN-ICS-007 (2 Oct. 2018).
URL https://www.mpia.de/~mathar/public/LN-MPIA-MAN-ICS-007.pdf

[9] U. Mall, LINC-NIRVANA - ROE Schematics, lN-MPIA-TN-ELEC-013 (30 Jan. 2013).
URL https://svn.mpia.de/trac/gulli/ln/archive/Archive/LN%20Documentation/Technical%20Notes%20(TN)/Electronics%2C%20including%20detectors%20(ELEC)/TN-ELEC-013-ROESchematics/LN-MPIA-TN-ELEC-013.pdf

[10] U. Mall, IR ReadOut Electronics Technical Manual, 1st Edition (Oct. 2014).

[11] Teledyne, Hawaii-2RG Technical Manual (25 Sep. 2007).

[12] J. R. Ramos, ROCON v3 Upgrade von ROCON v3.0 auf ROCON v3.1 (6 Nov. 2014).

2 FILE NAMES

The main body of the GEIRS, including the text file parser and communicator with the RoCon, is in the SVN repository https://svn.mpia.de/gulli/geirs/src. The patterns are in the subdirectory `pttrns`, one per instrument, for example https://svn.mpia.de/gulli/geirs/src/trunk/pttrns/Luci for the two LUCI's. In consequence,

- adding new features to the parser or any kind of extensions to the syntax described below, and occasionally also modifications of constants requires changes in the C-source code of the current branch plus recompilation;
- adding new patterns, modifications on the shuffling, insertion/changes of patterns and their orders and so on can usually be done on the spot by editing the textual representation in the `pttrns` subdirectory of the layout and restarting GEIRS. These files are interpreted, not compiled.

Pattern file names usually do not have suffices, so they are for example `init_pat_H2RG` and not `init_pat_H2RG.luci1`. The benefit is

- The output of the directory for example with `ls` is much easier to read

- Cross-linking of patterns that are shared by instruments of the same detector type is easy. One does note need separate files like `init_pat_H2RG.luci1` and `init_pat_H2RG.luci2` but just one file `init_pat_H2RG`, for example.

To mechanism in the GEIRS parser to look for a file is: first search for a file *with* the instrument suffix, and if that file does not exist search for the file without the instrument suffix. That means for feature tests while testing patterns, the developper can edit/modify a pattern file that *has* the suffix and which will be read with priority by the pattern constructor, while the pattern variant of some stable (or common) version *without* the suffix remains in the same directory.

Permanent changes anyway require uploading the modifications to SVN. Beware of symbolic links, which share patterns between instruments.

If the name of an instrument is changed—for example from `Lucifer` to `Luci1` and from `Luci` to `Luci2`—, one would

1. add an alias to the enumeration, the new name and the new extension in `camtypes.h`,
2. create a new subdirectory in `pttrns` reflecting the new name, copy all files of the old directory to the new one, and switch suffixes in the new `pttrns/`*instru* with a script similar to

```
#!/bin/bash

# Move all files with suffix .lucifer to files with suffix .luci1
for fil in *.lucifer ; do
        targ=${fil%".lucifer"}.luci1 ;
        mv $fil $targ
done
```

3. introduce the new `CAMERA` variable in the `scripts/GENERIC` and optionally add a new link.

Some overview of which exposure times have been calculated is kept in `$CAMHOME/log/itime*`. Detailed accounts of how these intervals accumulate appear in `$CAMTMP/timing*.log`. Software style and file layout are completely unaware of any LN-specific documented guidelines [5].

2.1 Start of Computation

The entry point in the file system where the construction of the pattern starts is `roe_init_ch`*ch.extension*, where

- *ch* denotes the one or two digits of the number of ADC channels in use. For multi-chip cameras like AIP or CARMENES, this is the number of channels dedicated to each individual chip,
- and where *.extension* is optional, `.nirvana` in our case.

The scanner steers the selection of parameters that build tables in the FPGA program by computing file names from variables that are known to the main parser. The selection of read-out modes and window modes, for example, triggers inclusion of a subset of files in the `pttrns` directory with the mechanism detailed later and illustrated in Figure 1. Conceptionally, all supported readout modes are implied by the files in one SVN revision.

The effect of scanning the pattern files is immediate. In particular each send command (Section 3.2.5) visited in a file interacts with the real hardware of the ROE; if GEIRS had been started with the ROE declared in an offline state, a form of software simulation within GEIRS takes over. This software simulator is essentially the only method for a dry-run syntax check of the pattern and timing files; it will detect problems with ill-defined variables, missing parentheses and other typographic errors, but it also is absolutely optimistic assuming that all commands to the ROE that are constructed would be returning no errors from the firmware.

The scanner interprets the pattern files line-by-line. So

- the contents of files that are not included (by the mechanism of chapter 3.2.6) does not matter.
- errors in commands sent to the ROE may leave the (partially initialized) ROE in some fuzzy state.

For that reason, the communication with the firmware may use a relaxed set of timeouts (see Section 3.2.5). With that concept of early setup, the `read` command will start an exposure with minimum time overhead, because this sends a kind of break-interrupt code with a pointer to the new pattern program to the (micro-processor of the) ROE, only a few tens of bytes via the internet. The delay of this exposure still depends on which type of idle-(break)-mode is currently executed by the firmware.

2.2 Conventions

The files named `biases` with commands in the 900 range contain voltage biases and register settings for the infrared chip [4, 6].

Files starting `roe*` are top level commands issued by GEIRS at certain strategic times (power-up, shutdown, re-init,...).

Files starting `lay*` and `pat*` emit lines for the up to five intermediate layers of the FPGA program and the lowest (RAM) layer. The name `pat` indicates these are the definitions of the actual patterns beyond the administrative other layers.

The `table*` files are tables that cross through one or more layers.

`incl*` are sequences in lower levels meant not to be started as a self-sustained program but to be included by other files.

The `timing*` files contain basically sets of embedded timing commands, see Section 4.

There is no formal difference in the handling of these files. Any file can incorporate other files with the `include` mechanism (Section 3.2.6); the variable syntax is the same. Lines within the `timing*` files that use the timing calculator still need to start with the special comment `#!timing=` to trigger some calculation. In that respect, it is a mere convention and is not syntactically required that the construction of the patterns and the timing calculation are kept in separate files.

2.3 Cycle Type Conventions

After an underscore, the standard acronyms of readout modes are often a substring of the file names [7]. The acronyms and the representation in the FITS headers are gathered in Table 1. This list in Table 1 is basically fixed if the file name is computed by expanding the `ctype` variable (see Section

Table 1: Supported GEIRS readout modes. The checked combinations are admitted by the software.

short	FITS	LN,AIP,PANIC,NTE	CARMENES,Luci
`rrr`	reset.read.read	✓	✓
`rr`	reset.read	✓	✓
`scr`	single.corr.read	✓	✓
`dcr`	double.corr.read	✓	✓
`mcr`	multi.corr.read	✓	✓
`mer`	multiple.endpoints	✓	✓
`rr-mpia`	fast-reset.read	✓	✓
`rrr-mpia`	fast-reset-read.read	✓	✓
`srr`	sample.ramp.read	✓	✓
`srre`	sample.ramp.read.embreset		✓
`fcr`	fast-rst-double.corr.read	✓	✓
`lir`	line.interlaced.read	✓	✓
`o2scr`	o2.single.corr.read	✓	✓
`o2dcr`	o2.double.corr.read	✓	✓
`msr`	multiple.successive.read	✓	✓
`spr`	single.pixel.read	✓	✓
`rlr`	reset.level.read	✓	✓
`sfr`	single.frame.read	✓	✓
`fecr`	fast.end-corr-dcr.read	✓	✓
`limer`	line.interlaced.multiple.endpoint.read	✓	✓
`lisrr`	line.interlaced.sample.ramp.read	✓	✓
`limsr`	line.interlaced.multiple.successive.read	✓	✓
`licntsr`	line.interl.continuous.sampling.read	✓	✓
`cntsr`	continuous.sampling.read	✓	✓

3.2). The cycle types listed above are a hard-coded list in the current source code, `cameratypes.h`, and depend on which version of GEIRS is used. The suffix of the file names is instrument specific, here `.nirvana`, and otherwise `.panic`, `.luci1`, `.luci2`, `.carmenes`, `.nteimg`, `.ntespec`, `.sc`. Lists with aspects like these contain many entries which may be obsolete for a decade or more and may certainly change in the future.

The read modi (cycle types) offered by the software for a specific instrument (camera) are hard-coded in the function `supported_ctype` in `nutil.c` as in Table 1.

Based on criteria which are not related to software, only the `lir` with electronic multi-sampling set to 4 and `rrr` are considered the readout modes for standard LN scientific operations.

2.4 Logging

The stream of commands to and responses from the ROE are logged as described in the base manual [8]. One may watch updates of this file with a GEIRS GUI, which opens a X-terminal executing a filtered `journalctl` on these entries. [8].

One may change the log level individually for the object files `camsend`, `camset` and `camtiming` by sending a `log` command to the interpreter shell—see the command list in the appendix of the GUI manual [8].

To aid debugging, the contents of the entries in levels 6 down to level 1 and the 'pattern' level most recently downloaded to the ROE may be summarized with the command `geirs_roeDump.pl` in the `devel` subdirectory. The command is called with a single argument which is the instrument to be debugged, for example

```
geirs_roeDump.pl Luci1
```

or

```
geirs_roeDump.pl Panic
```

and prints the patterns send to the RO, eliminating duplicates.

3 File Syntax

3.1 Command line expansion

Interpretation of the command line is done with the following steps executed in the order documented.

3.1.1 White Space and Old-fashioned Comments

The lines in the pattern files may contain comments that are first stripped off the contents:

- Letters/numbers/strings starting at and including the semicolon (;) up to the end of the line are discarded. This way of delimiting comments with the semicolon is obsolete and probably no longer found in any of the pattern files.
- White space then remaining at the start and/or end of the line is also removed.

3.1.2 Optional Repeat Count

Lines that start with the asterisk (`*`) followed by an (optionally parenthetically nested) integer expression followed by a second asterisk and more tokens have the expression between the asterisks evaluated. This `*...*` expression signals a count for a loop over the rest of the line.

- If the expression evaluates to a negative value, an error is raised. The remainder of the file is not parsed or executed.
- If the expression evaluates to zero, this line is effectively not executed, independent of what remains in the line.
- if the expression evaluates to a number ≥ 1, a repeat count applies to the rest of the line.

If the line does *not* start with a `*...*` expression, the repeat count is implicitly set to 1.

If the expression in `*...*` evaluates to 1 or 0, i.e., a boolean value, the semantics is equivalent to the singly branched if-statement of other programming languages.

3.1.3 Optional Embedded Verbose Comment

If the remaining line now starts either

```
#!verbose=on
```

or

```
#!verbose=off
```

logs are sent or not sent to the GEIRS shell that may have been opened by an operator.

3.1.4 Optional Embedded Timing Evaluation

If the remaining line contains an 'embedded' comment of the form

```
#!timing=
```

(without spaces), the followup timing expression is evaluated (see Section 4).

3.1.5 Further Comment Removal

Any characters/numbers/strings starting at a hash (sharp) mark (`#`) up to the end of the line are discarded. (At that point the embedded comments of the two formats mentioned in Sections 3.1.3 and 3.1.4 are wiped.) Note that removal of the (old) comments started with the semicolon or with the sharp *cannot* be masked by means known from other scripting or programming languages. Multiline comments or the C++/Java style comments with `//` are not supported.

Another round of removing white space at the start and end of the line follows. If the residual line is empty, the execution of this line is effectively done and the pattern parser moves on to the next line in the file. This faith happens to most of the simple free-form comment lines which start just with the hash.

3.1.6 Do Loop Expansion

If the command contains a substring of the form `do␣` (the white space after the `do` is mandatory), it is verified whether a repeat count had been found before the `do`. The repeat count has either the computed `*...*` format described above, including the implicitly defined 1, or is a variable name. If that check fails, an error is raised.

The first part of the expression after the `do` is a blank-separated loop variable—a name *without* `$` or `&`. The name is to be unique (as a syntax check of correctly nested loops) and reappears in the list of known variables (see Section 3.2.1). So the name may be used in arithmetic expressions.[1]

The next expression after the name of the loop variable evaluates to an integer, the loop counter; the remainder of the line is executed as often as requested by this counter. Concepts of individual start and skip values of the loop variable are not implemented; this is more like a `range` in Python than the flexibility of a for/while-loop in Fortran/C++/Java. (This is a loop executed within the GEIRS scanner simplifying the task of sending similar blocks of FPGA program lines to the ROE. It is *not* related to the loops in levels 2 and up of the Control-FPGA.)

Note that variables in the 'body' (after) the `do` accumulate any changes while looping. (They are passed by name, not by value.) So their values in any of the loops is whatever has been substituted (left over) by the previous loop.

Note also that the (optional) repeat count at the start of the line and the loop counter are two independent multipliers. The total number of executions of the expression after the loop counter is the *product* of both.

Do-loops may be nested, so the 'body' of the do-loop may contain another do-loop. The nesting is currently limited to 10 levels (see `MAXCMDLOOPINFO` in `camintf.h`). This containment typically occurs indirectly by an `include` command which calls another `do`-loop.

3.2 Expressions

An expression is evaluated by the following rules:

- Nested parentheses `(....(....)..(..).)` are evaluated right to left. Improperly nested parentheses — where a left-to-right scan would find more closing than opening parenthesis — and empty parentheses `()` result in errors.
- Assignments have the format

  variable`:` *value*

  or

  variable`=`*value*

  and are also evaluated right to left (if there is more than one).

  The first format with the colon (`:`) is an assignment which assumes that the variable has not been used before and is generally used to initialize auto-variables (see Section 3.2.2). The scanner checks that the variable is not known up to that point of the parsing; this helps for

[1] This is equivalent to loop variables in all major programming languages.

example to detect cyclic inclusions of files. (This check can be switched off by undefining the preprocessor variable `DEBUG_AUTOVAR_REASSIGN` in `camintf.cxx`). [2]

The second format with the `=` sign replaces the value of the variable on the left hand side by the value on the right hand side.

- State variables (indicated by a leading `$`) and auto-variables (indicated by a leading `&`) are substituted in the order right-to-left. The range of the name may be explicitly bound by curly parentheses of the format `${...}` or `&{....}`. This allows 'computed' variable names or file names of the format `${....&{...}...}`, for example. This is similar to constructs known in Tcl, Unice's shells,...

  The curly parentheses are not needed in general, because names are limited by the next white space, the next operator symbol (Section 3.2.4) or the next question mark `?`. So `&huu*...` takes the variable `huu` and not a variable `huu*`, for example.

3.2.1 State Variables

Definitions (values) of state variables are first searched in files `roe_variables.<extension>` in `pttrns`, which is e.g. `roe_variables.nirvana` for Linc-Nirvana, and if the file does not exist in `roe_variables`. The file contains zero or more lines with the following format:

- The embedded comment `#!verbose=on` or `#!verbose=off` at the start of a line switches verbosity on or off.
- Leading white space is ignored. Everything starting with `#` or `;` up to the end of a line is also ignored (comments). Leftover white space at the start or end of the line is also stripped off.
- After this comment reduction step, a matching line has the format of the variable name (starting with the `$`), one ore more blanks, and the substitutional value. (In between there is neither the colon nor the equal sign.)

  Variable names are case sensitive. The full name in the file must match, so if searching for `abeq1`, neither the value in a line starting `$abeq␣` nor a line starting `$abeq1o` is taken.

  The substitutional value may still contain embedded blanks. The main application of this feature (expressions with interlaced blanks) is to assemble the command and argument lines sent to the ROE by simply writing these side by side, separated by blanks, on the same line.
- If more than one line in the file matches the name, the first match sets the value—in a principle of early return from the file.

If the variable has not been found in the file, the second search place is the list of loop variables kept in scanner's state, introduced with a `do` (Section 3.1.6).

If the variable is not a loop variable either, a third fixed list is built into the scanner of the source code (and therefore weakly depending on which GEIRS version is run):

- `$adc36` The number of ADC36 boards in the ROE. Default values: 4 for the AIP mosaic, 2 for CARMENES and PANIC, 1 for the others. It often is the same as `$dma`, but not for example for the AIP setup where `$adc36` is 4 and `$dma` is 2.

[2] C. Storz 2013-05-16: The first format results in an empty string, where the second assignment is delivering the result to the interpreter

- `$canSrre` Boolean value (0 or 1). The 1 indicates that the current ROE and detector type support the `srre` mode, which means that the FPGA on the current ROE supports the commands related to the uploading of the associated serial registers of the chip and that the current chip type is the Hawaii-2RG or Hawaii-4RG. The main use of this variable is to skip inclus'ion of the pattern files `multi*` according to the recipe

  ```
  *($canSrre)*include multi_win_res_init
  ```

- `$chans` Number of detector channels (per chip if there is an array/mosaic). These are powers of 2 with a minimum of 1, a maximum of 32 for the Hawaii-2 and Hawaii-2RG and a maximum of 64 for the Hawaii-4RG. Further constraints exist as layed out in the reference manuals of these chips.

- `$chlines` Number of lines (slow direction) in each detector channel.

- `$chpixels` Number of pixels (fast direction) in each detector channel.[3]

- `$crep` Cycle repetition count. See the `crep` parameter of the `roe` command [8]. For the `lir` and `fullmpia` cycle types, this is the 'clean' count, not counting the first frame that will be discarded by the GEIRS process after being received.

- `$ctype` The code name of the current cycle type; one of the list in Section 2.3.

- `$dif_idle` Boolean value which indicates whether the idle-type differs from the read-without-conversion type.

- `$dma` The number of fiber/DMA channels of the data transfer from the ROE to the computer. This is generally 1 if there is one detector chip, 2 if there is more than one detector chip, but also 2 for the Hawaii-4RG PANIC default setup where the channels of the chip are distributed over two ADC36 boards.

- `$ems` Electronic multi-sampling count. 1, 2 or 4. See the `ems` parameter of the `roe` command [8].

- `$ffprot` Boolean value (0 or 1), indicating full-frame persistence protection with subwindows should be activated. See the `ffprot` parameter of the `roe` command [8].

- `$fpatall` The summatory (total) number of fast windows in all pattern windows in the slow direction.

- `$fpats`i The number of fast pattern windows in the slow pattern window number number i.

- `$gap` Remaining time up to the integration time *seconds or microseconds?* See the `gap` parameter of the `roe` command [8].

- `$hinvdir` A nonnegative integer represetning the bit pattern of HINVDIR[0..8] (Hawaii-2RG) or HINVDIR[8..9] (Hawaii-4RG, shifted right) to be placed into the register of the left-right direction of the scanner. For the Hawaii-4RG only bits 0 and 1 are relevant. for the Hawaii-2RG the relevant bits depend on the number of channels (32 channels: bits 0-7; 1 channel: bit 0; 4 channels: bits 0,3,4 and 7).

[3] For the purpose of this manual the fast direction is the horizontal direction of the detector, the slow direction the vertical direction of the detector. The appearance in the displays and FITS files may be different due to additional optional manipulations of flips and rotations selectable at GEIRS startup as detailed in the User's Manual.

- `$vinvdir` This is either the integer 0 or 1, indicating the register setting of the top-bottom direction of the vertical scanner.
- `$idle` The method to leave the idle mode, either the string `break` or the string `wait`.
- `$irmult` For the correlated reads, the number of frames. Since there are at least two frames needed for the reduction, the range of the variable is from 2 upwards.
- `$isH2RG` Indicates with a value of 1 that this is a Hawaii-2RG, and with a value of 0 that this is any other type (Hawaii-2, Hawaii-4RG, ...).
- `$isH4RG` Indicates with a value of 1 that this is a Hawaii-4RG, and with a value of 0 that this is any other type (Hawaii-2, Hawaii-2RG, ...).
- `$isHRG` Indicates with a value of 1 that this is a Hawaii-2RG or a Hawaii-4RG, and with a value of 0 that this is any other type (Hawaii-2, ...). Note that the complementary `$isH2` does *not* exist.
- `$detT` Expands to the string `H2`, `H2RG` or `H4RG` depending on the detector type.
- `$itime_cnt`i with $i = 1, 2, 3$ or 4.
- `$lskiptime` Line skip time in integer units of the base clock of this skip time. See the `lskip` parameter of the `roe` command, the output of `status roe` and the `lskp` entry of the GUI for the current base clock [8].
- `$ndet` Is the number of chips in the camera, 4 or 1 for the AIP test camera, 2 for CARMENES, 1 for all other instruments planned or in use.
- `$oflwprot` Boolean: 1 if the overflow protection against persistence effects should be activated, 0 if it should be off (deactivated). See the `oflwprot` parameter of the `roe` command [8]. This variable and `ffprot` are just flags from the point of view of the main software which can be set or reset. What they actually *mean* is entirely in the hands of the pattern definitions.
- `$padc36` The number of ADC36 boards plugged into the ROE. For a fully equipped instrument this is the same value as $adc36. For test purposes, and if compiled in the domain of a MPIA computer, a smaller value can be set by inserting a keyword with the name `P_ADC36` into the subdirectory of the `admin` directory into a file of the name of the IP address of the ROE. Currently an example is shown in `admin/192.168.3.163`. The only current use for this is to configure the pattern `roe_init` for the PANIC ROE, which requires two boards to be run, such that it can also be run with only one board and all data of the other board/fiber are fed with zeros and no concern of timeouts from that 'virtual' missing board.
- `$preadtime` Pixel read time in units of the base clock of this read time. See the `pread` parameter of the `roe` command, the output of `status roe` and the value after the `prd` label in the GUI [8].
- `$pskiptime` Pixel skip time in integer units of the base clock of this skip time. See the `pskip` parameter of the `roe` command, the output of `status roe` and the value of `pskp` in the GUI [8].
- `$ptime` Pixel time in integer units of the base clock of this time. See the `ptime` parameter of the `roe` command [8].

- `$pxllns` Appears to be the number of lines in the pattern RAM table spent for one conversion (including the time of the electronic or software multisampling). See the `pxllns` parameter of the `roe` command [8].
- `$readf` This is an abbreviation to mean the same as `readf1s1` (see below).
- `$readf`XsN with two integers $X \geq 1$ and $N \geq 1$ is the number of reads (pixels) to cover the fast hardware window number X along the slow direction.[4]
- `$reads` This is an abbreviation to mean the same as `reads1` (see below).
- `$reads`i with $i \geq 1$ is the number of reads (clocks) to cover the hardware window number X along the slow direction.
- `$shortlines` Is a power of two between 1 and the value of `chlines` (the latter is the default). It indicates that the readout (and associated resets) is not performed sequentially along the slow direction of the quadrant or detector, but that the number of pixels along that direction is divided into blocks of length `shortlines`. The readout-reset cycle is first done on the first block, then on the second and so on, such that the integration time on each pixel is 'short', that is one half, a quarter, one eights and so on compared to the standard integration time. To first order, this shuffling and subdivision leaves the cycle time unaffected.
- `$skipf` This is an abbreviation to mean the same as `skipf1s1` (see below).
- `$skipf`XsN with two integers $X \geq 1$ and $N \geq 1$ is the number of reads (clocks) to skip to reach the fast hardware window number X along the slow direction.[5]
- `$skips` This is an abbreviation to mean the same as `skips1` (see below).
- `$skips`i with integer $i \geq 1$ is the count of skips to reach subwindow i in the fast direction.
- `$spat` Number of hardware window patterns defined in the slow clocking direction.
- `$subwin` The string `_subwin` or the empty string. Useful to dispatch inclusion of files that use or do not use subwindows.
- `$tbl_idle` Integer representation of the idle type according to the enumeration in `cameratypes.h`:

  0 `ReadWoConv`

  1 `Reset`

  2 `Rlr`

  3 `Lir`

- `$time`
- `$useSHI` Boolean flag: if 1 (representing true), the variable `shortlines` is used, otherwise (if 0) the variable is not meaningful.

