SKA Project Series

SKA-PSS - Candidate Folding and Optimisation Processing - FLDO

Firenze 18 Dec 2017
Carlo Baffa, Elisabetta Giani, Prabu Thiagaraj

Arcetri Technical Report 3/2017

Abstract

The SKA Pulsar Search requires a fast real time data processing in order to keep-up to the SKA pulsar data production. Such pipeline (named CSP-NIP-PSS pipeline) will be implemented as a collection of software modules running on PSS nodes or on the softwaree accelerators (GPU or FPGA) attached to them.
The FLDO is a software module within the CSP-NIP-PSS pipeline. The FLDO receives the SCL and FFB and produces the optimized candidates and their associated data. This component is responsible for producing the periodicity search data products that form the output of the CSP-NIP-PSS sub-element.
The present document presents two possible implementations on GPU and on FPGA accelerators.

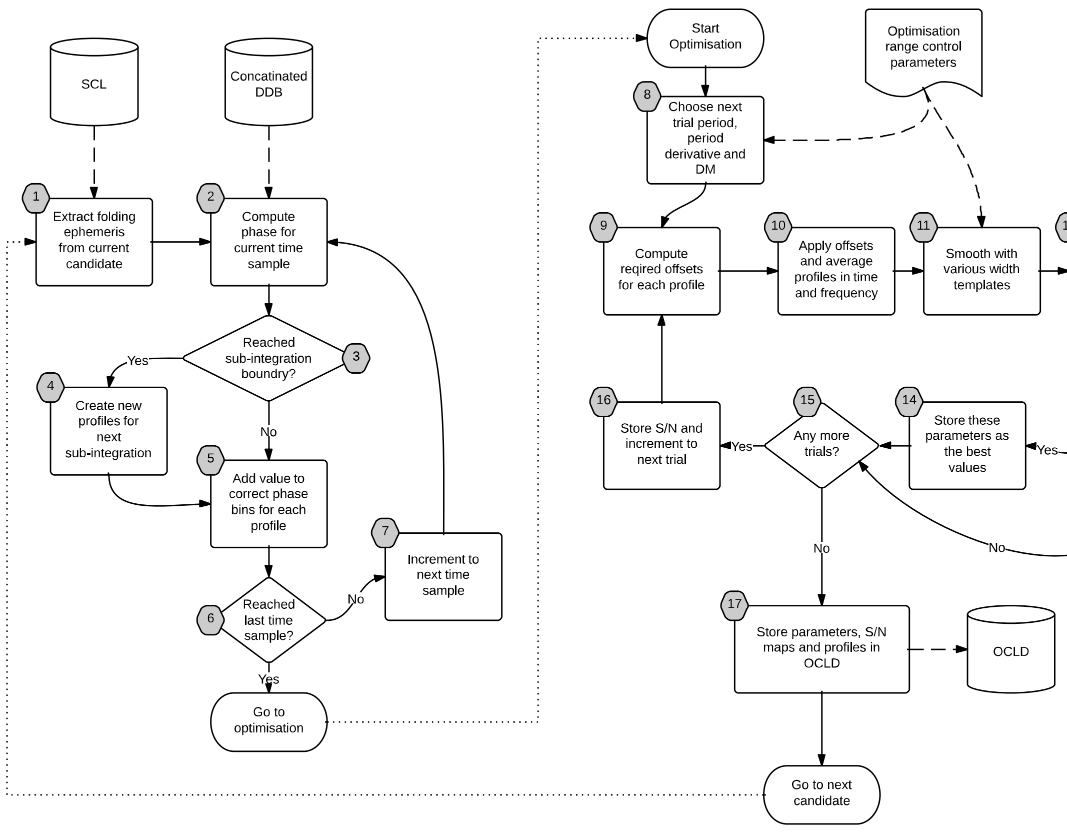
1 Overview

This section presents a general outline of the candidate folding and optimisation processing module (referred to as FLDO) with a brief description of its role in the PSS. FLDO is responsible for producing the periodicity search data products that form the output of the CSP-NIP-PSS sub-element. FLDO is a software/firmware module. A functional description of FLDO is shown in Figure 1.

Inputs to FLDO are:

- The candidate list (SCL) from SIFT,
- The 4096-channel time series data (FFB) from the filter bank buffer creator (FFBC)
- The meta-data from local monitor and control (LMC).

FLDO performs binning, dedispersion around the values specified in each SCL, produces a profile set for each SCL, computes statistics for each profile and sorts them to select the best optimization parameters. Outputs from FLDO are optimized parameters with associated profile (OCLD), all S/N and the relative meta-data.