New variables are added (others hardly ever removed by the principle designer's programming conventions) as new detector modes are implemented.

[4] C. Storz 2013-05-16: pixels of slow window N
[5] C. Storz: 2013-05-16: of slow window N

3.2.2 Automatic Variables

Automatic variables have no value beyond the computation of the (timing) pattern. The meaning of *automatic* is that the variable's value does not extend beyond the pattern evaluation and that it is created and initialized with the colon `:` assignment, see Section 3.2.

There are two subtypes of automatic variables. The variables with a name starting with capital-`A` are volatile and the others are resident.

- The volatile variables start to be known after their first assignment and are lost (forgotten) when the UNIX/Linux process that creates them terminates.
- The resident variables are held in the shared memory data base and keep/update their values as long as the shared memory manager (of that user for that instrument) is alive, which means, between startup and shutdown of GEIRS. Sending data to the RoCon is optimized to compare new requests by the command interface—originating from the GUI's or the command interpreter/shell or external command calls through the `cmdServer`—with the most recent values maintained in the data base; if the requested parameter values are the same as those memorized, they are not forwarded to the ROE.

A maximum of 100 volatile automatic variables may be in use (variable `MAXAUTOVARINFO` in `camintf.cxx`) and a maximum of 2000 resident automatic variables (variable `MAXGLOBALVARINFO` in `camera.h`).

3.2.3 Constants

- String constants are delimited by tic marks `'.....'`.
- Integers are written in the usual ASCII representation. If the representation is in base 16, the expression must start with `0x` in front of the digits. (There is no equivalent typography for base-8 that one might expect from other languages.)
- Floating point constants are symbolized by having a dot `.` in their value. The precision of the expression is defined by the highest precision of the individual values, that is, the largest numbers of digits after the dots of any of the values that are combined with the operators (Section 3.2.4). The precision (number of digits in the sense of the `f`-format of C) of the substituted (resulting) value is set to that precision of the expression.

3.2.4 Operators

Operators look like single-letter abbreviated operators of C/C++/csh and pattern matching operations in perl, awk etc:

1. The plus `+` initializes addition.
2. The dash `-` is the minus sign or operator. Caution: In cases without leading white space like `abc-de`, the dash is part of the name. To obtain the minus-operator, insert a blank in front of the minus.

3. The star `*` is the multiplication operator. The strict difference to the star in the loop count, Section 3.1.2, is that there is no variable/value to the left of the loop count. Therefore the notation is unambiguous with respect to the meaning of the `*`-symbol.
4. The slash `/` is the division operator.
5. The percent `%` is the remainder (modulo) operator.
6. The less-than `<` is the comparison operator.
7. The greater-than `>` is another comparison operator.
8. The bang `!` is the comparison on *not* equal. (Think of the C/C++/Java style binary operator.) This is a *binary* operator with two operands, one to the left and one to the right.
9. The tilde `~` is the comparison on equal, opposite to the `!`.

The four comparison operators are also applicable to strings with the standard `strncmp(3)` library definition (where shorter strings are less than longer strings, and strings of equal length are compared left-to-right based on position of the letters in the ASCII table).

There are *no* `>=` or `<=` comparison operators.

The 9 operators are evaluated left-to-right; there is *no* standard priority like multiplication and division ranking above addition and subtractions. If order matters, an additional inclusion of the higher order operation in `(...)` ensures that the calculations are performed in the desired order.

The value of a comparison operator becomes either 1 (representing the boolean value true) or 0 (false).

The following expressions, for example, should all evaluate to 0, where `pattern` is defined to be 711 in `roe_init`:

```
*('${ctype}'~'(&pattern)')*(&Alay5Ind: &Alay5Ind+1)
*('(&pattern)'~'${ctype}')*(&Alay5Ind: &Alay5Ind+1)
*('711'~'${ctype}')*(&Alay5Ind: &Alay5Ind+1)
*('${ctype}'~'711')*(&Alay5Ind: &Alay5Ind+1)
*(${ctype}~'711')*(&Alay5Ind: &Alay5Ind+1)
*('${ctype}'!'${ctype}')*(&Alay5Ind: &Alay5Ind+1)
*('(&pattern)'!'(&pattern)')*(&Alay5Ind: &Alay5Ind+1)
*('(&pattern)'<'(&pattern)')*(&Alay5Ind: &Alay5Ind+1)
*('(&pattern)'>'(&pattern)')*(&Alay5Ind: &Alay5Ind+1)
*('${ctype}'!${ctype})*(&Alay5Ind: &Alay5Ind+1)
*(${ctype}!'${ctype}')*(&Alay5Ind: &Alay5Ind+1)
*(${ctype}!${ctype})*(&Alay5Ind: &Alay5Ind+1)
```

The standard way building an integer-based de-Morgan Algebra starts from the 4 comparison operators:

- To construct an expression which is `true` if a and b are true, the values of a and b are multiplied to form a new expression.
- To build an expression which is `true` if a or b or both are true, the values of a and b can be added and compared to zero to form a new expression.

- To build an expression which is `true` if exactly one of a or b is true, the values of a and b can be compared with the not-equal operator to form the combined expression.

If either the left or right operand of one of these operators is a hex-based integer, the result will also be written in that base with a `0x` in front.

The arithmetics is generally done in double precision mode, but if neither the left nor the right operand are floating point values, the result is converted back to integer after the calculation. This implies that the `/` is the round-to-minus-infinity integer division if the two operands are integers.

There are no trigonometric, exponential, logarithmic, number-theoretic or other functions or operations dealing with lists, vector or similar set-constructions recognized by this calculator. There are no compute-update operators like `+=` or `*=` or `++` known to other programming languages.

3.2.5 Send Expressions

A command is forwarded to the ROE if an optional expression with zero, one or two exclamation marks `!` is followed by an expression with one or two question marks. Each line of that type must—after substitution of any parentheses—represent a valid command to the microprocessor interface as specified in the RoCon drafted manual [4].

- `?` Send the following expression and wait for an answer.
- `!?` Send the following expression, wait for an answer, and do not check the answer.
- `!`n`!?` Send the following expression, wait for an answer, re-try this up to n times, and do not check the answer.
- `!-`n`!?` Send the following expression, wait for an answer, retry this up to n times, and check the answer.

The timeout in these 4 formats may be changed by using two—not one—question marks with a timeout (an integer, in units of seconds) in between: `?`*seconds*`?`. That specification of a timeout becomes global and permanent (until revoced by the next explicit modification) because it is stored in the variable `CAMCURTOUT` of the shared memory data base.

3.2.6 Include Expression

The expression of the form `include` followed by white space and a filename—without the instrument suffix—starts to scan that file from the top and returns to the file of the `include` that started this discursion. The log files (Section 2.4) write `Start` and `__End` marks when entering and exiting the files.

The dependencies in the current `pttrns/Nirvana` directory are shown in Figures 1 and 2. The arrow point from the files that include others (noted without the suffix) to the files that are included. Some files appear in disconnected clusters from others. Some of these are orphaned and remain in the SVN for unspecified reasons, but others only *appear* to be orphaned and are actually connected to the full graph by the variable extension mechanism described on page 10.

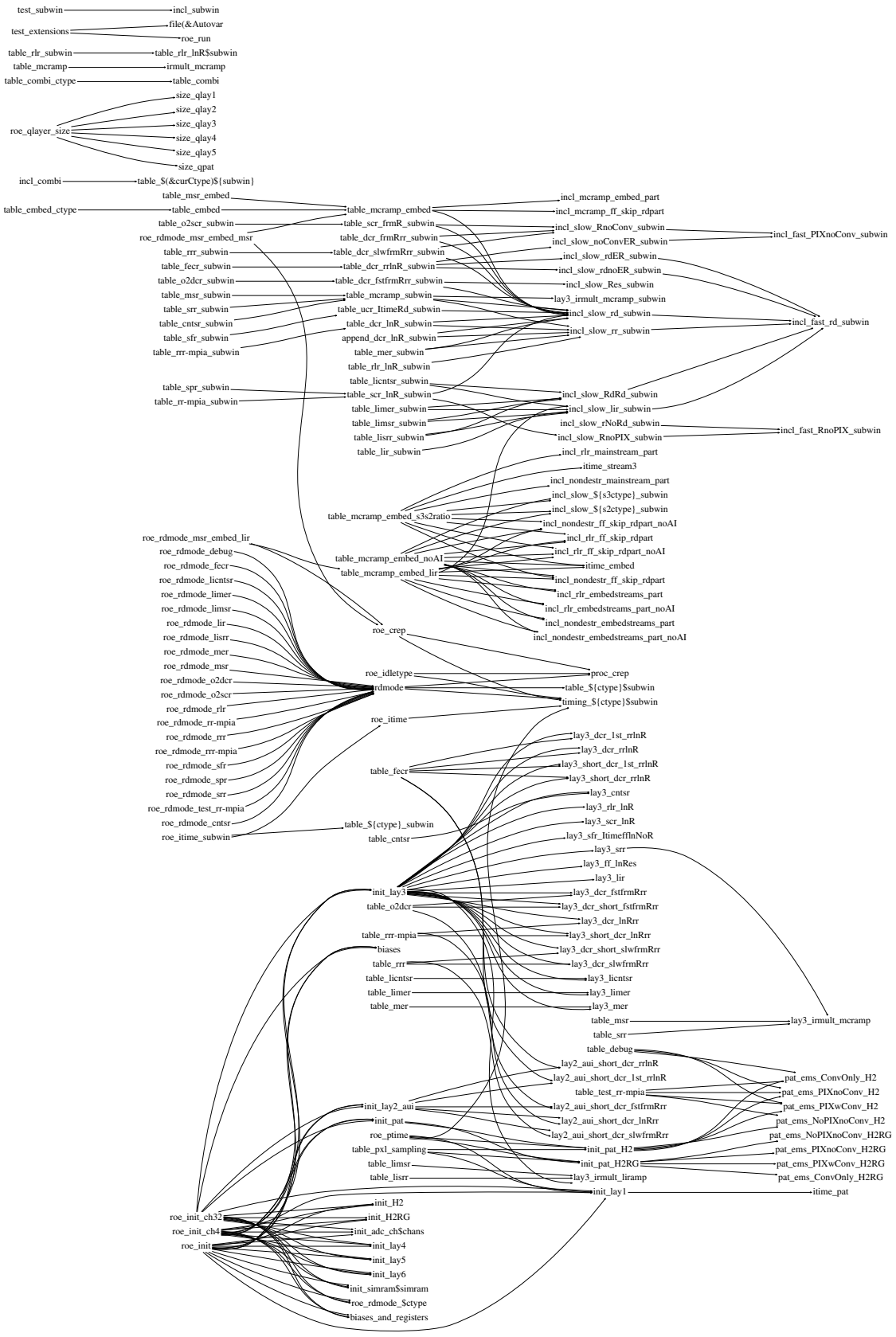


Figure 1: Mutual inclusion of files by other (non-timing) files.

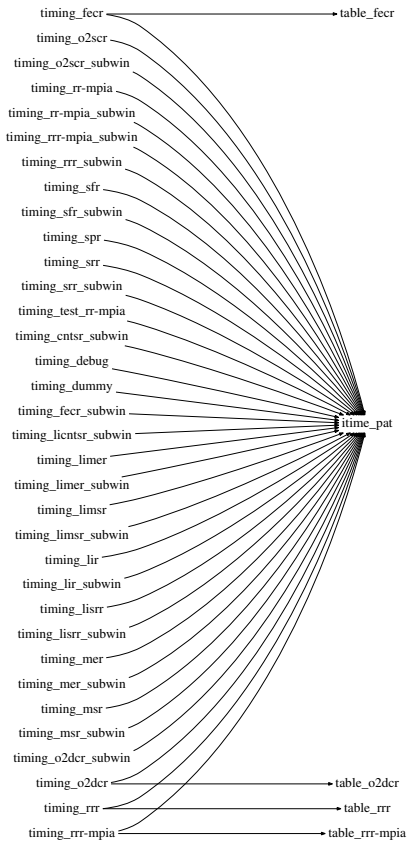


Figure 2: Mutual inclusion of files by timing files.

4 TIMING CALCULATIONS

4.1 Aim

The timing calculations accumulate the clock cycles of the stages of (virtually) running a pattern. In that sense they are informative and do not generate ROE command lists, but serve to predict the total time spent in readout cycles depending on chip capabilities, windowing, readout modes, ROE base clock frequencies etc., and to calculate how many idle-clocks must be inserted in the patterns to fill up the (operator's specific) integration time. If there is demand of fixing the duration of the readout cycles, such calculation also allows to defer an integration time by calculation of the overhead times as a function of all these parameters.

In summary, this part of the GEIRS software predicts how much time will be spent on which stages/levels of the ROE's pattern generator, but the *actual* timing is independent of these calculations. The result of summing up all the loops and steps down to the lowest (RAM) level of the pattern program in the associated FPGA of the ROE over one frame of the exposure appears as the cycle time of the control GUI [8].

4.2 Timers

The implementation defines a set of named timers which are individually started, incremented and stopped as a function of tracing the actions of the pixel reads and skips. The names of the timers are used *without* the `$` or `&` of the other variables (those of Sections 3.2.1 and 3.2.2). The unit of values of timers is microseconds.

Access to the timer list with its variables is exclusively achieved by using the subcommands of the embedded timing comment. So the timers are entities separated from, but have access to the variables of the pattern constructor (Sections 3.2.1, 3.2.2). There is a close handshake with the variables of the main pattern construction. The position of the timing commands in the pattern files of (Section 3.1.4) defines the values of the variables that are the basis of the timing evaluations.

The evaluation of a line with a `timing=` command is done by dropping everything starting with any optional second `#` character (comments) that may follow the initial `#` at the start of the line (Section 3.1.3), and then looking for one of the following subcommands (Sections 4.3.1–4.3.7) after the `=`-character.

4.3 Subcommands

4.3.1 set

The subcommand `set` in the timing calculation sets the timer value named after the `set` to the value of the expression of the rest of the line. If the named timer does not exist yet, it is created, obtains the value, and is marked as deactivated.

4.3.2 define

Define followed by a timer's name uploads the value of the timer to the shared memory data base, so it basically lets know the other parts of the software (command interpreter and data read-interface) what the value is. If no name follows the `define`, all timer's values are published that way.

The following list of timers is actually known in the shared memory data base, i.e., fixed in the scanner: `ctime`, `creptime`, `roitime`, `gapitime`, `skiptime`, `shortflag`, `irmult`, `Nreads`, `patitime`, `itime`. Any other names after the `define` will not be recognized.

Note that the command interpreter's `pipe` command (Appendix in [8]) may send bare formatted commands to the ROE which potentially change the state of the ROE without leaving any traces in the data base or the pattern generator's variable tables.

4.3.3 state

Logs the values of the current set of timers.

4.3.4 add

The subcommand `add` in the timing calculation increases the timer value with the name after the `set` by the value of the expression in the rest of the line. Warning: If the name of the timer is empty, *all* timers are incremented by the value — which is probably not what you want.

4.3.5 on

Activates the timer with the name after the `on` so it will be modified/effected by subsequent commands. If used without the name of a timer, all timers are activated.

4.3.6 off

Deactivates the timer with the name after the `off` so its value is frozen until re-enabled. If used without the name of a timer, all timers are deactivated.

4.3.7 end

Forgets (removes) all timers, as if they never had been created with the `set`.

4.4 Functions

Two functions with one respectively two arguments may appear in the expression of the subcommands `set` or `add`. The functions are `range(...,...)` and `timeof(...)`; each of them returns a floating point value.

- There is no evaluation of expressions within the pair of parentheses that encompasses the one or two arguments!
- No white space between the function name and the left, opening parenthesis!

4.4.1 range

The two arguments of `range` must be two auto-variables in the sense of Section 3.2.2, which each contain one common substring which is either `pat`, or one of the 6 possible `lay`i with $1 \leq i \leq 6$, or `proc`. The values of these variables are supposed to be two non-negative integers, a start and a stop (line) address of a 'program' (command) in the detector FPGA, which refer as usual to a sequence of commands in the next lower layer.

The value of the function is the duration of executing this part of the detector program, recursively including the times of the sub-loops and auto-increment loops in the lower layers.

4.4.2 timeof

Reads the current value of the timer with the name which appears as the argument in the function.

5 DETECTOR WINDOWS

5.1 Principles of Operation

GEIRS spans a 2D coordinate system across the detector area, which defines the user's coordinates of windows (rectangular subareas). For cameras with single chips (like LINC-NIRVANA), the origin

of coordinates is in the lower left edge of the chip, for mosaics (AIP, CARMENES) the origin is in the lower left edge of the lower left chip and stretches seemlessly—without noticing the detector gaps—to the upper right corner of the upper right chip.

Windows defined in this coordinate system are transformed by GEIRS to a symmetry-adapted set of *pattern windows* by (i) copying the windows to each chip for multi-chip cameras and (ii) replicating the windows in each quadrant for detectors with quadrants (Hawaii-2), and (iii) clamping the windows into a single channel (Detector channels and ADC channels match for all current implementations which aims at maximum parallelization of readout and ADC operation or shortest possible integration times, respectively.) This implements a sychronous timing of read-out and data transfer, defines the interface relevant to the pattern construction, and settles the speed (exposure time, frame rate).

Finally there is a set of *detector windows* which is obtained by (i) replicating the *pattern windows* across all channels and (ii) merging/glueing (where possible) windows that are direct neighbours. This happens within GEIRS upon receipt of the data and does not effect the patterns. Mapping the union of the pixels of these *detector windows* data back onto the user's coordinate system may yield a very rugged, complicated and fragmented tiling with holes and/or non-connected patches.

The most efficient way of saving these data would be an option to deliver a FITS file with a single frame defined by a bounding box around all of these, filling holes with the `BLANK` value. (*Most efficient* means that all available ADC values are saved; disk space consumption is not optimized because holes are also covered.) The actual methodology implemented in GEIRS removes all pixels data that are not in any of the user's windows and stores only the user's windows into the FITS files.

5.2 Example

An example of the subwindow parameters is given next to illustrate the parameters `readf`X`s`N and `skipf`X`s`N of Section 3.2.1. We assume that the user has requested a single software window with a lower left pixel at [700,900] and an upper right pixel at [1209,989] with

```
> subwin clear
> subwin SW 1 700 900 510 90
> subwin auto on
```

which is the solid rectangle crossing from quadrant II to quadrant III in Figure 3. For Hawaii-2 detectors (including LINC-NIRVANA), GEIRS adds symmetric copies to the other quadrants by three times 90 degree rotation around the center of the detector such that each quadrant is covered by the same number and shape of windows. The principle of symmetry is: if a pixel must be read in one channel, the equivalent pixels in all the other 31 channels must also be read. This adds the dashed windows in Figure 3, which may overlap. (This step is absent for detectors without quadrants like the Hawaii-2RG.)

The output of the `status subwin` then shows

```
subwin SWwin: id=1 xs=700 ys=900 xw=510 yw=90
subwin DETwin: id=0 fs=36 ss=990 fw=90 sw=35
subwin DETwin: id=1 fs=36 ss=700 fw=90 sw=200
subwin DETwin: id=2 fs=1 ss=900 fw=128 sw=90
```

which is illustrated for the pattern windows numbered 0 to 2 in Figure 4. This represents a 128-

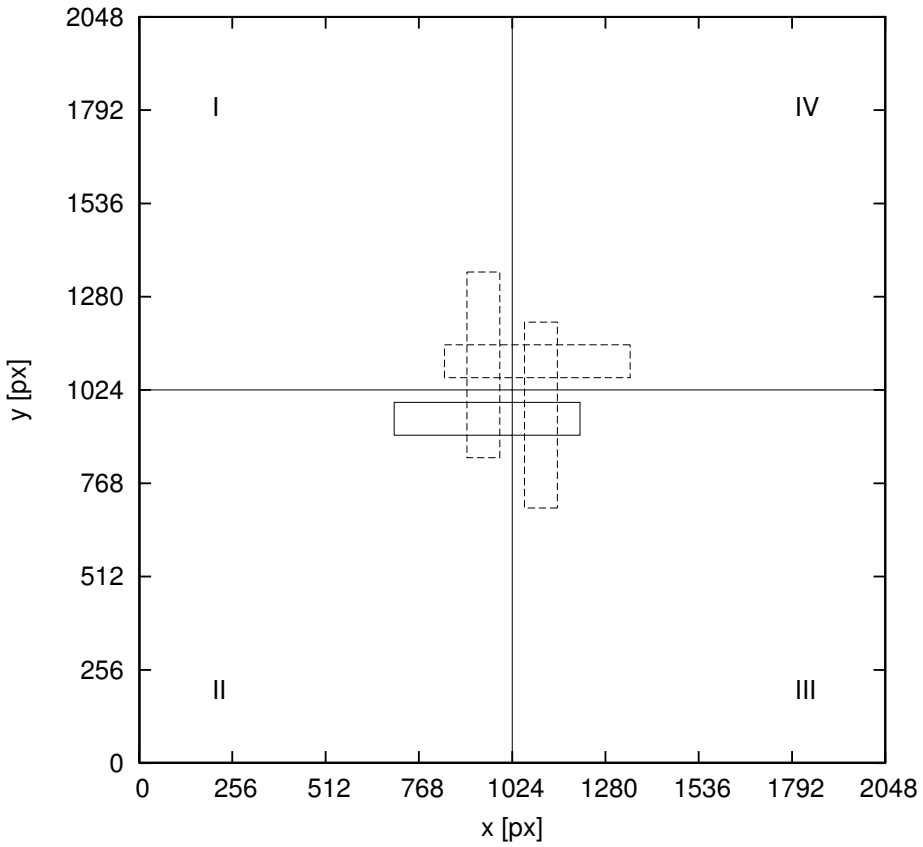


Figure 3: Example of a software window requested (horizontal quadrangle) and three symmetry-adapted, rotated copies (dashed quadrangles) added.

wide slice through one (any one) quadrant along any of the 8 channels after the pixels in use are copied 8-fold to all channels: second application of the principle of symmetry.

In the first quadrant, the fast direction is along the x axis, the slow direction along the y axis. The geometries of the three non-overlapping windows are characterized by a starting pixel index in the slow direction (`ss`), a starting pixel index in the fast direction (`fs`), a width along the slow direction (`sw`), and a width along the fast direction (`fw`). The clocking of the detector pushes the charges to the detector edge, so the pixels of $y = 2048$ of quadrant I are received first and the pixels of $y = 1025$ are received last in time. The values of the *skip* variables refer to that order in time, so in quadrant I `ss` is counted from the top edge, and `fs` from the right border of each channel. (Note that this relation between timing and geometry is not necessarly fixed for the Hawaii-2RG chips, where the order along both directions, the fast and the slow, may be reverted/flipped by appropriate setting of registers.)

Slicing this intermediate result into channel boundaries (each channel of width 128 in the fast direction) and 8-fold replication along the fast direction fuses the windows with `id=2` into a single long horizontal bar across the entire first quadrant, and creates 8 copies of `id=0` and `id=1`. These 17 windows are effectively 'executed' in parallel in the other quadrants, which gives Figure 5. This pattern of an overlay of rectangular windows that are sent from the ROE to the GEIRS software on the workstation can also be investigated by selection of the `show all RO information` in the

fourth menu at the top of the display GUI [8]. An example of this feature of the GEIRS engineering GUI has been shown in a SPIE article [7, fig. 5].

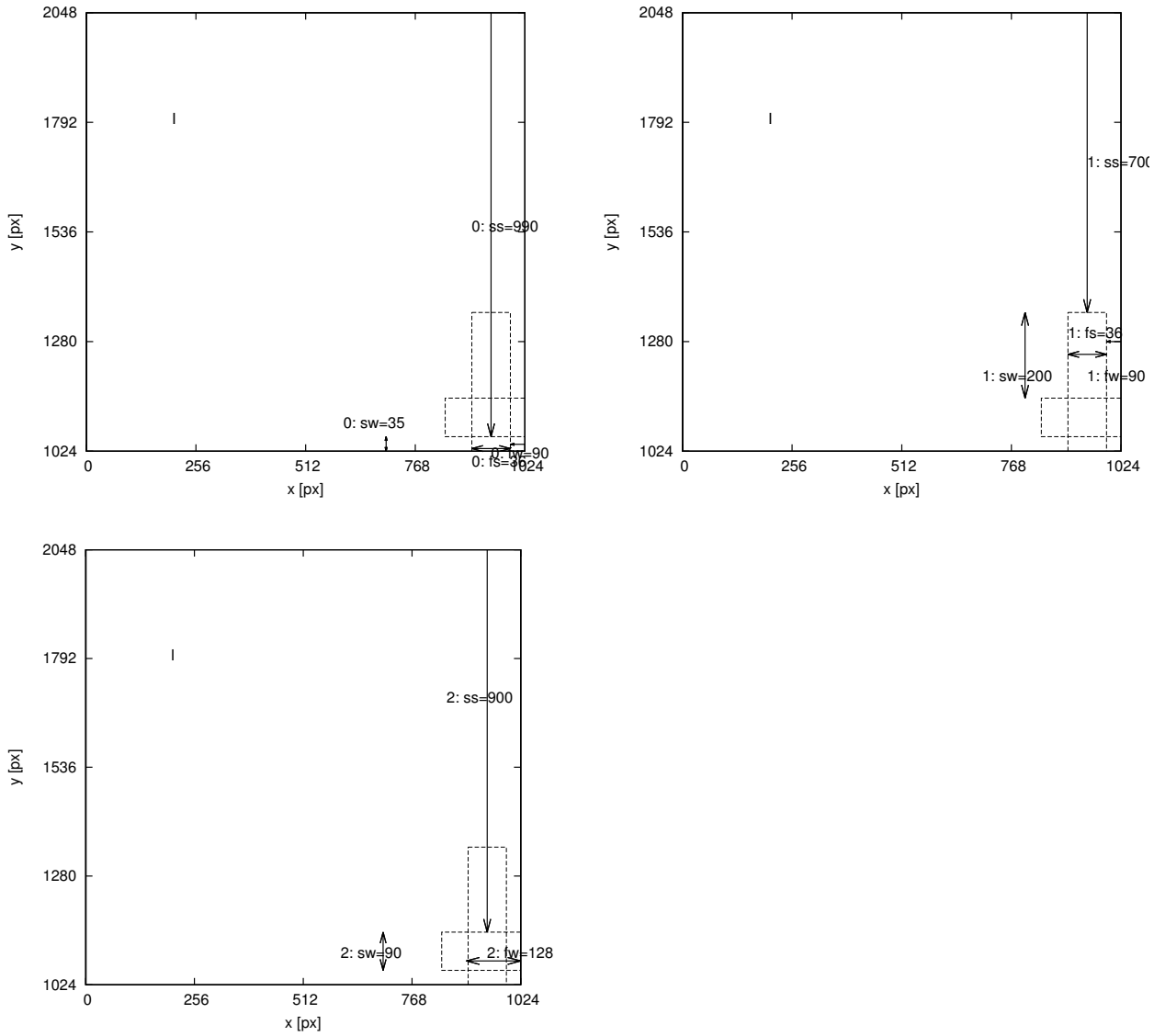


Figure 4: Geometry of the 3 pattern windows labeled `id=0`, `id=1` and `id=2` within the first quadrant of Figure 3. Each window is characterized by a s(tart) and a w(idth) coordinate in the f(ast) and s(low) direction. The window with index 2 is actually wider than 128, but in these cases where the window is wider than the channel width of 128 pixels, the width is reduced to 128 in preparation for the next step.

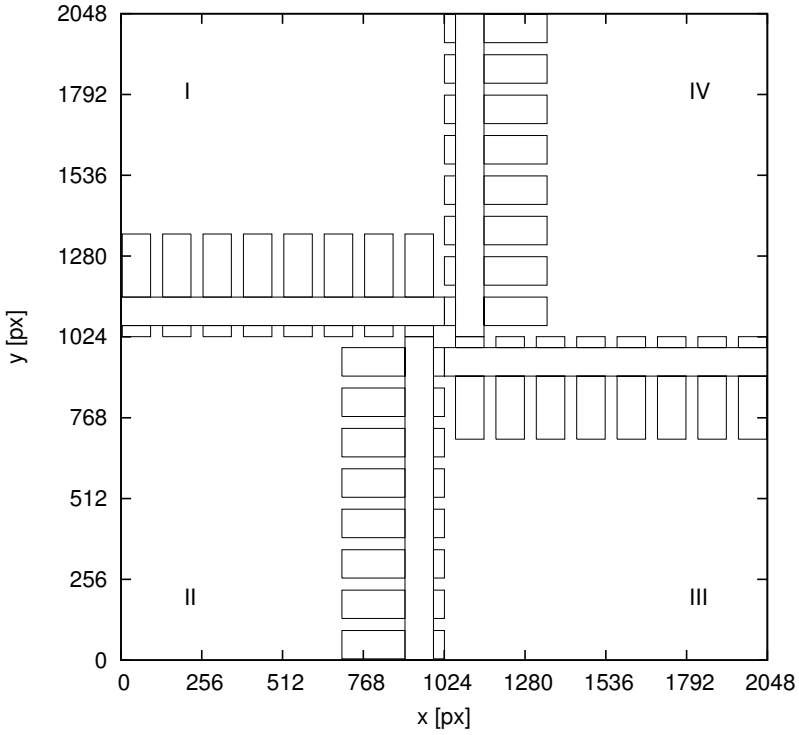


Figure 5: $4 \times 17 = 68$ detector windows reported by `status subwin` for the software window of Figure 3. 8 copies of the windows of Figure 4 are created along the fast direction, where 8 is the number of detector and ADC channels in each quadrant.

6 PATTERN SCRIPTING

6.1 Auto-increment Layer-2 Loops

Since early 2010 the functionality of the layer-2 commands (command number 713) of the Detector-FPGA has been enhanced by two further parameters after the *loop* count. These are called *increment* and *limit*. If the value of *increment* is zero, executing the layer-2 program (by a layer-3 program) executes the layer-1 program as many times as specified by the *loop* parameter. Otherwise the *increment* is a signed integer that works like an arithmetic increment/decrement of the *loop* variable by that amount after each execution of the layer-2 program. The first call of the layer-2 program by the layer-3 program executes the layer-1 program *loop* times, the second call excutes the layer-1 program *loop+increment* times, the third call `loop+2×` `increment` times and so on, up to and including the case where *loop+l×increment*≤*limit* for *increment*> 0 and up to and including the case where `loop-`$l \times$`|increment|` ≥`limit` for *increment*< 0. Once the loop count has surpassed the *limit*, it is reset to the *loop* parameter—with the same type of crescendo of loop counts following in further executions of the layer-2 program.

We see that the case where the *increment* parameter is zero is not special with respect to changing the loop counter.

- If the *increment* is non-zero and *loop* equals *limit*, the loop counter is effectively always the *loop* parameter—because at the first executation of the layer-2 program always is that value, and because already the next excuation resets the 'hidden' counter to *loop*.
- If the sign of the *increment* differs from the sign of the arithmetic difference *limit-loop*, the reset condition is fulfilled after each execution of the layer-2 program, and the `loop` counter is always the same for execution of the layer-1 program.

The effective offset $l\times$ *increment* is reset to $l = 0$ if

1. the entry in layer-2 is overwritten in conjunction with a new 713 command—this implies stopping the process and resetting it and is just a further instance of the reset cause further down,
2. or as the 730 command resets the entire layer,
3. or as the command 740 starts the pattern process
4. or as the command 741 stops the pattern process

The EXIT-loop command 742 does not alter the loop arithmetics of the *loop*, *increment* and *limits*. One may think of the auto-increment functionality as some hidden but global 'static' variable which remembers its history without taking notice of the EXIT-loop command.

6.2 Hawaii 2 (i.e., LN)

6.2.1 RAM Layer Command Format

In the pattern-layer of the detector-FPGA, the 711 command to define one word (out of up to 1024) has the format [4]

```
711 0 address 0x0to15 0x16to31 0x32to47 0x48to63
```

where the first parameter (the zero) is not used, and 4 groups of 16 bits each carry a load of 64 bits, starting with the least significant bit in the leftmost group.

6.2.2 Initial Pattern

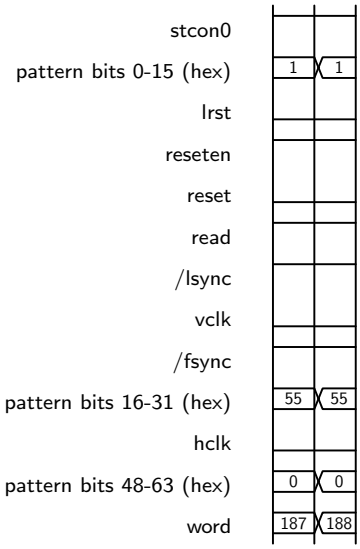
The functionality to the load of 64 bits of the 711-command parameters to detector pins:

signal	hclk	lrst	reseten	reset	read	(hclk)	/lsync	vclk	/fsync	stcon1	stcon0
bit	48	23	22	21	20	19	18	17	16	1	0

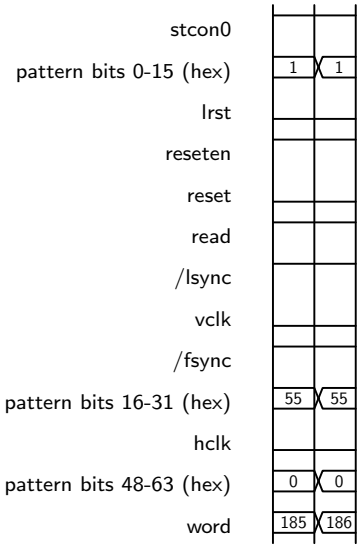
The signals in parentheses, `hclk`, is a leftover from older ROEs (?) and can effectively be ignored for the LN case.

After booting (starting GEIRS), the ROE registers contain the following bits in the detector-FPGA RAM layer. Words are enumerated from 0 to 1023, the maximum capacity of the FPGA, but less than 200 are actually defined at that point in time. The `(stcon1)` and `(hclk)` signals at bit 1 and 19 are not plotted and should effectively considered always to have bit-value zero.

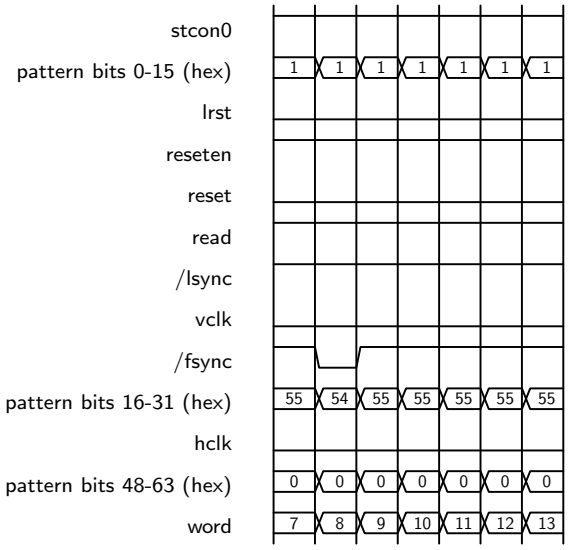
Defined in `init_pat_H2` with the name `patSetup2`:



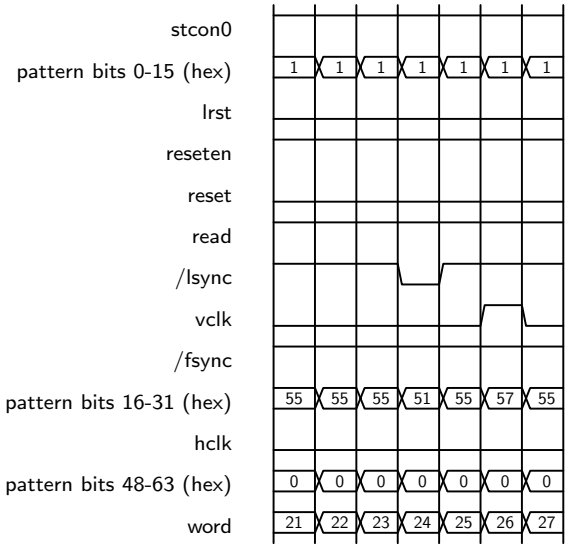
Defined in `init_pat_H2` with the name `patNone`:



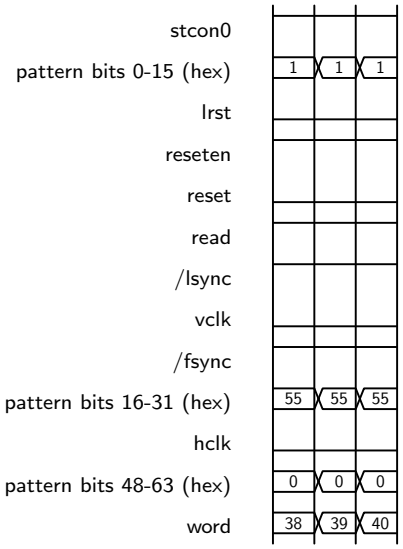
Defined in `init_pat_H2` with the name `patFSync`:



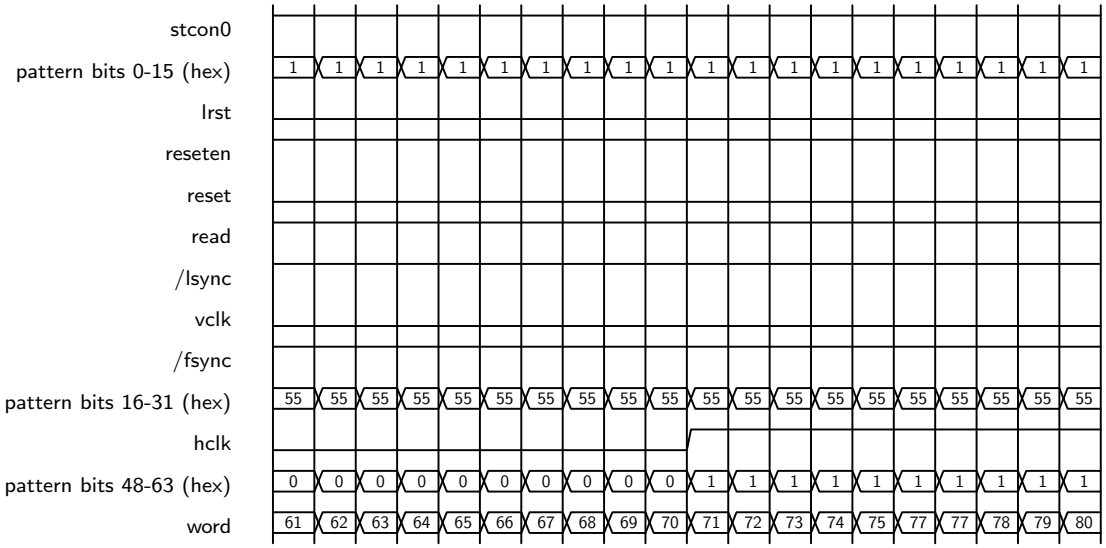
Defined in `init_pat_H2` with the name `patVLS`:



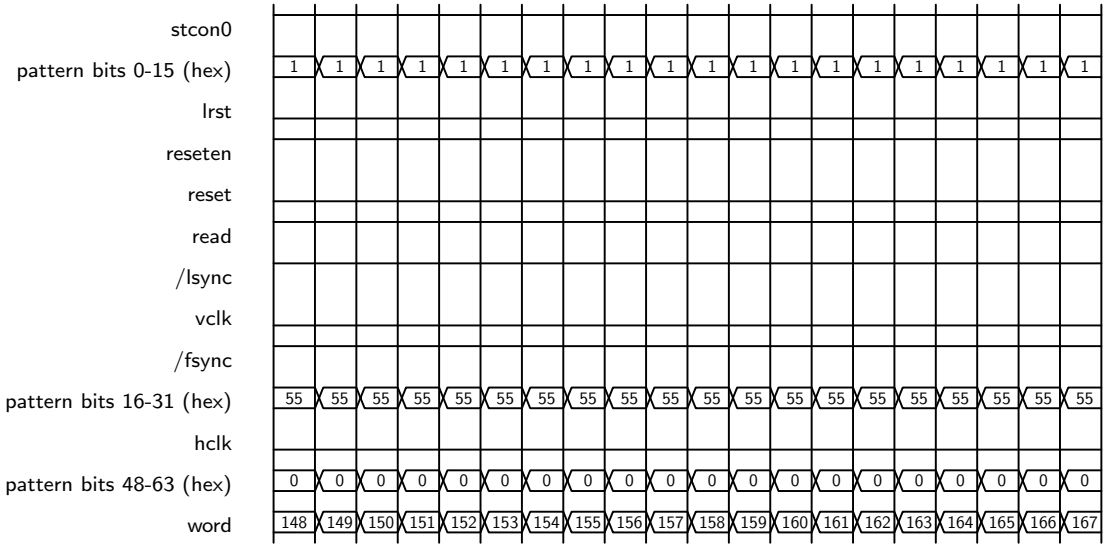
Defined in `init_pat_H2` with the name `patNoRes`:



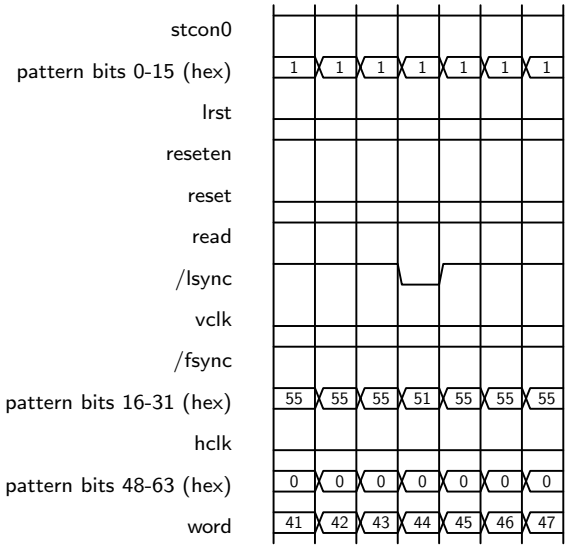
Defined in `init_pat_H2` with the name `patPIXnoConv`, where the value of `pxllns` is effectively 7 for `ems=1` as discussed in `camintf.cxx`, and including `pat_ems_PIXnoConv_H2`:



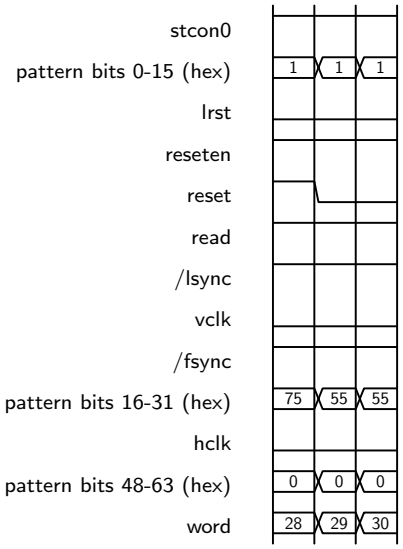
Defined in `init_pat_H2` with the name `patNoPIXnoConv`, where the value of `pxllns` is effectively 7 for `ems=1` as discussed in `camintf.cxx`, and including `pat_ems_NoPIXnoConv_H2`:



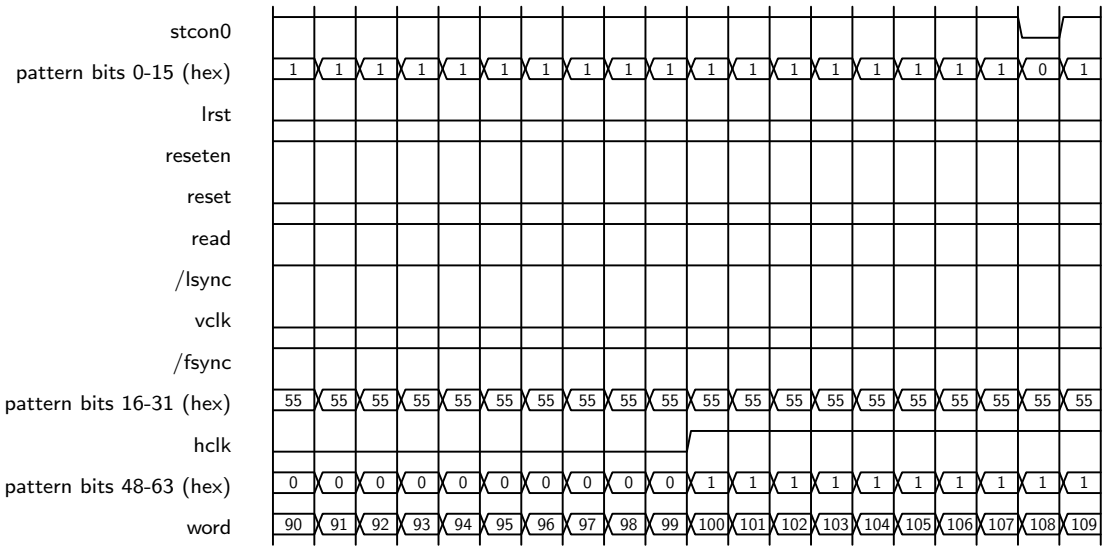
Defined in `init_pat_H2` with the name `patLSyncNoRes`:



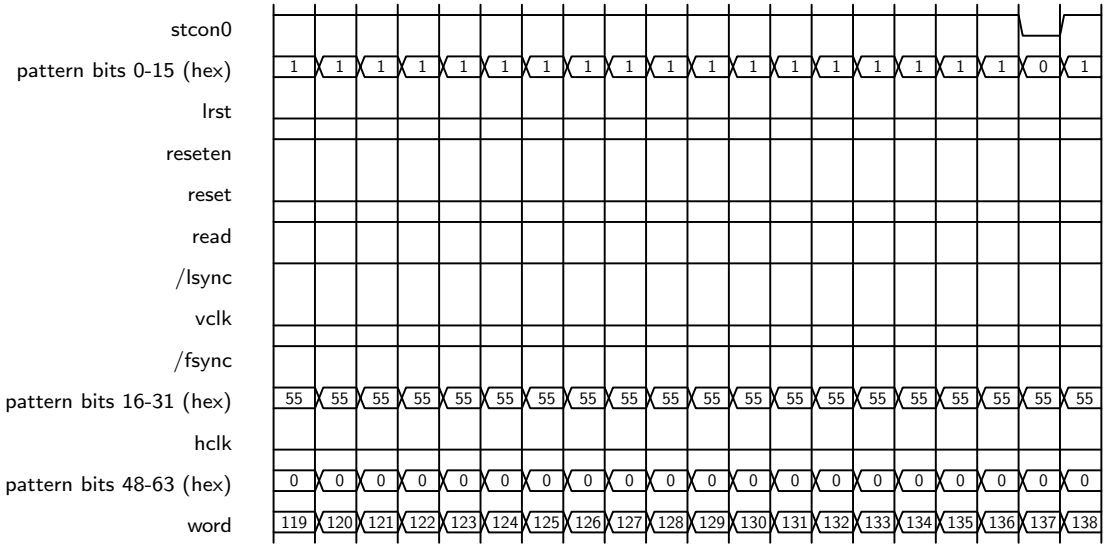
Defined in `init_pat_H2` with the name `patRes`:



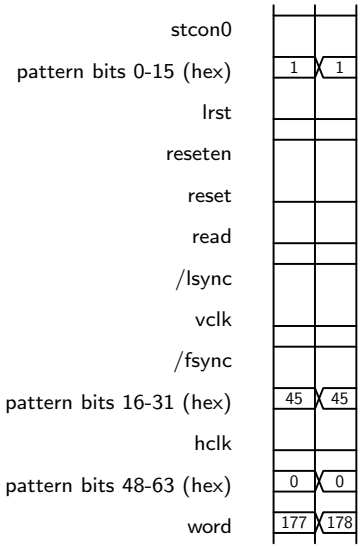
Defined in `init_pat_H2` with the name `patPIXwConv`, where the value of `pxllns` is effectively 7 for `ems=1` as discussed in `camintf.cxx`, and including `pat_ems_PIXwConv_H2`. The *falling* edges of the hclk increase the pixel address, so the placement of the rising edge in the middle of the pattern is almost arbitrary:



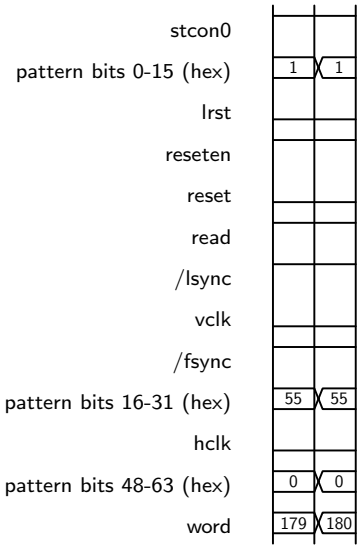
Defined in `init_pat_H2` with the name `patConvOnly`, where the value of `pxllns` is effectively 7 for `ems=1` as discussed in `camintf.cxx`, and including `pat_ems_ConvOnly_H2`:



Defined in `init_pat_H2` with the name `patItime`:



Defined in `init_pat_H2` with the name `patItimeRDen`:



Note that there are holes in the words explained above, where other patterns are defined/uploaded but not yet relevant after boot time. They exist in preparation for subwindows, electronic multi-sampling or other parameters that may be relevant when the operator starts to use the detector. Note also that there are some redundant pieces here.[6]

1. Number 0 keeps all signals frozen (at 616 nsec clock) and enabled as in `patSetup2`, but may toggle from an infinite loop to a single execution in layer 3:

```
718 0 0 0 1 1
        717 0 0 0 3 1
                715 0 0 0 1 65535
                        714 0 0 0 1 1
                                713 0 0 0 1 1 0 0
                                        712 0 0 187 188 1 60 1 4088
                                                patSetup2
                715 0 1 1 2 1
                        714 0 1 1 2 1
                                713 0 1 1 2 1 0 0
                                        712 0 1 187 188 1 60 1 4088
                                                patSetup2
                715 0 2 0 1 65535
```

2. Number 1 is a single action that disables the `stcon`s by masking a byte (selected with a specific bit in the last parameter, 4080) (51 kHz clock):

```
718 0 1 4 5 1
        717 0 4 3 4 1
                715 0 3 7 8 1
                        714 0 7 3 4 1
                                713 0 3 3 4 1 0 0
                                        712 0 3 185 186 1 4 1 4080
                                                patNone
```

3. Number 2 is an infinite loop, containing two sequential subroutines in layer 4. The first subroutine is essentially a frame sync followed by 1024 advances along the slow direction

[6] e.g. because patNone and patSetup2 are the same, lay1ConvOn and lay1Setup2 are actually the same pattern.

(where 1024 is the number of pixels along each of the 32 channels of the Hawaii 2), and the second subroutine sets the bit for the break point ID 1 (represented by the final 8 in the 712 command). Each advance along the slow direction contains $1 + 127 = 128$ horizontal clocks without triggering ADC's—where 128 is the width of a channel for the Hawaii 2—a line sync and reset, and again 128 horizontal clocks, altogether a LIR pattern without conversion:

```
718 0 2 3 4 4294967295
        717 0 3 9 11 1
                715 0 9 5 6 1
                        714 0 5 28 30 1
                                713 0 28 16 17 1 0 0
                                        712 0 16 7 13 1 60 1 0
                                                patFSync
                                713 0 29 56 66 1024 0 0
                                        712 0 56 21 27 1 60 1 0
                                                patVLS
                                        712 0 57 38 40 1 60 1 0
                                                patNoRes
                                        712 0 58 61 80 1 51 1 0
                                                patPIXnoConv
                                        712 0 59 61 80 127 51 1 0
                                                patPIXnoConv
                                        712 0 60 148 167 1 51 1 0
                                                patNoPIXnoConv
                                        712 0 61 41 47 1 60 1 0
                                                patLSyncNoRes
                                        712 0 62 28 30 1 60 1 0
                                                patRes
                                        712 0 63 61 80 1 51 1 0
                                                patPIXnoConv
                                        712 0 64 61 80 127 51 1 0
                                                patPIXnoConv
                                        712 0 65 148 167 1 51 1 0
                                                patNoPIXnoConv
                715 0 10 6 7 1
                        714 0 6 2 3 1
                                713 0 2 2 3 1 0 0
                                        712 0 2 185 186 1 4 1 8
                                                patNone
```

This implements the initial default of the idle loop (`wait` in `Lir` mode).

4. Number 3 enables all outputs with a 4088 value of the last parameter, so it would revert the status left by Number 1:

```
718 0 3 5 6 1
        717 0 5 4 5 1
                715 0 4 8 9 1
                        714 0 8 4 5 1
                                713 0 4 4 5 1 0 0
                                        712 0 4 185 186 1 4 1 4088
                                                patNone
```

A subsequent `read` will use the 764 command with the break point in the Program 2 to execute the fourth to last pattern in an endless loop (`764 0 3 7 65535 1`).

5. Number 4 is like Number 2, but the conversion (ADC's triggers) are enabled in the subroutine that walks through the 1024 pixels along the slow direction, and there is a second walk through the 1024 pixels along the slow direction. Before the step that disables the byte with the conversion bit are three actions in layer-1 with zero loop count, and one action at divisor 899 and pre-scaler 2 (9.09 msec), effectively zero additional integration time:

```
718 0 4 12 14 1
        717 0 12 21 23 1
                715 0 21 27 29 1
                        714 0 27 30 32 1
                                713 0 30 16 17 1 0 0
                                        patFSync
                                713 0 31 36 46 1024 0 0
                                        712 0 36 21 27 1 60 1 0
                                                patVLS
                                        712 0 37 38 40 1 60 1 0
                                                patNoRes
                                        712 0 38 61 80 1 51 1 0
                                                patPIXnoConv
                                        712 0 39 90 109 127 51 1 0
                                                patPIXwConv
                                        712 0 40 119 138 1 51 1 0
                                                patConvOnly
                                        712 0 41 41 47 1 60 1 0
                                                patLSyncNoRes
                                        712 0 42 28 30 1 60 1 0
                                                patRes
                                        712 0 43 61 80 1 51 1 0
                                                patPIXnoConv
                                        712 0 44 90 109 127 51 1 0
                                                patPIXwConv
                                        712 0 45 119 138 1 51 1 0
                                                patConvOnly
                        714 0 28 34 35 1
                                713 0 34 12 16 1 0 0
                                        712 0 12 177 178 0 999 3 0
                                                patItime
                                        712 0 13 177 178 0 99 2 0
                                                patItime
                                        712 0 14 177 178 0 4 1 0
                                                patItime
                                        712 0 15 179 180 1 899 2 0
                                                patItimeRDen
                715 0 22 6 7 1
                        714 0 6 2 3 1
                                713 0 2 2 3 1 0 0
                                        712 0 2 185 186 1 4 1 8
                                                patNone
        717 0 13 23 24 1
                715 0 23 47 48 1
                        714 0 47 30 32 1
```

The readout of the 128 pixels in the horizontal direction of a channel is split into 1× `patPIXnoConv`, 127× `patPIXwConv` and 1×`patConvOnly`, so the pixel 'address' stands at the 'end' of the line at the start of the (optional, residual) integration time.

After splitting the residual integration time into three different intervals (at 10.1 sec, 1.01 msec, 50.5 nsec with three different prescaler choices) the main program will later on overload the layer-1 commands (words 12, 13 and 14) at the three actions to realize any integration time that is longer than the minimum integration time.

6. Number 5 is the same as Number 1:

```
718 0 5 4 5 1
```

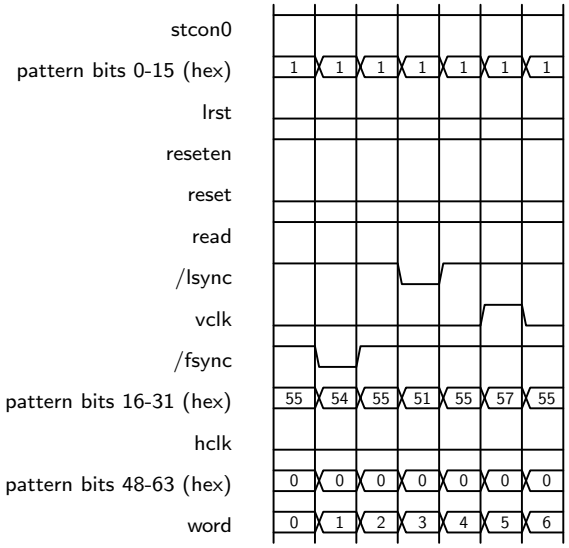
This means after exit of the read-loop of the Number 4 the idle loop is started.

7. Number 6 is the same infinite loop as Number 2:

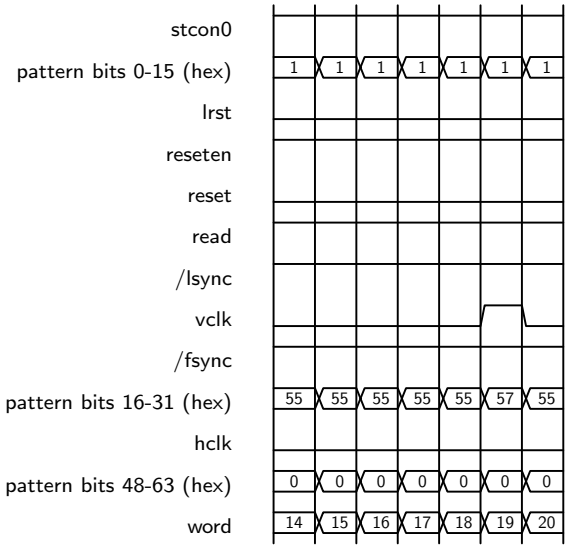
```
718 0 6 3 4 4294967295
```

6.2.3 Disconnected Patterns

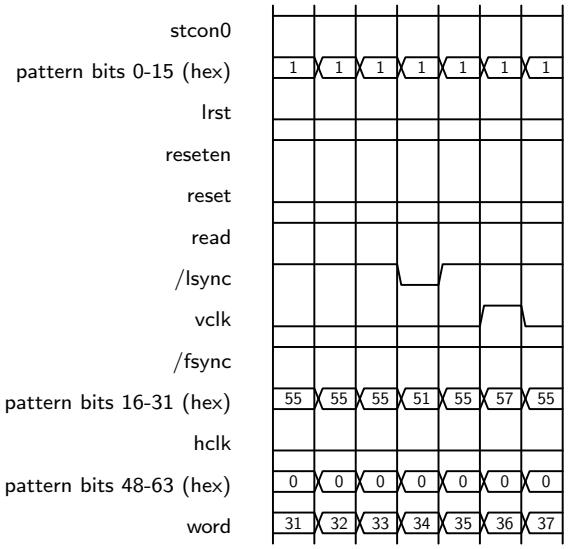
The patterns not linked to any higher layer but fillers uploaded after initialization are: Defined in `init_pat_H2` with the name `patFSwVLS`:



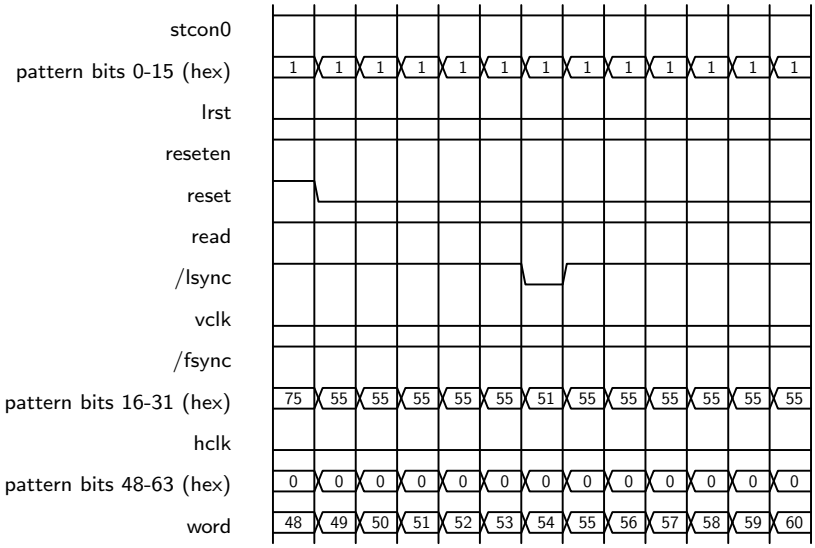
Defined in `init_pat_H2` with the name `patVSync`:



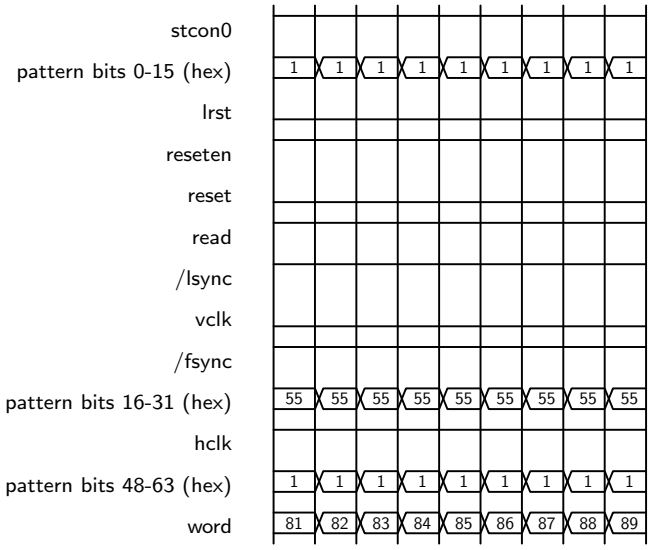
Defined in `init_pat_H2` with the name `patVLSnoRes`:



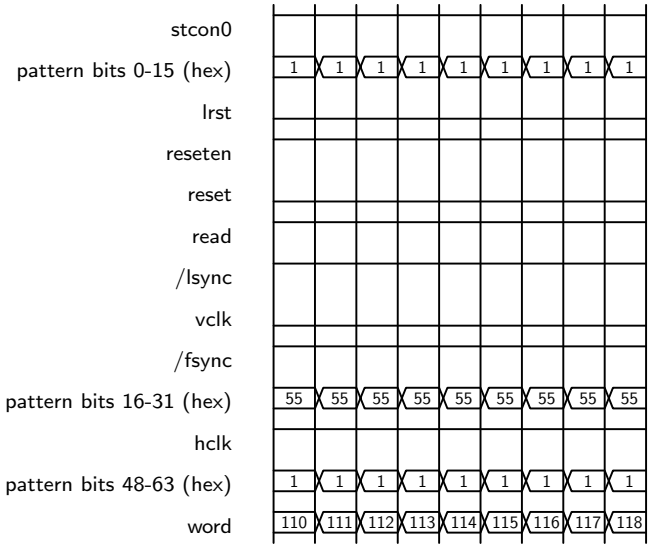
Defined in `init_pat_H2` after `patENDLSync`:



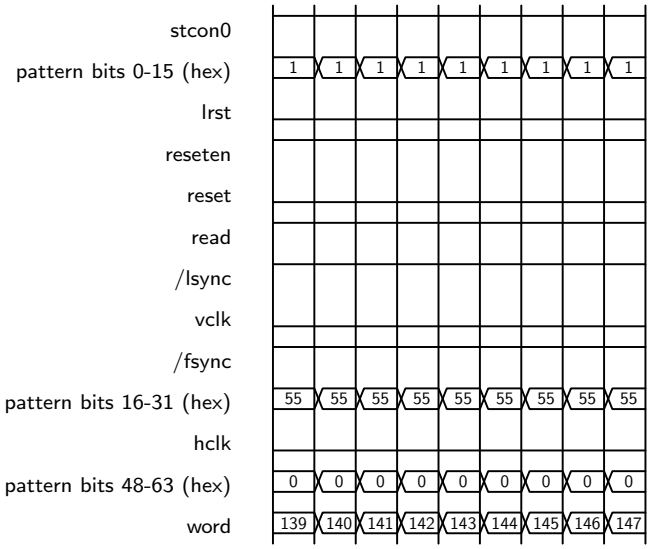
Defined in `init_pat_H2`:



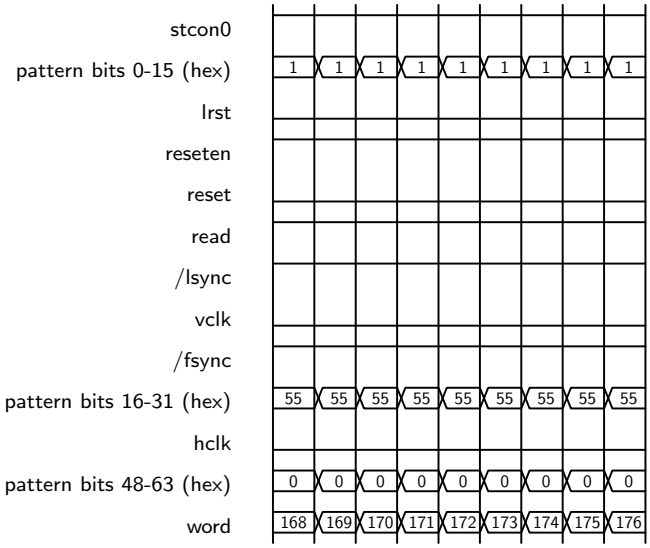
Defined in `init_pat_H2` (same as in words 81–89):



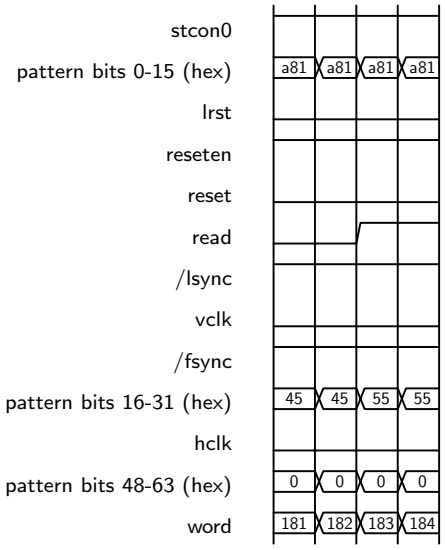
Defined in `init_pat_H2`:



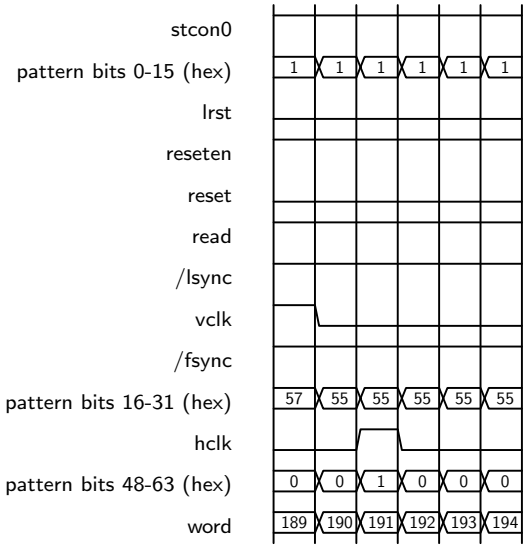
Defined in `init_pat_H2` (same as in words 139–147):



Defined in `init_pat_H2` as `patItimeLED` and `patItimeRDenLED`:



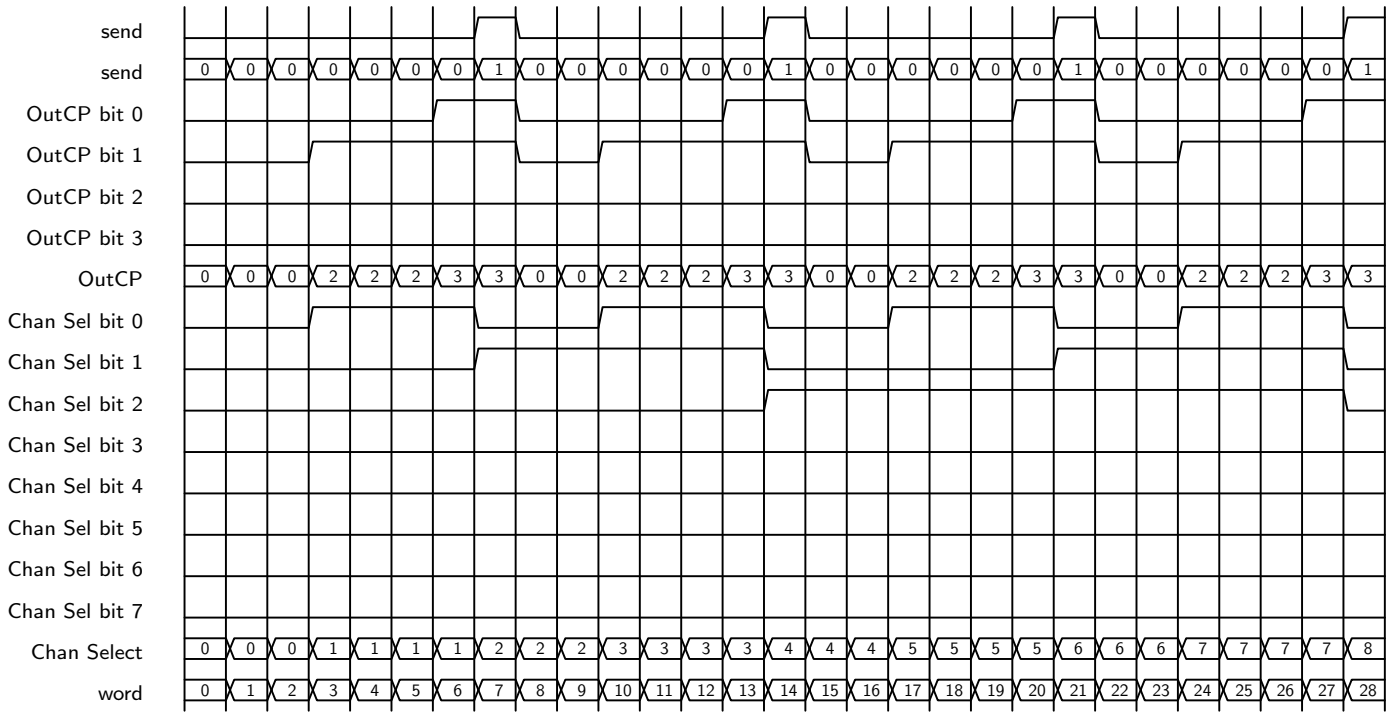
Defined in `init_pat_H2` as `patSkipLine`:



6.2.4 ADC pattern

The pattern-RAM of the ADC pattern generator is loaded with the following pattern in `init_adc_ch32`, executed in layer 4 in an endless loop. This has an intrinsic periodicity of 7 clocks and extends from words 0 to 112, reaching a channel-select value of 32 at the end.[7] The following diagram shows the transmission of the first 8 pixels (out of 128, the channel width of the Hawaii-2) via 4 `send` triggers

[7] The 4 reference ADCs of the board [2] are not used. The channel-select value of 32 has no associated followup OutCP edges and no side effect.




```
515 0 0 0 1 65535
        514 0 0 0 1 1
                513 0 0 0 1 1
                        512 0 0 0 112 1 0
                                511 0 0 0 0 0 1 0
                                511 0 1 0 0 0 1 0
                                511 0 2 0 0 0 1 0
                                511 0 3 0 2 0 1 1
                                511 0 4 0 2 0 1 1
                                511 0 5 0 2 0 1 1
                                511 0 6 0 3 0 1 1
                                511 0 7 1 3 0 1 2
                                511 0 8 0 0 0 1 2
                                511 0 9 0 0 0 1 2
                                ...
                                511 0 107 0 0 0 1 30
                                511 0 108 0 2 0 1 31
                                511 0 109 0 2 0 1 31
                                511 0 110 0 2 0 1 31
                                511 0 111 0 3 0 1 31
                                511 0 112 1 3 0 1 32
```

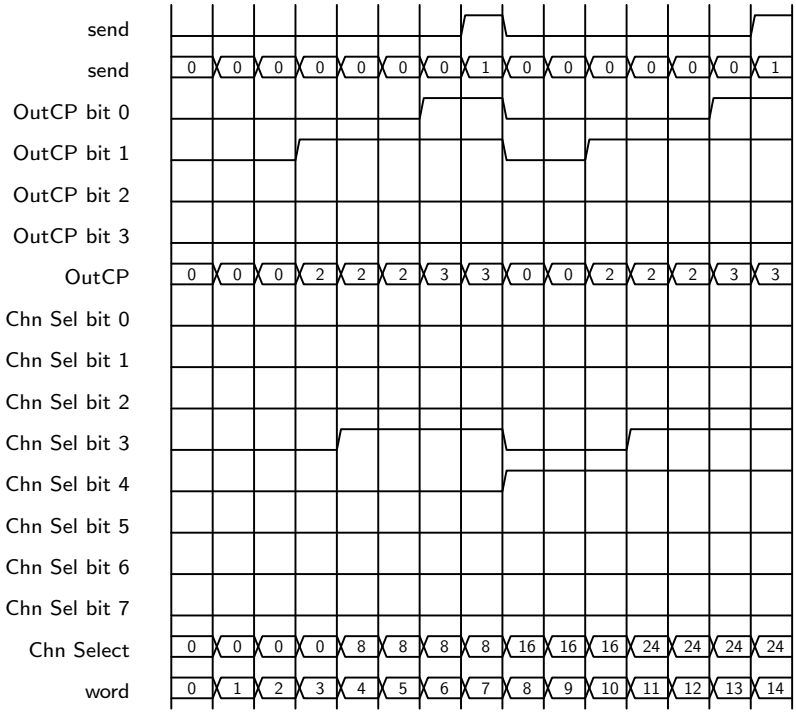
Not shown in the pattern are the 2 bits of the `In_CP` (always 0) and the 5 bits of the `Board-select` that have only the least-sigficant set (always 1). The two high-significant bits of the `OutCP` are always 0, which means that fiber does not transmit any relevant data; data of all 32 channels are fed into the other fiber.

Two consecutive ADC channels are packed into a 32-bit over the fiber such that we have only 16 send-triggers for the 32 channels.

The divisors value of the 512 command is 0, which means the period is close to 7.5 nsec, and the entire process of pushing the 32 data into the transmitter needs of the order of 112 × 7.575 ns = 0.85 μsec (equivalent to 1.2 MHz, and therefore faster than the 800 kHz of the ADC's AD7677 (normal mode, 250 ns acquisition time plus 1 μs converstion time) [9, p 52])

The file `init_adc_chan4` serves to read the Hawaii-2 in a 4-channel mode (1 channel per quadrant,

channels are 1024 pixels wide) and transports only the outputs 0, 8, 16, ... 24:



The modes with less-than-maximum number of channels of the detectors are initiated by setting the appropriate bits in the processors registers in the files `pttrns/*/registers_*`. The triggers in the patterns cause conversions in *all* ADCs; the data of the ignored channels are discarded by *not* feeding their outputs into the fibers, so they don't arrive at the computer. This means there is effectively no power-savings by using a lower number of channels here because all ADCs are still working in parallel at the usual rate (althoug some maybe in vain).

6.2.5 Idle ReadWoConv

If one sends the command to GEIRS to switch to the idle mode `ReadWoConv` word 2 of the layer-6 is overwritten to execute lines 12 to 14 of layer-5 (triggered in `roe_variables`, so the idle loop pattern now is a copy of the readout pattern with 'active' `scon0` edges:

```
718 0 2 12 14 4294967295
        717 0 12 21 23 1
                715 0 21 27 29 1
                        714 0 27 30 32 1
                                713 0 30 16 17 1 0 0
                                        712 0 16 7 13 1 60 1 0
                                                patFSync
                                713 0 31 36 46 1024 0 0
                                        712 0 36 21 27 1 60 1 0
                                                patVLS
                                        712 0 37 38 40 1 60 1 0
                                                patNoRes
                                        712 0 38 61 80 1 51 1 0
                                                patPIXnoConv
                                        712 0 39 90 109 127 51 1 0
                                                patPIXwConv
                                        712 0 40 119 138 1 51 1 0
                                                patConvOnly
                                        712 0 41 41 47 1 60 1 0
```

```
                                        patLSyncNoRes
                                712 0 42 28 30 1 60 1 0
                                        patRes
                                712 0 43 61 80 1 51 1 0
                                        patPIXnoConv
                                712 0 44 90 109 127 51 1 0
                                        patPIXwConv
                                712 0 45 119 138 1 51 1 0
                                        patConvOnly
                        714 0 28 34 35 1
                                713 0 34 12 16 1 0 0
                                712 0 12 177 178 0 999 3 0
                                        patItime
                                712 0 13 177 178 0 99 2 0
                                        patItime
                                712 0 14 177 178 0 4 1 0
                                        patItime
                                712 0 15 179 180 1 899 2 0
                                        patItimeRDen
                715 0 22 6 7 1
                        714 0 6 2 3 1
                                713 0 2 2 3 1 0 0
                                712 0 2 185 186 1 4 1 8
        717 0 13 23 24 1
                715 0 23 47 48 1
                        714 0 47 30 32 1
```

There is no change in the patterns of the control-FPGA, so currents from the detector/preamplifier to the ROE run with the same timing as in the readout cycle. The process on the ADC pattern generator (Section 6.2.4) remains active.[8]

The obvious disadvantage of that implementation is that also the time spent as a filler for the integration time in words 12, 13 and 14 of layer-1 remains the same. This means that in the usual `wait` choice for leaving the idle loop, reaching the associated break-point may need as long as a full readout-cycle.[9]

6.2.6 Idle Rlr

Switching the idle mode to reset-level-read (Rlr) changes the Program Number 2 to

```
718 0 2 2 3 4294967295
        717 0 2 7 9 1
                715 0 7 3 5 1
                        714 0 3 25 26 1
                                713 0 25 16 17 1 0 0
                                        712 0 16 7 13 1 60 1 0
                                                patFSync
                        714 0 4 9 11 1024
                                713 0 9 31 33 1 0 0
                                        712 0 31 21 27 1 60 1 0
                                                patVLS
                                        712 0 32 28 30 1 60 1 0
                                                patRes
                                713 0 10 9 12 1 0 0
                                        712 0 9 61 80 1 51 1 0
                                                patPIXnoConv
                                        712 0 10 61 80 127 51 1 0
                                                patPIXnoConv
                                        712 0 11 148 167 1 51 1 0
                                                patNoPIXnoConv
```

[8] GEIRS does not stop and restart it for each readout cycle, but only once within `roe_init` at startup time.

[9] Operators tend to get angry then because they realize that telescope time is wasted here.

```
715 0 8 6 7 1
        714 0 6 2 3 1
                713 0 2 2 3 1 0 0
                        712 0 2 185 186 1 4 1 8
```

This is for each line of the detector a reset followed by 128 advances along the horizontal direction, with no conversions nor any wait to recognize the integration time.

6.3 Hawaii-2RG, Hawaii-4RG

6.3.1 RAM Layer

In the pattern-layer of the detector-FPGA, the 711 command to upload one word (out of up to 1024) has the format [4]

```
711 0 address 0x0to15 0x16to31 0x32to47 0x48to63
```

where the first parameter (the zero) is not used, and 4 groups of 16 bits each carry a load of 64 bits, starting with the least significant bit in the leftmost group. Assignment of functionality to the 64 bits of the 711 parameters:

signal	bufdis	hvw	reseten	readen	hclk	/lsync	vclk	/fsync	stcon1	stcon1
bit	23	22	21	20	19	18	17	16	1	0

The signal in parentheses, `stcon1`, is a leftover from older ROEs (?) and can effectively be ignored for all cases. So instead of patterns `0x3` and `0x1` listed below one will effectively only meet `0x1` and `0x0` in bits 0–15.

The bits 16 to 23 are usually duplicated for each of the chips in the mosaic into bits 24–31, 32–39 and 40–47, and this can be done without harm for few-chip instruments because the superfluous outputs will end nowhere.[10] So instead of

```
711 0 address 0x1 0x95 0x0 0x0
```

for example one will typically write

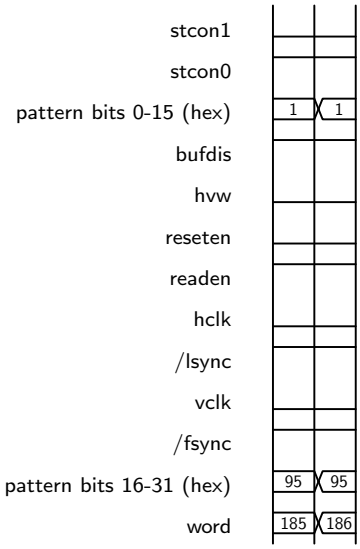
```
711 0 address 0x1 0x9595 0x9595 0x0
```

in the pattern files to allow code sharing between the AIP 4-chip setup, the CARMENES 2-chip setup and the other 1-chip setups.

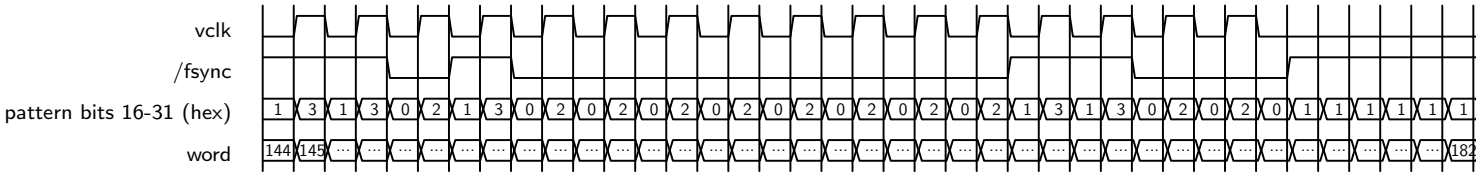
After booting (starting GEIRS), the ROE registers contain the following bits in the detector-FPGA RAM layer. Words are enumerated from 0 to 1023, the maximum capacity of the FPGA, but less than 200 are actually defined at that point in time.

[10] . . . assuming appropriate bus terminations. This allows to share pattern code among instruments with 1, 2 or 4 detector chips. This does not occur for LN patterns because MPIA has build only that one instrument with the Hawaii 2 detector and that is equipped only with one of these.

Defined in `init_pat_H2RG` with the name `patSetup1`:

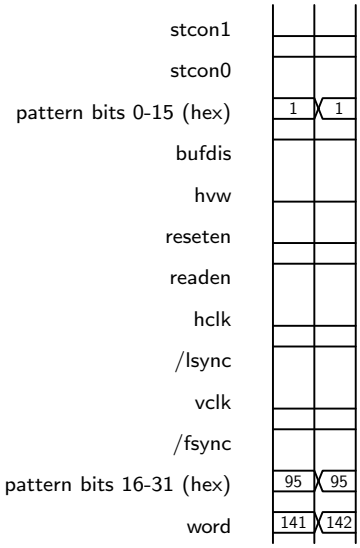


Defined in `init_pat_H2RG` with the name `patSetup2`, assuming that the CSB has been used to switch to the serial interface. Only VCLK and FSYNCB are relevant then. 16 edges VCLK are used to trigger the transmission of 16 bits:

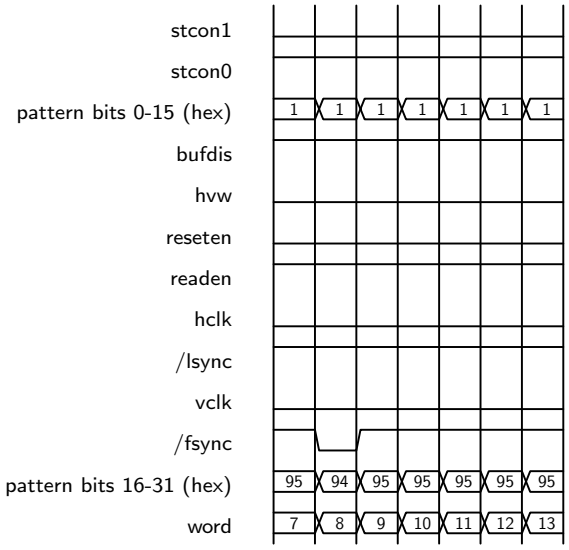


PDCK = PDCKWM = 0) This effectively sets IFCTRL = DOEN = 1, PDCK = PDCKWM = 0 (data bits 0000 0000 1100) in the OptionsReg (address 1101) by including `init_pat_H2RG_ser`. The `init_pat_H4RG_ser` does the same for the Hawaii-4RG OptionsReg (address 1111).

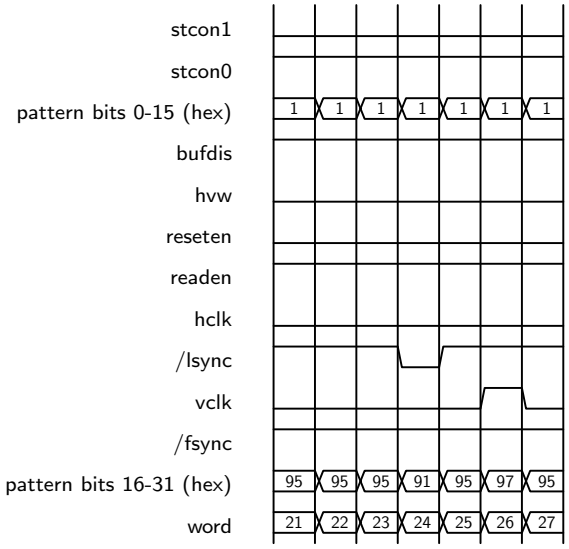
Defined in `init_pat_H2RG` with the name `patNone`:



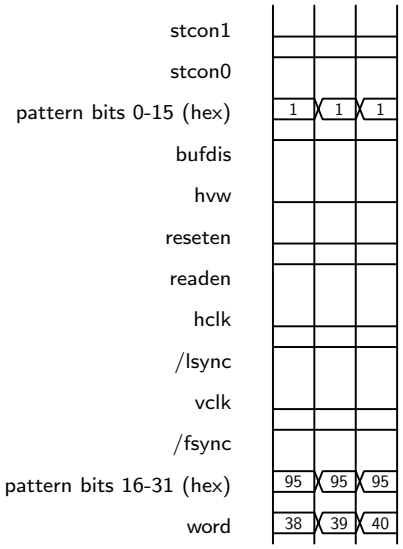
Defined in `init_pat_H2RG` with the name `patFSync`:



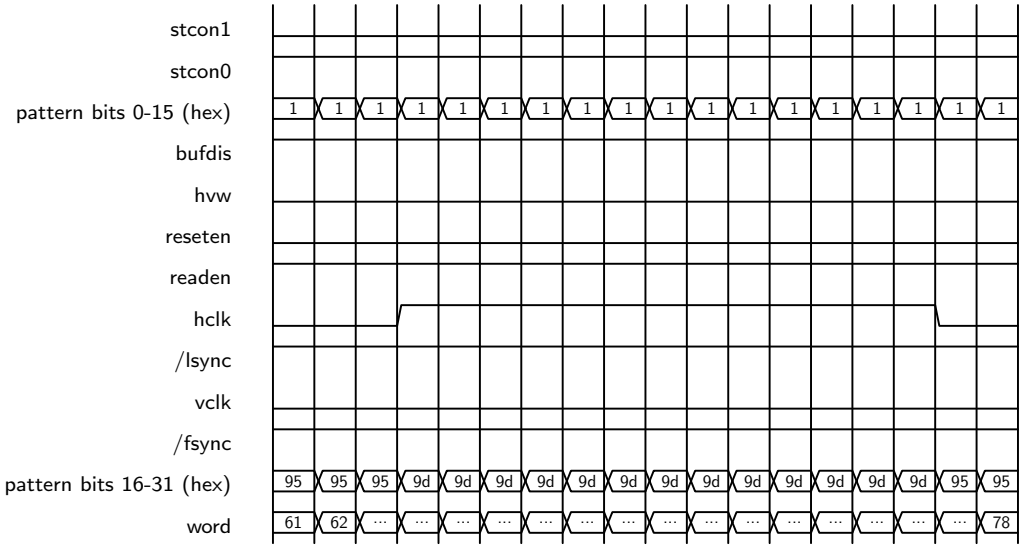
Defined in `init_pat_H2RG` with the name `patVLS`:



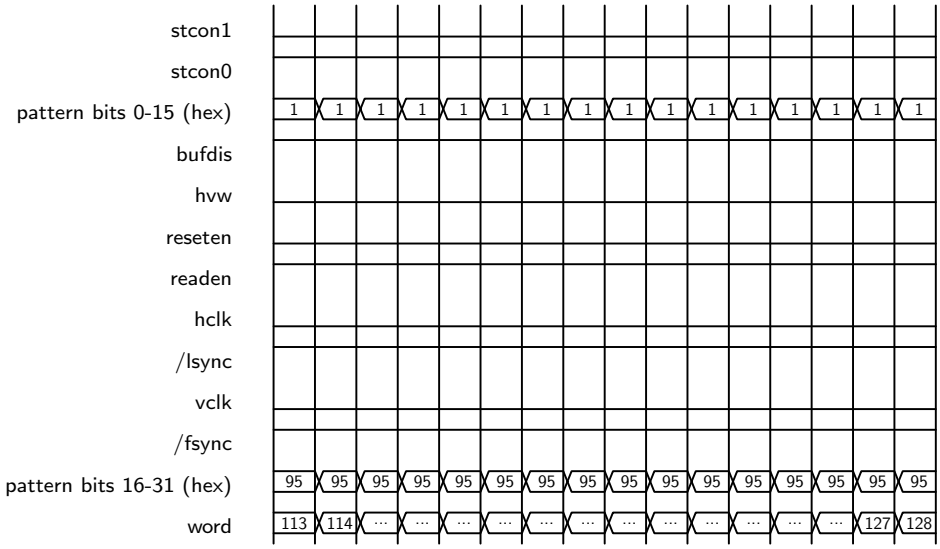
Defined in `init_pat_H2RG` with the name `patNoRes`:



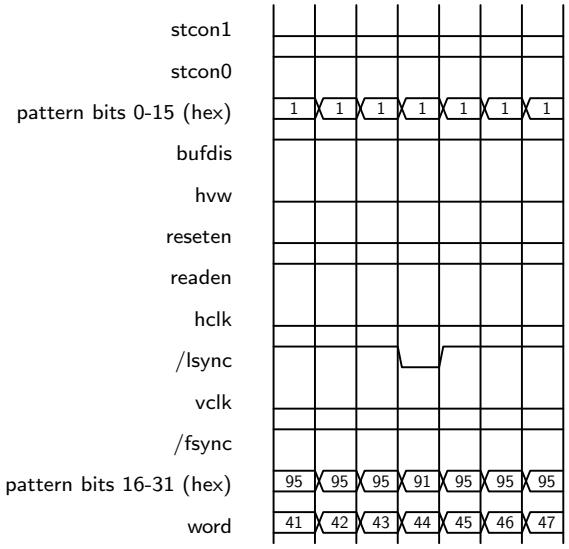
Defined in `init_pat_H2RG` with the name `patPIXnoConv`, including `pat_ems_PIXnoConv_H2RG`:



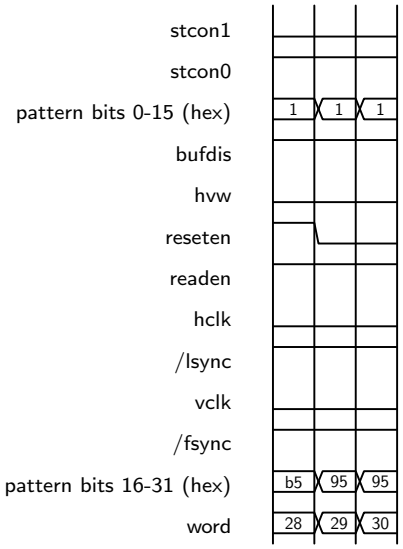
Defined in `init_pat_H2RG` with the name `patNoPIXnoConv`, including `pat_ems_NoPIXnoConv_H2RG`:



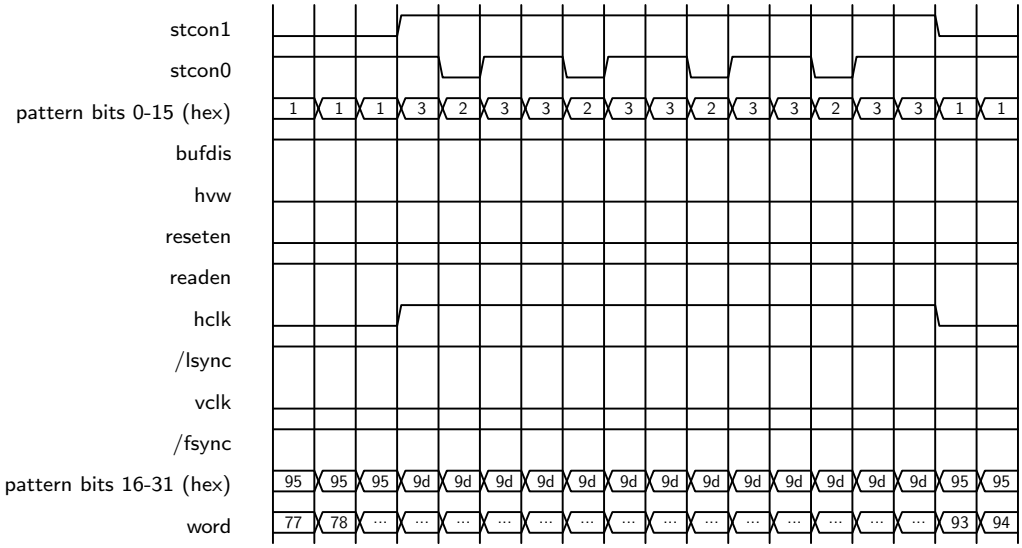
Defined in `init_pat_H2RG` with the name `patLSyncNoRes`:



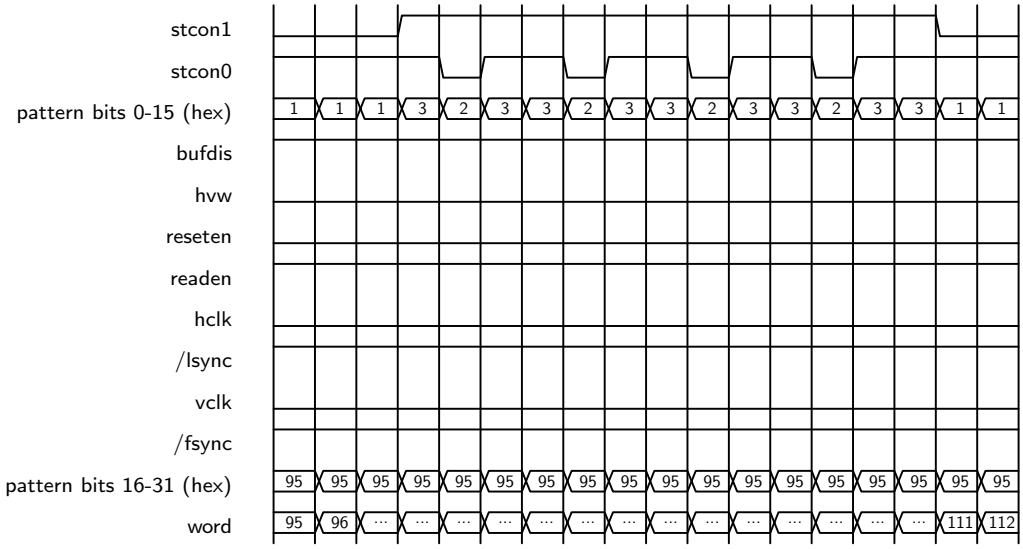
Defined in `init_pat_H2RG` with the name `patRes`:



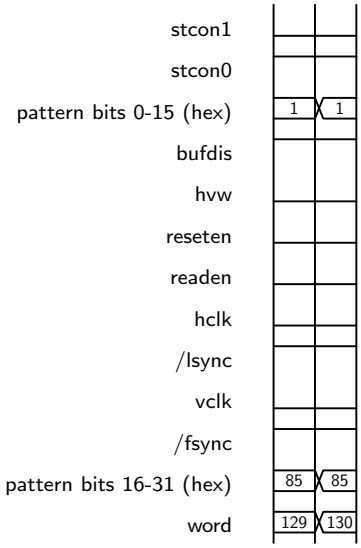
Defined in `init_pat_H2RG` with the name `patPIXwConv`, and including `pat_ems_PIXwConv_H2RG` where `ems=4` [10, Fig. 7]:



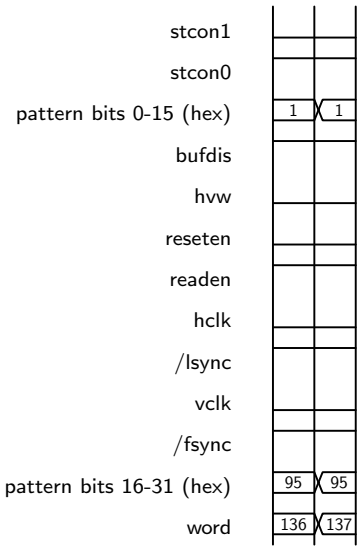
Defined in `init_pat_H2RG` with the name `patConvOnly`, and including `pat_ems_ConvOnly_H2RG` where `ems=4`:



Defined in `init_pat_H2RG` with the name `patItime`:



Defined in `init_pat_H2RG` with the name `patItimeRDen`:



6.3.2 Initial Pattern

Note that there are holes in the words explained above, where other patterns are defined/uploaded but not yet relevant after boot time. They exist in preparation for subwindows, electronic multi-sampling or other parameters that may be relevant when the operator starts to use the detector. Note also that there are some redundant pieces here.[11]

There are 7 layer-6 control-FPGA programs defined at that time.

1. Number 0 keeps all signals frozen (at 616 nsec clock) and enabled as in `patSetup2`, but may toggle from an infinite loop to a single execution in layer 3:

```
718 0 0 0 1 1
        717 0 0 0 3 1
                715 0 0 0 1 65535
                        714 0 0 0 1 1
                                713 0 0 0 1 1 0 0
                                        712 0 0 185 186 1 60 1 4088
                                                patSetup1
                715 0 1 1 2 1
                        714 0 1 1 2 1
                                713 0 1 1 2 1 0 0
                                        712 0 1 144 183 1 60 1 4088
                                                patSetup2
                715 0 2 0 1 65535
```

2. Number 1 is a single action that disables the `stcon` by masking a byte (selected with a specific bit in the last parameter, 4080) (51 nsec clock):

```
718 0 1 4 5 1
        717 0 4 3 4 1
                715 0 3 7 8 1
                        714 0 7 3 4 1
                                713 0 3 3 4 1 0 0
                                        712 0 3 141 142 1 4 1 4080
                                                patNone
```

[11] e.g. because patNone and patSetup2 are the same, lay1ConvOn and lay1Setup2 are actually the same pattern.

3. Number 2 is an infinite loop, containing two sequential subroutines in layer 4. The first subroutine is essentially a frame sync followed by 2048 advances along the slow direction (where 2048 is the number of pixels along each of the 32 channels of the Hawaii-2RG and is replaced by 4096 if a Hawaii-4RG is present), and the second subroutine sets the bit for the break point ID 1 (represented by the final 8 in the 712 command). Each advance along the slow direction contains $2 + 63 = 65$ horizontal clocks without triggering ADC's—where 64 is the width of a channel for the Hawaii-2RG and also the envisioned Hawaii-4RG—a line sync and reset, and again 128 horizontal clocks, altogether a LIR pattern without conversion:

```
718 0 2 3 4 4294967295
        717 0 3 9 11 1
                715 0 9 5 6 1
                        714 0 5 28 30 1
                                713 0 28 15 16 1 0 0
                                        712 0 15 7 13 1 60 1 0
                                                patFSync
                                713 0 29 55 65 2048 0 0
                                        712 0 55 21 27 1 60 1 0
                                                patVLS
                                        712 0 56 38 40 1 60 1 0
                                                patNoRes
                                        712 0 57 61 76 2 52 1 0
                                                patPIXnoConv
                                        712 0 58 61 76 63 52 1 0
                                                patPIXnoConv
                                        712 0 59 113 128 1 52 1 0
                                                patNoPIXnoConv
                                        712 0 60 41 47 1 60 1 0
                                                patLSyncNoRes
                                        712 0 61 28 30 1 60 1 0
                                                patRes
                                        712 0 62 61 76 2 52 1 0
                                                patPIXnoConv
                                        712 0 63 61 76 63 52 1 0
                                                patPIXnoConv
                                        712 0 64 113 128 1 52 1 0
                                                patNoPIXnoConv
                715 0 10 6 7 1
                        714 0 6 2 3 1
                                713 0 2 2 3 1 0 0
                                        712 0 2 141 142 1 4 1 8
                                                patNone
```

   This implements the initial default of the idle loop (`wait` in `Lir` mode).

4. Number 3 enables all outputs with a 4088 value of the last parameter, so it would revert the status left by Number 1:

```
718 0 3 5 6 1
        717 0 5 4 5 1
                715 0 4 8 9 1
                        714 0 8 4 5 1
                                713 0 4 4 5 1 0 0
                                        712 0 4 141 142 1 4 1 4088
                                                patNone
```

   A subsequent `read` will use the 764 command with the break point in Program 2 to execute Programs 3 to last pattern in an endless loop (`764 0 3 7 65535 1`).

5. Number 4 is like Number 2, but (i) the conversion (ADC's triggers) are enabled in the subroutine that walks through the pixels along the slow direction, and (ii) there is a filler

with `patItime` patterns with by three actions in layer-1 with zero loop counts, effectively zero additional integration time: (iii) there is a second walk through the pixels along the slow direction.[12]

```
718 0 4 12 14 1
        717 0 12 21 23 1
                715 0 21 27 29 1
                        714 0 27 30 32 1
                                713 0 30 15 16 1 0 0
                                        712 0 15 7 13 1 60 1 0
                                                patFSync
                                713 0 31 35 45 2048 0 0
                                        712 0 35 21 27 1 60 1 0
                                                patVLS
                                        712 0 36 38 40 1 60 1 0
                                                patNoRes
                                        712 0 37 61 76 2 52 1 0
                                                patPIXnoConv
                                        712 0 38 77 94 63 52 1 0
                                                patPIXwConv
                                        712 0 39 95 112 1 52 1 0
                                                patConvOnly
                                        712 0 40 41 47 1 60 1 0
                                                patLSyncNoRes
                                        712 0 41 28 30 1 60 1 0
                                                patRes
                                        712 0 42 61 76 2 52 1 0
                                                patPIXnoConv
                                        712 0 43 77 94 63 52 1 0
                                                patPIXwConv
                                        712 0 44 95 112 1 52 1 0
                                                patConvOnly
                        714 0 28 34 35 1
                                713 0 34 12 15 1 0 0
                                        712 0 12 129 130 0 999 3 0
                                                patItime
                                        712 0 13 129 130 0 99 2 0
                                                patItime
                                        712 0 14 129 130 0 4 1 0
                                                patItime
                715 0 22 6 7 1
                        714 0 6 2 3 1
                                713 0 2 2 3 1 0 0
                                        712 0 2 141 142 1 4 1 8
                                                patNone
        717 0 13 23 24 1
                715 0 23 47 48 1
                        714 0 47 30 32 1
```

The readout of the 64 pixels in the horizontal direction of a channel is split into 2× `patPIXnoConv`, 63× `patPIXwConv` and 1×`patConvOnly`, so the pixel 'address' stands at the 'end' of the line at the start of the (optional, residual) integration time. (The readout of the 64 pixels requires 66 falling HCLK edges [11, Fig. 5-4]. Isn't one missing, or is the last, overclocking one only needed to generate the optional LINECHK signal?).

After splitting the residual integration time into three different intervals (at 10.1 sec, 1.01 msec, 50.5 nsec with three different prescaler choices) the main program will later on overload the layer-1 commands (words 12, 13 and 14) at the three actions to realize any integration time that is longer than the minimum integration time.

[12]The second ramp of the LIR pattern is basically enforced here, and manipulation of the loop count in the first ramp may increase the cycle repetition to larger values...

6. Number 5 is the same as Number 1:

   ```
   718 0 5 4 5 1
   ```

   This means after exit of the read-loop of Number 4 the idle loop is started.

7. Number 6 is the same infinite loop as Number 2:

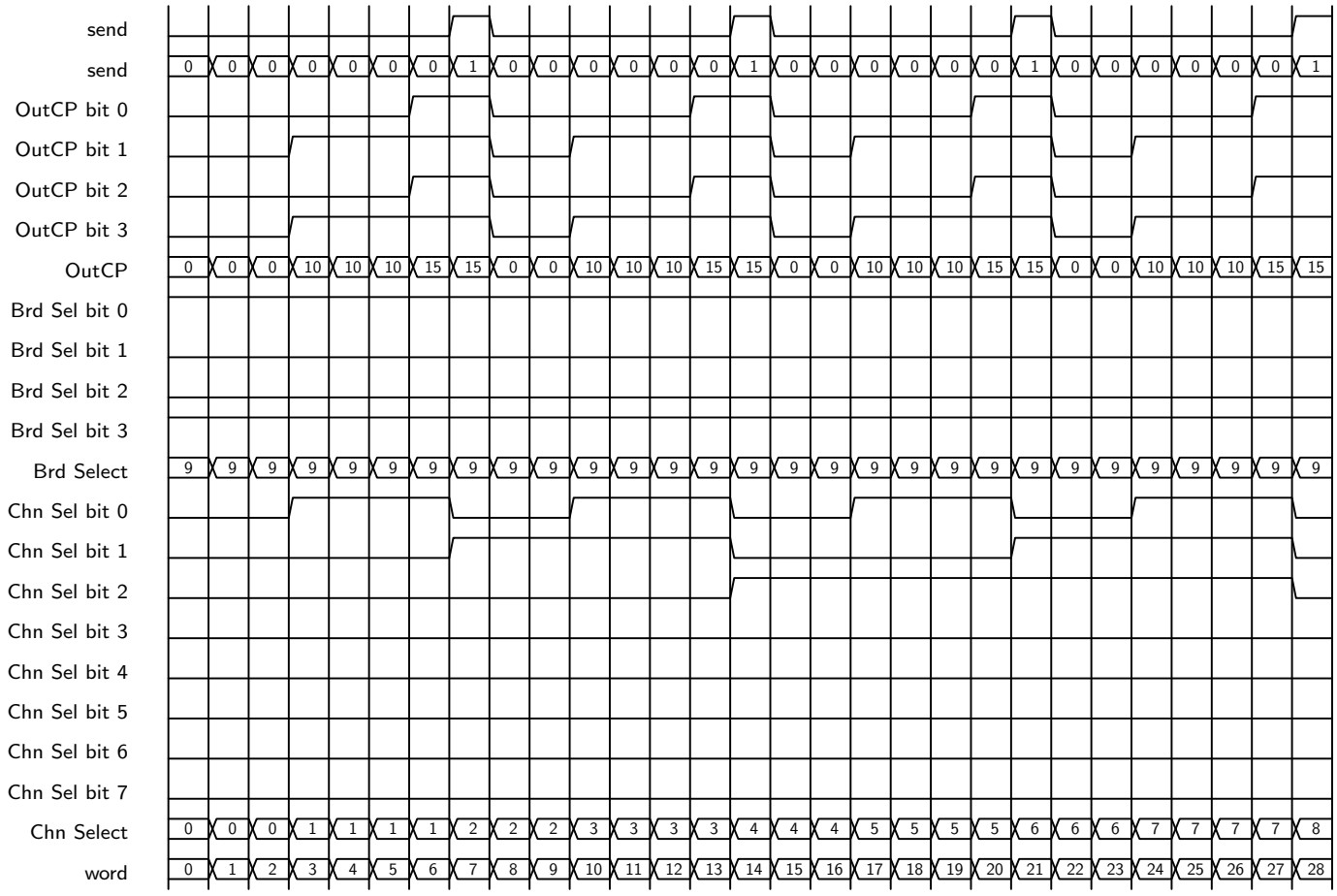
   ```
   718 0 6 3 4 4294967295
   ```

The commands in `init_H2RG` reset some registers to default values, start to run Program 0, switch the CSB, end the inherent infinite loop at layer 4 such effectively `patSetup2` is executed, switch again the CSB and end the loop at layer 6, such that Program 1 is executed and then Program 2 (the idle loop).

6.3.3 ADC pattern

For a single Hawaii-2RG the ADC pattern is exactly the same as in section 6.2.4.

For the CARMENES case each of the two chips is served by its dedicated ADC36 board, and the serialization of the two data streams runs in parallel on these two boards. The board-select bit pattern is $1001_2 = 9$ to mark two boards. The following diagram shows the transmission of the first 8 pixels of each chip via 4 `send` triggers:




```
515 0 0 0 1 65535
        514 0 0 0 1 1
                513 0 0 0 1 1
                                511 0 0 0 0 0 9 0
                                511 0 1 0 0 0 9 0
                                511 0 2 0 0 0 9 0
                                511 0 3 0 10 0 9 1
                                511 0 4 0 10 0 9 1
                                511 0 5 0 10 0 9 1
                                511 0 6 0 15 0 9 1
                                511 0 7 1 15 0 9 2
                                511 0 8 0 0 0 9 2
                                511 0 9 0 0 0 9 2
                                511 0 10 0 10 0 9 3
                                511 0 11 0 10 0 9 3
                                511 0 12 0 10 0 9 3
                                511 0 13 0 15 0 9 3
                                511 0 14 1 15 0 9 4
                                511 0 15 0 0 0 9 4
                                511 0 16 0 0 0 9 4
```

```
                        511 0 17 0 10 0 9 5
                        511 0 18 0 10 0 9 5
                        511 0 19 0 10 0 9 5
                        ..
                        511 0 107 0 0 0 9 30
                        511 0 108 0 10 0 9 31
                        511 0 109 0 10 0 9 31
                        511 0 110 0 10 0 9 31
                        511 0 111 0 15 0 9 31
                        511 0 112 1 15 0 9 32
```

The envisioned scheme for the PANIC Hawaii-4RG is again the same, because 32 channels are served by one board, the other 32 by the other board. The ROE wiring selects that outputs 0–31 are moved to one board, outputs 32–63 into the other.

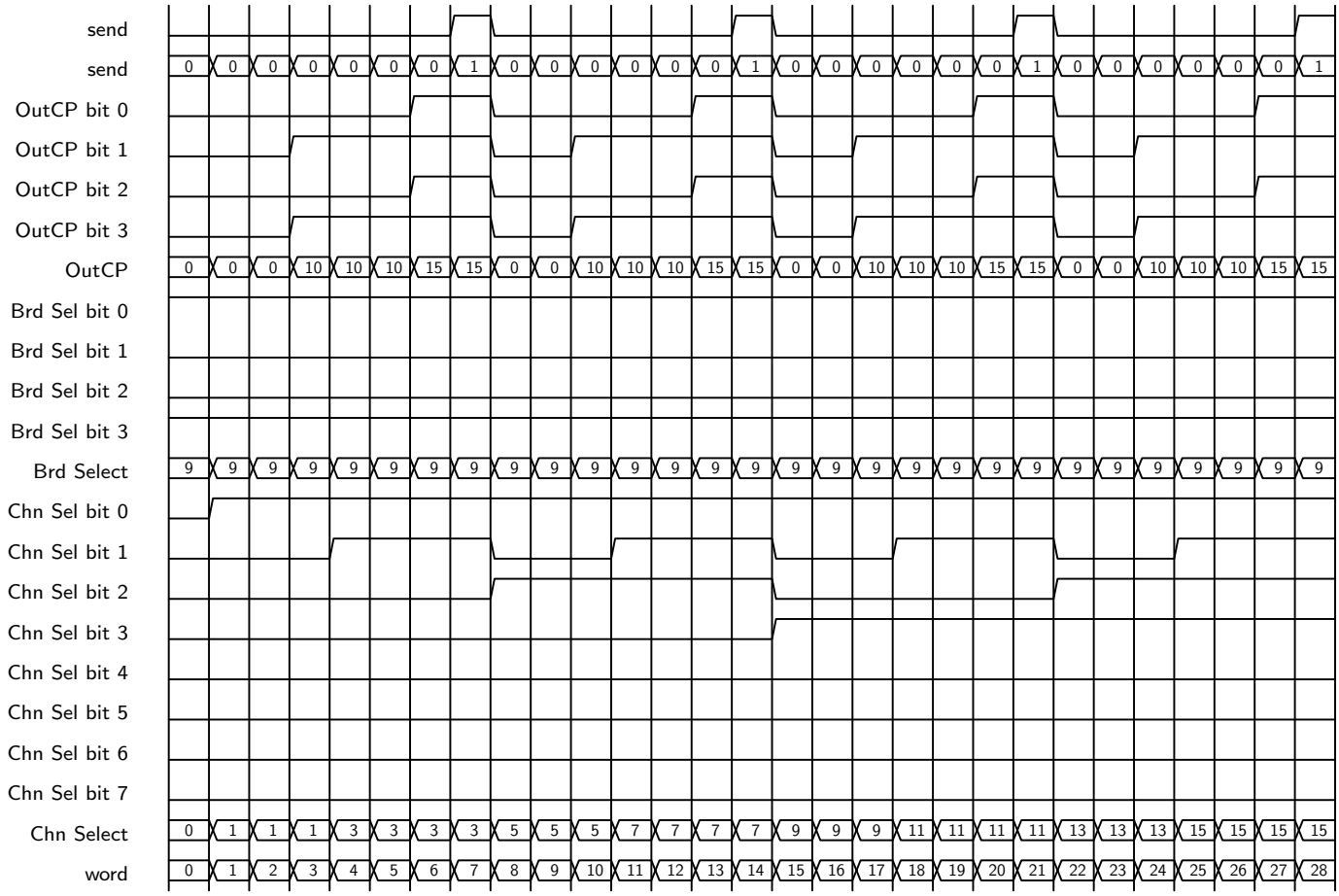
For the AIP configuration, two chips share the same fiber channel, so there is a pattern where the first and the last board are selected (Board-select $1001_2 = 9$), followed by a second pattern where the two middle boards are selected (Board-select $0110_2 = 6$):

```
515 0 0 0 1 65535
       514 0 0 0 1 1
              513 0 0 0 1 1
                        511 0 0 0 0 0 9 0
                        511 0 1 0 0 0 9 0
                        511 0 2 0 0 0 9 0
                        511 0 3 0 10 0 9 1
                        511 0 4 0 10 0 9 1
                        511 0 5 0 10 0 9 1
                        511 0 6 0 15 0 9 1
                        511 0 7 1 15 0 9 2
                        511 0 8 0 0 0 9 2
                        511 0 9 0 0 0 9 2
                        511 0 10 0 10 0 9 3
                        511 0 11 0 10 0 9 3
                        511 0 12 0 10 0 9 3
                        511 0 13 0 15 0 9 3
                        511 0 14 1 15 0 9 4
                        511 0 15 0 0 0 9 4
                        511 0 16 0 0 0 9 4
                        511 0 17 0 10 0 9 5
                        511 0 18 0 10 0 9 5
                        511 0 19 0 10 0 9 5
                        ..
                        511 0 107 0 0 0 9 30
                        511 0 108 0 10 0 9 31
                        511 0 109 0 10 0 9 31
                        511 0 110 0 10 0 9 31
                        511 0 111 0 15 0 9 31
                        511 0 112 1 15 0 9 32
                        511 0 113 0 0 0 6 0
                        511 0 114 0 0 0 6 0
                        511 0 115 0 0 0 6 0
                        511 0 116 0 10 0 6 1
                        511 0 117 0 10 0 6 1
                        511 0 118 0 10 0 6 1
                        511 0 119 0 15 0 6 1
                        511 0 120 1 15 0 6 2
                        511 0 121 0 0 0 6 2
                        511 0 122 0 0 0 6 2
                        511 0 123 0 10 0 6 3
                        511 0 124 0 10 0 6 3
                        ..
                        511 0 219 0 0 0 6 30
                        511 0 220 0 0 0 6 30
                        511 0 221 0 10 0 6 31
                        511 0 222 0 10 0 6 31
                        511 0 223 0 10 0 6 31
```