SCL
Concatinated DDB
1 Extract folding ephemeris from current candidate
2 Compute phase for current time sample
3 Reached sub-integration boundry?
Yes
No
4 Create new profiles for next sub-integration
5 Add value to correct phase bins for each profile
6 Reached last time sample?
No
Yes
7 Increment to next time sample
Go to optimisation
Start Optimisation
Optimisation range control parameters
8 Choose next trial period, period derivative and DM
9 Compute reqired offsets for each profile
10 Apply offsets and average profiles in time and frequency
11 Smooth with various width templates
12 Compute best S/N ratio
13 Is this the largest S/N
Yes
No
14 Store these parameters as the best values
15 Any more trials?
Yes
No
16 Store S/N and increment to next trial
17 Store parameters, S/N maps and profiles in OCLD
OCLD
Go to next candidate

Figure 1: Functional description of FLDO

2 Input Parameters

Table 1: Input parameters used as reference during development

Number of samples	2^{23}
Total Bandwidth (MHz)	~320.0
Channel width (kHz)	~78.125
Sub-bands	64
Number of channels	4096
Sample interval (us)	64
Sub integrations	64
Observation time (s)	536
Bits per sample	8
Number of bytes (GiB)	32
Number of candidates	128 - 1000

Table 1 reports the data input parameters used during development. The code can cope with different values set inside the ranges allowed for real operations.
The details of the FPGA and GPU technology based prototype that we developed are presented here.

3 FPGA

An FLDO architecture design for FPGA technology has been designed and is based on an Arria-10 FPGA PCIe accelerator. This work was done with an industry partner, namely Covnetics. Based on a technology study that was performed, the Arria-10 was found to be a state of the art FPGA that supports hardware based floating point signal processing, high-level design tool support based on OpenCL and is power efficient. Hence, the Arria-10 FPGA was used for this prototype work.
The design developed gives early estimates on power and a feel for expected performance.

3.1 FPGA Design Summary

The parameters used as a reference during the development are:

- observation time is ~536 seconds
- two beams are processed by one FLDO instance
- about 1000 candidates are distributed between every two beams
- about 1024 (by reducing the input from 4096) frequency channels are used for FLDO
- typically 2^{23} time samples are used at each channel
- time samples arrive at 64 us intervals and are 8-bits wide

The FLDO optimises the candidates across 256 DM steps X 256 Period steps X 256 Period Derivative steps [AD33]. The outputs will be folded profile cubes (FPC) with dimensions: 128 phase-bins X 128 sub-bands X 64 sub-integrations. These parameters are programmable to a reasonable extent.

3.2 The Design

The FLDO consists of functions to perform sample integration, channel integration, dedispersion,

folding, optimiser and SNR computation modules as shown in Figure 2.
During the design process, a detailed analysis was made of the algorithm and FPGA resources used by the sub modules and planned for vectorising the functions where suitable. For a few critical modules such as for dedispersion and optimisation, multiple processing architectures were derived, and the one that was best optimised between resource usage and performance was selected. Two major options were identified to fit the design into a target FPGA, viz. a) a monolithic design where the entire FLDO fits into a single FPGA (in other words as a single image), and b) a two part design (as two images partitioning the FLDO processing flow at the dashed line shown in Figure 3), where the entire pre-processing happening before the optimization stage (later referred to as the folding stage) forms one part of the design and the remaining part that includes the optimization and SNR calculation stages forms the second part of the design. The respective merits of the two options were analysed. The two-image based approach would be useful in a situation where the FLDO's external DDR memory is not sufficient to store the entire FFB data for the full observation.