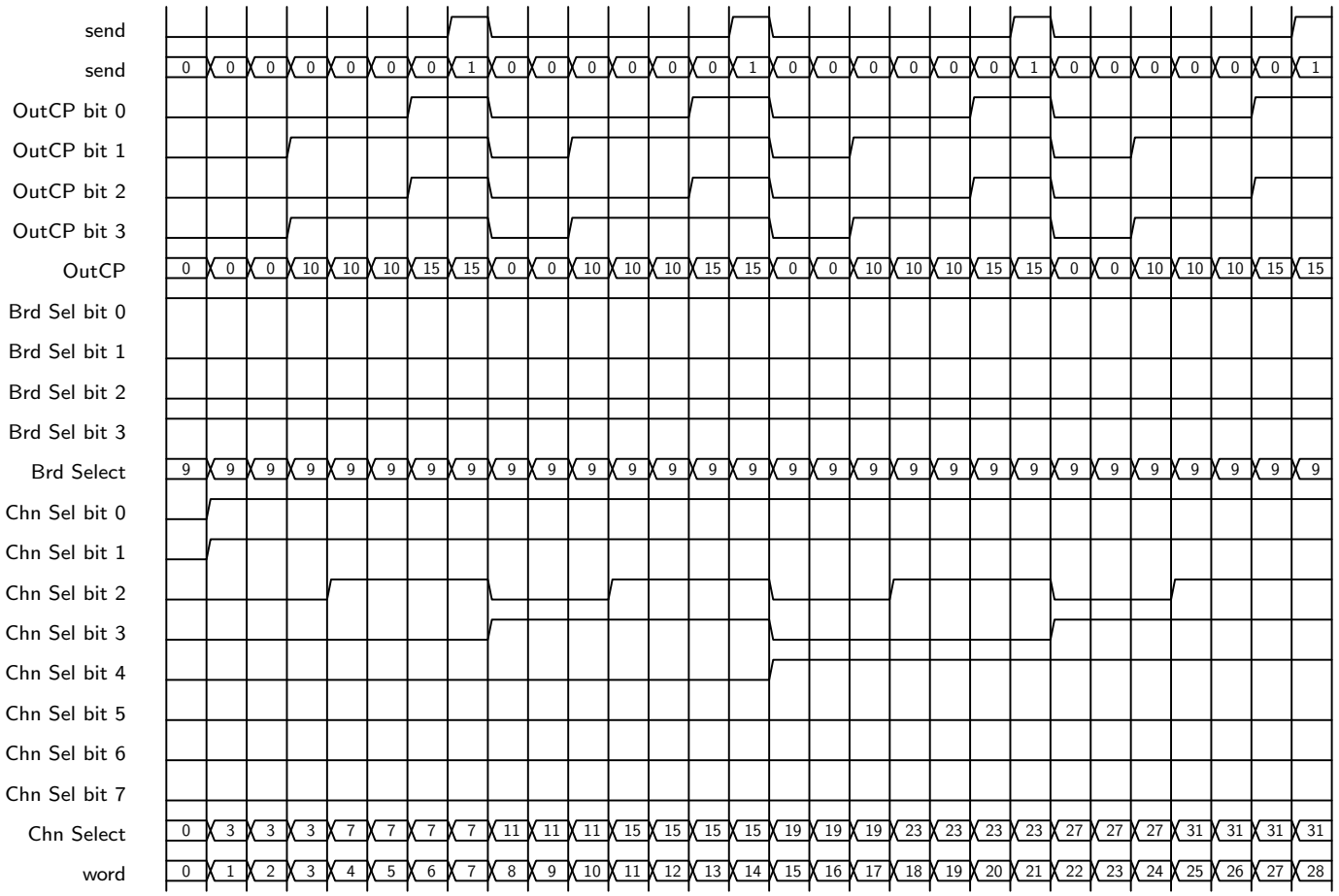
```
511 0 224 0 15 0 6 31
511 0 225 1 15 0 6 32
```

Since this serialization needs roughly twice as long as the one of Section 6.2.4, the transfer of the 4×32 pixels is equivalent to a pixel clock limited to 590 kHz such that the speed limit is set by the serialization, not by the 800-kHz ADC conversion.

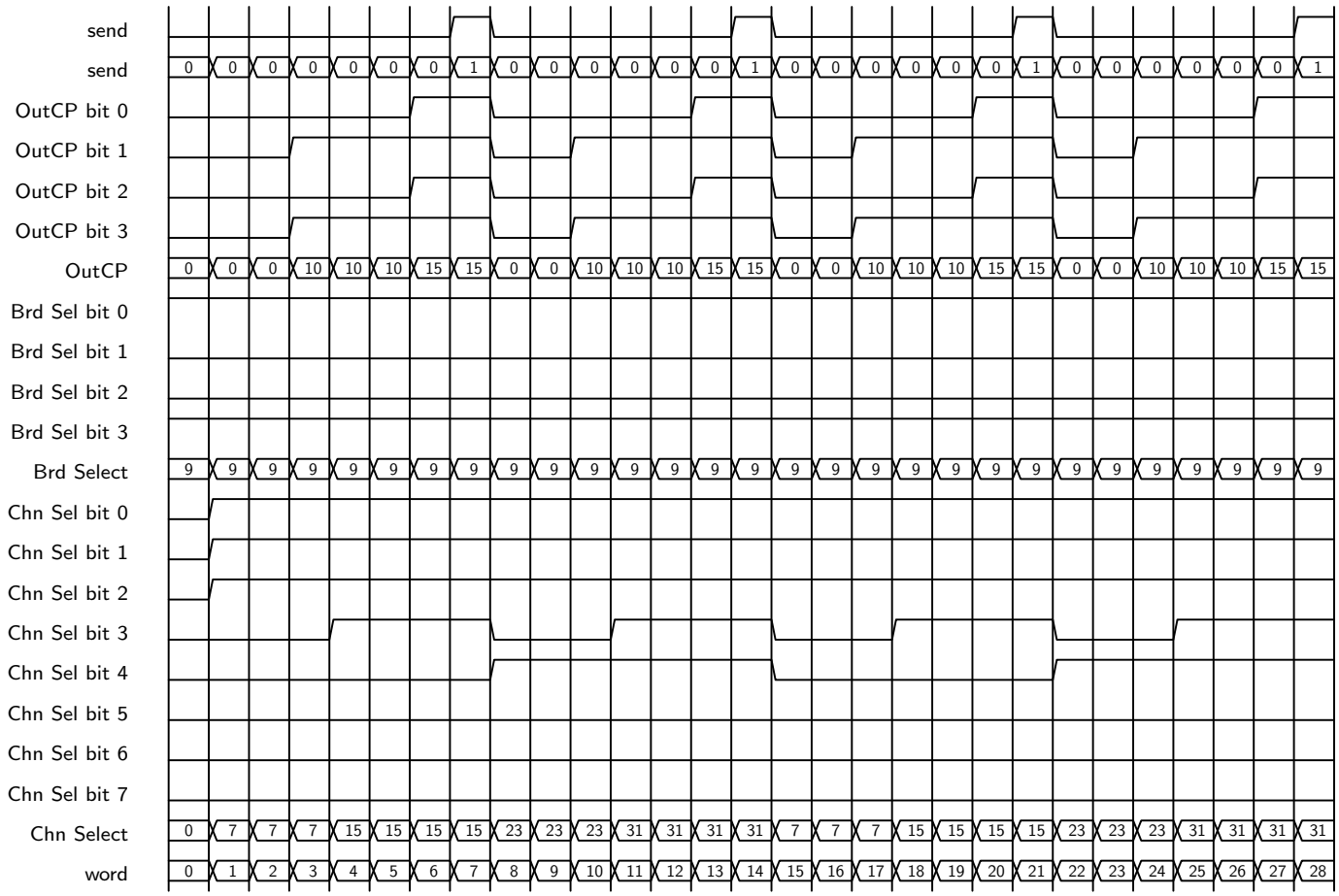
For the Hawaii-4RG in a 32-channel mode we still need both boards to read the entire detector, because in 32-output mode the chip pushes data through the odd-numbered outputs 1, 3, 5,... 63 and the MPIA wiring connects 0–31 with one board, 32–63 with the other. Only 16 of the ADC's of each board are streamed into the fibers. The associated ADC timing in the file `init_adc_chan16` starts like this:



The file `init_adc_chan8` serves to read the Hawaii-4RG in a 16-channel mode and transporting only the outputs 3, 7, 11, ... 63:



`init_adc_chan4` can be used to read the Hawaii-2RG in a 4-channel mode handling only the outputs 7, 15, 23 and 31:



Here a cycle is 2 triggers sending 4 pixel values.

Then there is `init_adc_chan4.panic` which is used to read the Hawaii-4RG in a 4-channel mode handling only the outputs 15, 31, 47 and 63, which from the two individual ADC-boards of view means the outputs 15 and 31 are send with a single trigger through the fibers. (Timing not shown here.)

The base clock for the ADC process is slowed down from the 7.5 ns by factors of 2 and 4 for these slowlier cases, and their channel select bits moved away from the OutCP bit triggers by one clock.

6.3.4 Idle ReadWoConv

If one sends the command to GEIRS to switch to the idle mode `ReadWoConv` the patterns that clock through detector chip are replaced by patterns in Program 2 that clock through the detector *and* trigger ADC conversions, just as for the Hawaii-2 case. Assuming the LIR main read mode, the pattern layers then become

```
718 0 1 4 5 1
        717 0 4 3 4 1
                715 0 3 7 8 1
                        714 0 7 3 4 1
                                713 0 3 3 4 1 0 0
                                        712 0 3 145 146 1 4 1 4080
                                                patNone
718 0 2 12 14 4294967295
        717 0 12 21 23 1
```

```
                715 0 21 27 29 1
                        714 0 27 30 32 1
                                713 0 30 15 16 1 0 0
                                        712 0 15 7 13 1 60 1 0
                                                patFSync
                                713 0 31 35 45 2048 0 0
                                        712 0 35 21 27 1 60 1 0
                                                patVLS
                                        712 0 36 38 40 1 60 1 0
                                                patNoRes
                                        712 0 37 61 78 2 52 1 0
                                                patPIXnoConv
                                        712 0 38 79 96 63 52 1 0
                                                patPIXwConv
                                        712 0 39 97 114 1 52 1 0
                                                patConvOnly
                                        712 0 40 41 47 1 60 1 0
                                                patLSyncNoRes
                                        712 0 41 28 30 1 60 1 0
                                                patRes
                                        712 0 42 61 78 2 52 1 0
                                                patPIXnoConv
                                        712 0 43 79 96 63 52 1 0
                                                patPIXwConv
                                        712 0 44 97 114 1 52 1 0
                                                patConvOnly
                                714 0 28 34 35 1
                                        713 0 34 12 15 1 0 0
                                                712 0 12 133 134 0 999 3 0
                                                        patItime
                                                712 0 13 133 134 0 99 2 0
                                                        patItime
                                                712 0 14 133 134 0 4 1 0
                                                        patItime
                715 0 22 6 7 1
                        714 0 6 2 3 1
                                713 0 2 2 3 1 0 0
                                        712 0 2 145 146 1 4 1 8
                                                patNone
        717 0 13 23 24 1
                715 0 23 47 48 1
                        714 0 47 30 32 1
718 0 3 5 6 1
        717 0 5 4 5 1
                715 0 4 8 9 1
                        714 0 8 4 5 1
                                713 0 4 4 5 1 0 0
                                        712 0 4 145 146 1 4 1 4088
718 0 4 12 14 1
718 0 5 4 5 1
718 0 6 12 14 4294967295
```

Note that Program 0, 5 and 6 are still present in the layers, but GEIRS sends a `716 0 1 5 65535` to the ROE such that only Programs 1 to 4 are executed. Program 1 still disables the output of the `stcon` signals.

6.4 Pattern Examples

The basic coordinate system on each detector is a horizontal direction ranging in each channel from 1 to 64 (Hawaii-2RG, Hawaii-4RG) or 1 to 128 (Hawaii-2), and an orthogonal vertical direction ranging from 1 to 2048 (Hawaii-2RG), 1 to 4096 (Hawaii-4RG), or 1 to 1024 (Hawaii-2); we assume readout patterns with 32 channels for Hawaii-2 and Hawaii-2RG, and readout patterns with 64

channels for Hawaii-4RG .[13]. The horizontal direction is also called the fast (f) direction, because the pixel pointer of reading the detector is along that direction changes fastest. The vertical direction is also called the slow (s) direction because that coordinate changes slowlier in time.

6.4.1 LIR

Figure 6 shows the scanning of director with the LIR pattern over the time coordinate [7]. Green is a pixel read, red is a line reset. We address the first row ($s = 1$) and read each channel along its fast direction ($f = 1 \ldots 74$). This happens for all 32 channels at the same time, and only one of these channels is pictures. After ($s = 1, f = 64$) has been read, the first row is completed; a reset over the entire line follows, and once again the first row ($s = 1$) is read over all pixles ($f = 1 \ldots 64$). This triple of read-reset-read is repeated for all coordinates, incrementing the slow coordinate until we have reached the last row and last pixel, which is $s = 2048$ for the Hawaii-2RG case. The duration of one green block is the pixel clock, 10 μs for the standard 100 kHz readout speed. Ignoring the small overhead of resets and sync operations, the pattern needs $64 \times 2 \times 2048 \times 10\mu$ s $= 2.53$ s for this standard scan of the Hawaii-2RG.

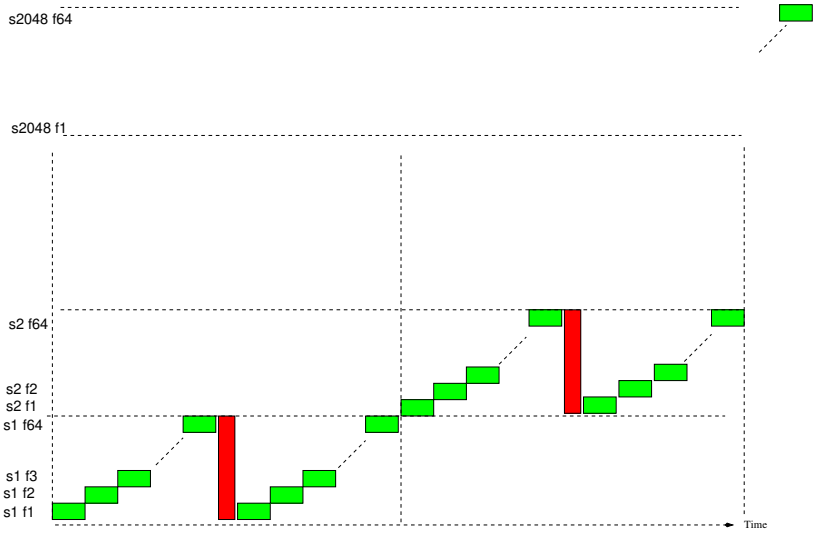


Figure 6: The LIR (line integrated read) pattern. Only the first read-reset-read scan is shown. Line sync, frame syncs and other administrative details are not shown.

The scans of Figure 6 are repeated one more times than the number of images requested, twice for 1 image, thrize for 2 images and so on, as illustrated in Figure 7. The gap time between the two vertical lines is filled with the `patItime` pattern tuned such that the distance between two correlated reads of a pixels becomes the integration time.

6.4.2 SRR

Figure 8 illustrates the first three ramps in the sample-up the ramp (SRR) mode. The first ramp uses a single line reset which is followed by a single read of each pixel in the line. Again we assume

[13]These are the two directions from the pattern perspective; The FITS or display directions may differ because GEIRS has two parameters that allow to flip and/or rotate these coordinate systems.

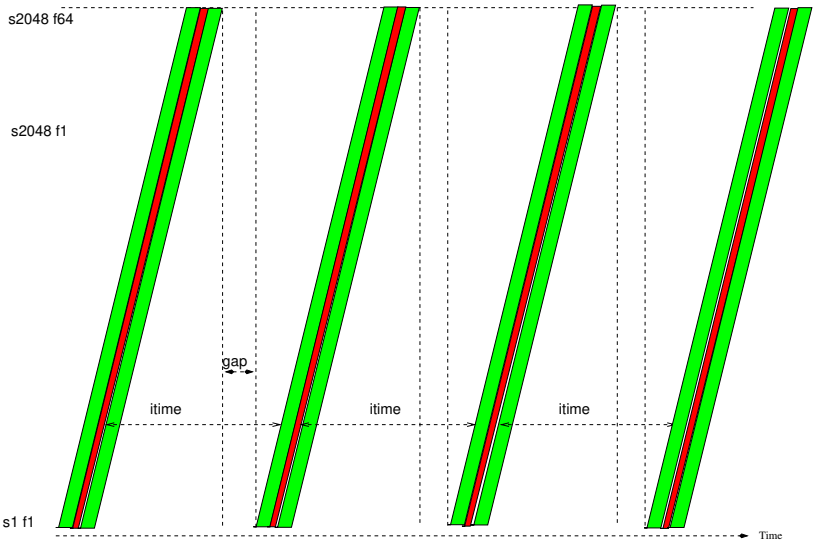


Figure 7: The LIR (line integrated read) pattern on a coarse time scale, assuming 4 ramps that yield 3 correlated double reads. Each of the green-red-green patterns summarizes one of the scans of Figure 6.

32 readout channels run in parallel, such that we need 64 reads in each channel for a Hawaii-2RG. The following ramps have no line resets.

Ignoring the small times for resets and sync overheads, a ramp needs $64 \times 2048 \times 10\mu$ s $= 1.3$ s assuming 64 pixels in each channel, the 2048 pixels along the slow direction of the Hawaii-2RG, and the 10 μs (100 kHz) standard pixel clock. The integration time divided by one less than the number of ramps (both parameters set by the operator) defines a subintegration time, and what is left over from one ramp time is filled with the `patItime`.

Looking at Figure 8 on coarser time scales yields Figure 9.

6.4.3 O2DCR

The Omega-2000 douple-correlated read pattern is [7]

```
718 0 4 11 12 1
        717 0 11 19 21 1
                715 0 19 14 20 1
                        714 0 14 15 17 1
                                713 0 15 15 16 1 0 0
                                        712 0 15 7 13 1 60 1 0
                                                patFSync
                                713 0 16 30 32 2048 0 0
                                        712 0 30 21 27 1 60 1 0
                                                patVLS
                                        712 0 31 28 30 1 60 1 0
                                                patRes
                        714 0 15 25 26 1
                                713 0 25 15 16 1 0 0
                                        712 0 15 7 13 1 60 1 0
                                                patFSync
                        714 0 16 18 19 2048
                                713 0 18 35 40 1 0 0
                                        712 0 35 21 27 1 60 1 0
                                                patVLS
```

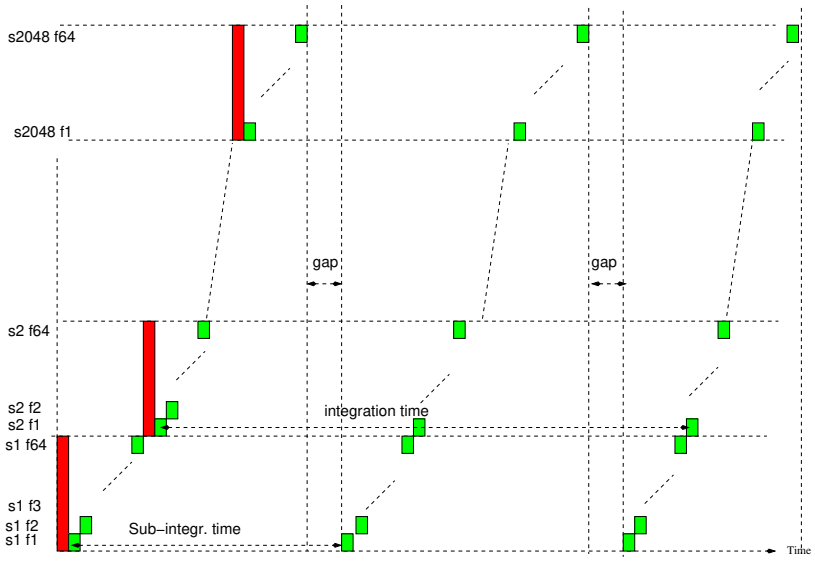


Figure 8: The SRR (sample-up the ramp) pattern. Only the first 3 ramps are indicated. Line sync, frame syncs and other administrative details are not shown.

```
                                712 0 36 38 40 1 60 1 0
                                  patNoRes
                                712 0 37 61 80 2 51 1 0
                                  patPIXnoConv
                                712 0 38 81 100 63 51 1 0
                                  patPIXwConv
                                712 0 39 101 120 1 51 1 0
                                  patConvOnly
                        714 0 17 34 35 1
                                713 0 34 12 15 1 0 0
                                712 0 12 141 142 0 999 3 0
                                  patItime
                                712 0 13 141 142 0 99 2 0
                                  patItime
                                712 0 14 141 142 0 4 1 0
                                  patItime
                        714 0 18 25 26 1
                                713 0 25 15 16 1 0 0
                                712 0 15 7 13 1 60 1 0
                                  patFSync
                        714 0 19 18 19 2048
                                713 0 18 35 40 1 0 0
                                712 0 35 21 27 1 60 1 0
                                  patVLS
                                712 0 36 38 40 1 60 1 0
                                  patNoRes
                                712 0 37 61 80 2 51 1 0
                                  patPIXnoConv
                                712 0 38 81 100 63 51 1 0
                                  patPIXwConv
                                712 0 39 101 120 1 51 1 0
                                  patConvOnly
715 0 20 6 7 1
        714 0 6 2 3 1
                713 0 2 2 3 1 0 0
                712 0 2 153 154 1 4 1 8
                  patNone
```

where the main difference to the LIR mode is that the line resets are bundled first, executed as fast as possible, such that the first read (the refrence frame) has an integrated flux linearly depending

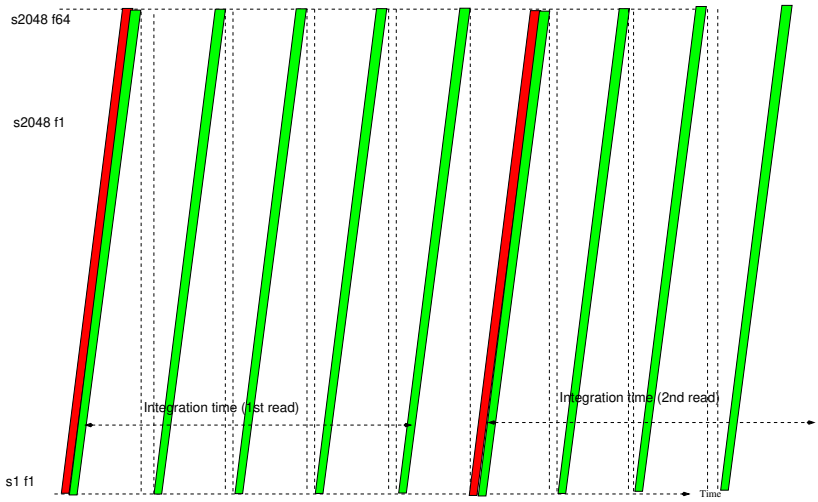


Figure 9: The SRR (sample-up the ramp) pattern with 5 ramps for each read. The figure sketches the first read plus (assuming there is more than one read) the first 4 ramps of the next read.

on the position of the pixel along the slow direction: Figure 10.

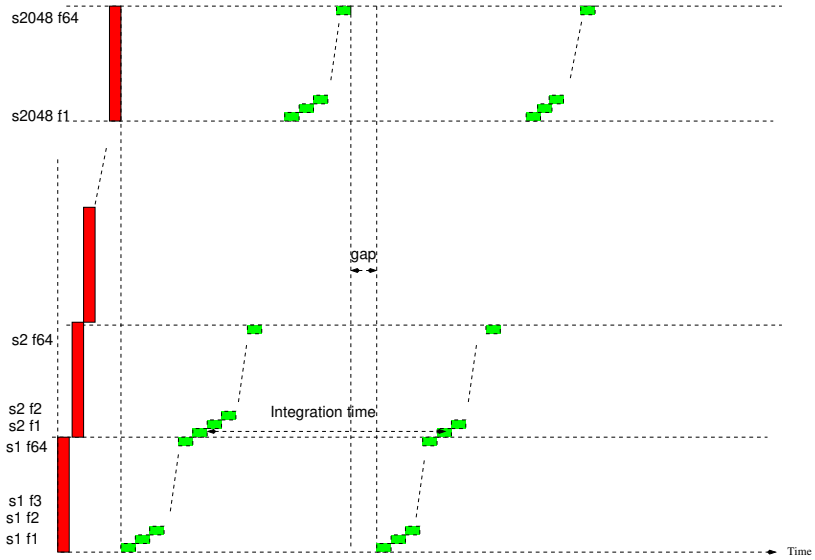


Figure 10: The O2DCR (Omega-2000 douple correlated read) pattern. Only one read is indicated. Line sync, frame syncs and other administrative details are not shown.

6.4.4 FECR

The `FECR` (fast end-of-line correlated read) pattern is [7]

```
718 0 4 27 29 1
        717 0 27 24 25 1
                715 0 24 63 65 1
                        714 0 63 25 26 1
                                713 0 25 15 16 1 0 0
                                        712 0 15 7 13 1 60 1 0
```

```
                                patFSync
                714 0 64 12 15 2048
                        713 0 12 35 37 1 0 0
                                712 0 35 21 27 1 60 1 0
                                        patVLS
                                712 0 36 38 40 1 60 1 0
                                        patNoRes
                        713 0 13 9 12 1 0 0
                                712 0 9 61 80 2 51 1 0
                                        patPIXnoConv
                                712 0 10 61 80 63 51 1 0
                                        patPIXnoConv
                                712 0 11 121 140 1 51 1 0
                                        patNoPIXnoConv
                        713 0 14 40 42 1 0 0
                                712 0 40 41 47 1 60 1 0
                                        patLSyncNoRes
                                712 0 41 28 30 1 60 1 0
                                        patRes
717 0 28 43 45 1
        715 0 43 58 63 1
                714 0 58 25 26 1
                        713 0 25 15 16 1 0 0
                                712 0 15 7 13 1 60 1 0
                                        patFSync
                714 0 59 20 21 2048
                        713 0 20 45 52 1 0 0
                                712 0 45 21 27 1 60 1 0
                                        patVLS
                                712 0 46 38 40 1 60 1 0
                                        patNoRes
                                712 0 47 61 80 2 51 1 0
                                        patPIXnoConv
                                712 0 48 81 100 63 51 1 0
                                        patPIXwConv
                                712 0 49 101 120 1 51 1 0
                                        patConvOnly
                                712 0 50 41 47 1 60 1 0
                                        patLSyncNoRes
                                712 0 51 38 40 1 60 1 0
                                        patNoRes
                714 0 60 34 35 1
                        713 0 34 12 15 1 0 0
                                712 0 12 141 142 0 999 3 0
                                        patItime
                                712 0 13 141 142 0 99 2 0
                                        patItime
                                712 0 14 141 142 0 4 1 0
                                        patItime
                714 0 61 25 26 1
                        713 0 25 15 16 1 0 0
                                712 0 15 7 13 1 60 1 0
                                        patFSync
                714 0 62 19 20 2048
                        713 0 19 35 42 1 0 0
                                712 0 37 61 80 2 51 1 0
                                        patPIXnoConv
                                712 0 38 81 100 63 51 1 0
                                        patPIXwConv
                                712 0 39 101 120 1 51 1 0
                                        patConvOnly
        715 0 44 6 7 1
                714 0 6 2 3 1
                        713 0 2 2 3 1 0 0
                                712 0 2 153 154 1 4 1 8
                                        patSetup1
```

So the line resets occur first and are slowed down (by pixel clocks) such that their average slope is

the same as for a single read. The resets for the first read of the second image are integrated into the reads of the second frame of the first image (Figure 11).

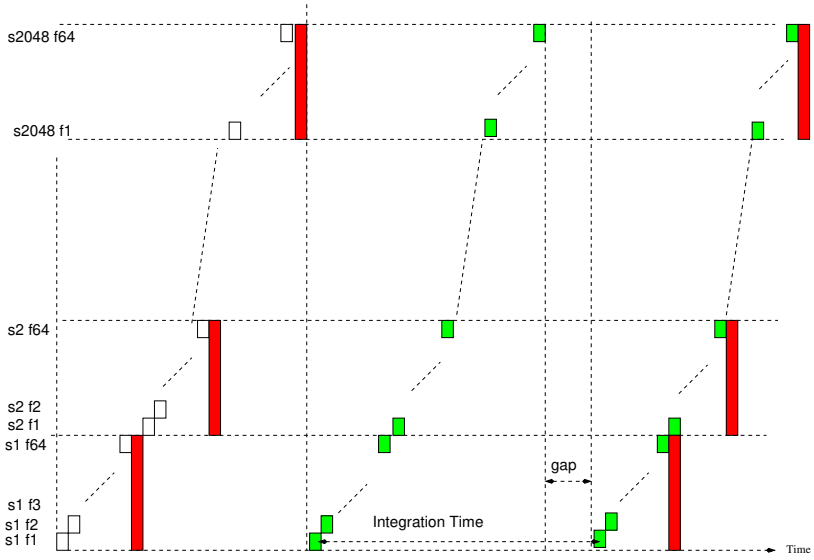


Figure 11: The FECR douple correlated read pattern. Only one read is indicated. The white rectangles are dummy pixel advances without ADC conversion, which do not generate data. Line sync, frame syncs and other administrative details are not shown.

7 TROUBLE-SHOOTING

7.1 Connectivity

7.1.1 Linux Driver

1. If the two parts of the software, shared memory manager and command manager have been shut down inappropriately (by using kill(1)), shared memory blocks are still allocated and sockets left open. This will let another attempt to start GEIRS fail. The general advice is to call

   ```
   geirs_cleanup -t
   geirs_cleanup -v
   ```

   and then try again to start the server; `geirs_cleanup` is in the `PATH` as set up in [8].

2. Error messages of the form

   ```
   libplxmpia.c:233: [plx_find_device] ERROR)  Error in Plx device found
   (u=2/chan=0): ffff ffff [bus ff slot ff fn ff]
   ```

   or

   ```
   ERROR Error: plx_find_device: 'PLX ApiError 516 - ApiNoActiveDriver'
   ```

   mean that the driver for the board that interfaces with the RoCon fiber optics has died or not been installed. This is usually cured by either fixing this at boot time (see [8]) or executing

   ```
   cd $CAMHOME/scripts
   sudo plxstartup
   ```

   It may also mean that the currently installed driver is not the one (not the version) with which the GEIRS source code has been compiled. A typical scenario for this error is that after an upgrade of the PLX driver `start_`*instrument*`_old` fails because the binary residing in the old `bin*` directory is linked to a library in `/usr/src` which has been changed.

7.1.2 Workstation to ROE

Some generic attempts to open a port to the ROE are the commands

`telnet` *server port*

and

`nc -z` *server port*

detailed in the manuals `telnet(1)` and `nc(1)`.

The first sign of then being connected to the ROE are two lines of the kind

```
INFO Seen ROE3 rocon 'DETFPGA' version '3 1 7 5'
INFO Seen ROE3 rocon 'ADCFPGA' version '3 0 2 2'
```

printed by GEIRS to `stdout` at startup (unless the offline mode had been selected). The *Seen* actually indicates that the firmware identifications of the two FPGA had been read. This may later on be repeated from the GEIRS shell with the 550 command in the style of [4]

```
Nirvanamathar> pipe 33 550 0
INFO Seen ROE3 rocon 'ADCFPGA' version '3 0 2 2'
33 550 0 2 3 0 2 2
33 550 0 1
```

or using a neutral (harmless) query for the status of the LED's (510) like

```
Nirvanamathar> pipe 33 510 0
33 510 0 2 0
33 510 0 1
```

In addition one may watch the effect of the associated enable/disable commands for the lights, 508 and 509. (Their meaning is erroneous and swapped in the current documentation [12]).

If the readout electronics does not receive any data from the detector, the ADCs are reading only their internal noise. The detector images then do not show any of their characteristic bad pixels and look very smooth, with a very shallow impression of the horizontal and vertical stripes of the channels, similar to Figure 12, and almost all ADU values in the range ±1.

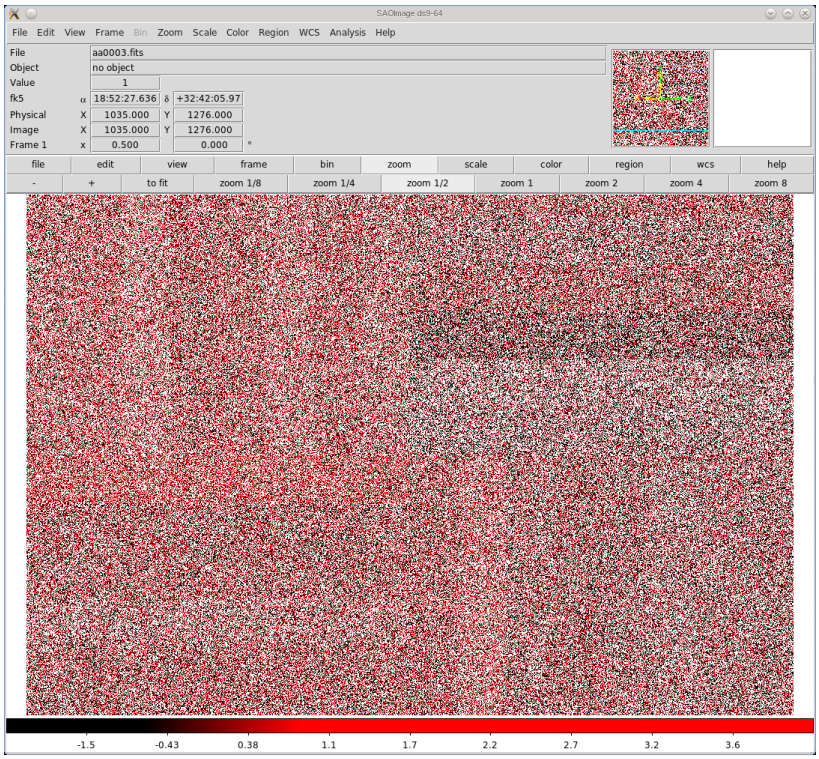


Figure 12: Noisy image of the ROE running ADC's without detector.

Another test picture can be taken by switching the ROE to data simulator mode

```
ctype sfr
roe simadc 1
```

and comparing the frame with Figure 13. This shows actually 32 different values (which appear as

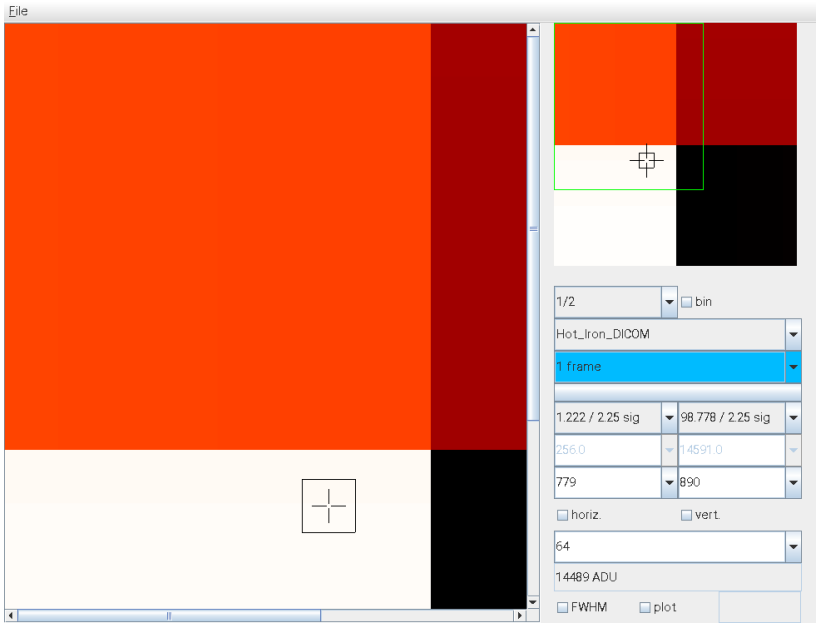


Figure 13: Image from the ROE FPGA data simulator with a single-frame-read for the Hawaii-2.

only 4 different values caused by the contrast) with the following values [6]:

- Quadrant I, channels 1 to 8 (right to left): 256, 273, 290, 307, 324, 341, 358, 375.
- Quadrant II, channels 9 to 16 (top to bottom): 4744, 4761, 4778, 4795, 4812, 4829, 4846, 4863.
- Quadrant III, channels 17 to 24 (left to right): 9216, 9233, 9250, 9276, 9284, 9301, 9318, 9335
- Quadrant IV, channels 25 to 32 (bottom to top): 14472, 14489, 14506, 14523, 14540, 14557, 14574, 14591.

If fiber channels were swapped, only zeros appear.

7.1.3 Data Generator (with GEIRS)

Another basic test of the connectivity between the PCIe board and the workstation is to generate a single full-frame image with the data generator [6],

```
> rotype dgen 30
> ctype sfr
```

which ought to look like Figure 14 in FITS readers or the GEIRS engineering GUI. For other `ctype` settings one should switch the GEIRS GUI to the `single frames` style to observe how the ramp of integers of the data generators is distributed in the GEIRS or FITS coordinates via the index tables (depending on the current rotation and flip parameters).[14]. Also the `BAD` button should

[14]To disable the sometimes confusing rotation and flip matrix, it may make sense to set the two environment variables `CAM_DETROT90` and `CAM_DETXYFLIP` to 0 in the shell before starting GEIRS

be de-selected and the button next above be switched to `AllPix`. For instruments with a single Hawaii-2 RG chip, the pattern of stripes appears as in Figure 15. This tests that the PCIe board

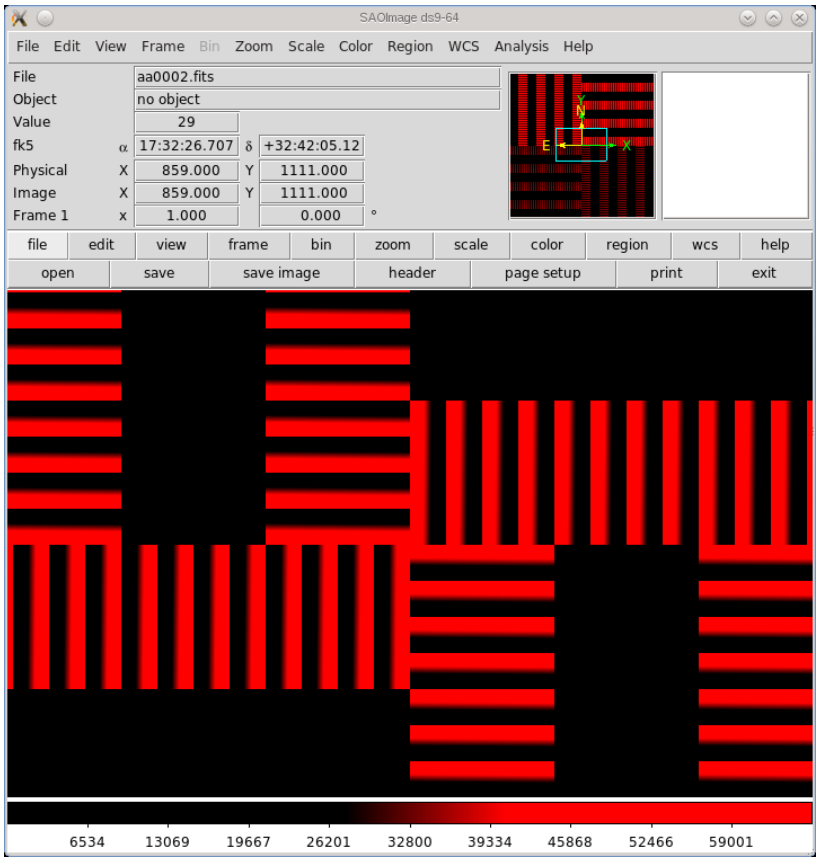


Figure 14: Image with 8 stripes per Hawaii-2 quadrant from the ROE data generator.

plugged into the workstation can be addressed by the driver software. This is more fundamental than the check in Section 7.1.2, because it does not reach further than the 'local' workstation. It does not involve any interface between the workstation and the ROE and should even work if fibers and/or Ethernet connection to the ROE are missing/unplugged.

If this (possibly with a repeat factor larger than 1) is saved with the 'dump' option, `save -d`, a binary file is created in the standard `save`-directory with a file suffix `.dump`. It contains either 16 bit per pixel or 32 bit per pixel (the latter if coadding was involved) in the native byte order of the GEIRS workstation, in the 'raw' order defined by the read-order of the individual ADC's and sequentialization with the data transfer to the workstation. (This is $2 \times 2048 \times 2048$ =8,388,608 bytes in the file for each full Nirvana frame in the `sfr` mode, and a multiple of this for repeats larger than one.)

The data generator creates a ramp with a 32-bit counter, so the output of

```
od -i *.dump
```

should be the list of integers starting at 1:

```
0000000           1           2           3           4
0000020           5           6           7           8
0000040           9          10          11          12
```

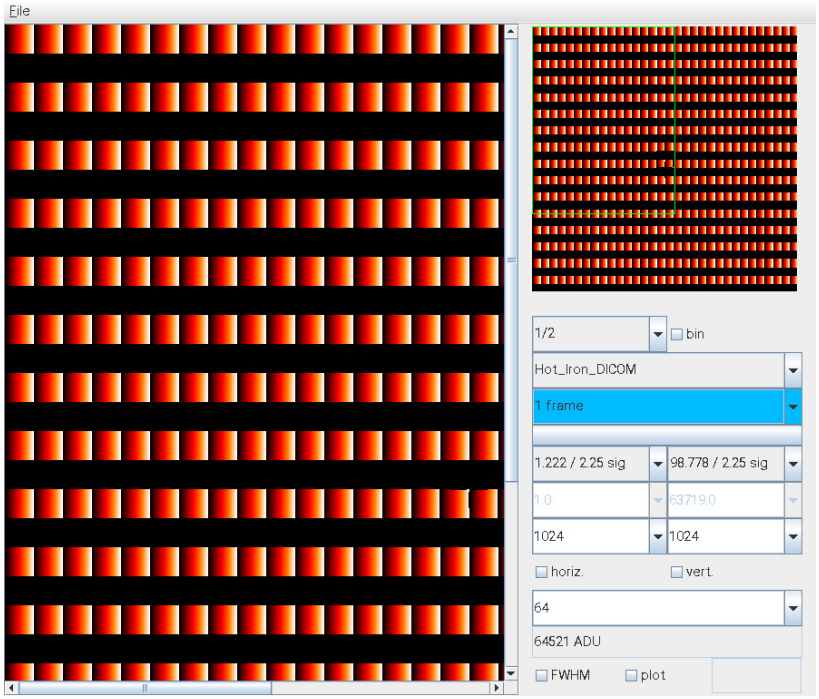


Figure 15: Image with 32 stripes on a Hawaii-2RG from the ROE data generator. This is the view for LUCI if one has set `export CAM_ROTFLIPXY=0 ; export CAM_DETROT90=90` before starting GEIRS

```
0000060          13          14          15          16
0000100          17          18          19          20
0000120          21          22          23          24
0000140          25          26          27          28
..
```

Each 32 datum (double word) of the data generator is intepreted as two successive 16-bit words when received by GEIRS, because this matches the ADC bit size. The stream of $1, 2, 3, \ldots$ $65535, 65536, \ldots$ $4194303, 4194304, \ldots$ issued by the data generator for the first 4k pixels is interpreted in low-word-first order as $1, 0, 2, 0, 3, \ldots$ $0, 65535, 0, 0, 1, 1, 1, \ldots$ $65535, 63, 0, 64, \ldots$. The pattern in the display and FITS files then depends on the type, number and orientation of the chips in the instrument, plus the sequentialization of the one or more 32 channels by the ROE and re-shuffling (demultiplixing) within GEIRS.

Another aspect is that in the `lir` modes GEIRS effectively removes the associated lines of the first frame. The 32-bit counter counts from 1 to 1024 (in the GEIRS sense) while generating one line of 2048 pixels. Because the first line is discarded, the lowest integer actually encountered then is 1025 (in single frame viewing mode...).

The 'dump' option of the `save` command also creates a raw ASCII dump of the primary FITS header in a file named `*.info`. It contains the standard 80 bytes per line without any line feeds or carriage returns. One way to read this strange format for a file named, say, `genDump4.info` is

```
fold genDump4.info | less
```

An endurance test with the data generator is available. The test is prepared by compiling a data parser `dgenDump` and installing the macro `dgenDump.mac` and the 'hook' `dgenDumpQf` as described

in the file `test/INSTALL`. Each time the macro `genDump` then is called it puts a statistics of the `*.dump` files into `$CAMHOME/log/dgenDump.log`.