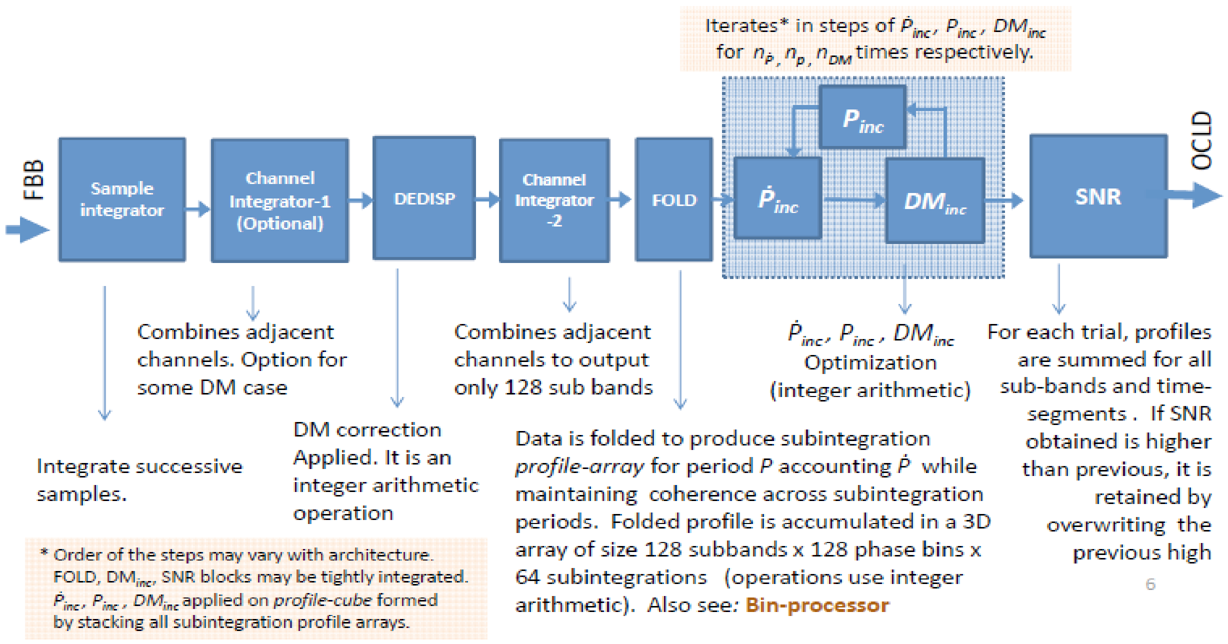

Iterates* in steps of $\dot{P}_{inc}$, P_{inc}, DM_{inc} for $n_{\dot{P}}$, n_P, n_{DM} times respectively.
FBB
Sample integrator
Channel Integrator-1 (Optional)
DEDISP
Channel Integrator -2
FOLD
P_{inc}
$\dot{P}_{inc}$
DM_{inc}
SNR
OCLD
Combines adjacent channels. Option for some DM case
Combines adjacent channels to output only 128 sub bands
$\dot{P}_{inc}$, P_{inc}, DM_{inc} Optimization (integer arithmetic)
For each trial, profiles are summed for all sub-bands and time-segments . If SNR obtained is higher than previous, it is retained by overwriting the previous high
Integrate successive samples.
DM correction Applied. It is an integer arithmetic operation
Data is folded to produce subintegration *profile-array* for period P accounting $\dot{P}$ while maintaining coherence across subintegration periods. Folded profile is accumulated in a 3D array of size 128 subbands x 128 phase bins x 64 subintegrations (operations use integer arithmetic). Also see: Bin-processor
* Order of the steps may vary with architecture. FOLD, DM_{inc}, SNR blocks may be tightly integrated. $\dot{P}_{inc}$, P_{inc}, DM_{inc} applied on *profile-cube* formed by stacking all subintegration profile arrays.
6

Figure 2: Functional description of the FLDO that is considered in the prototype design development

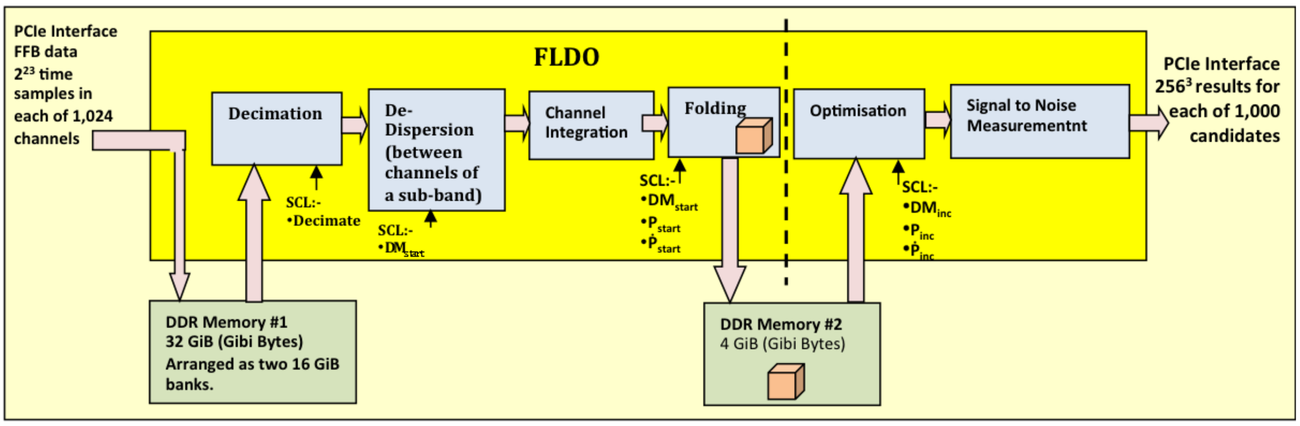
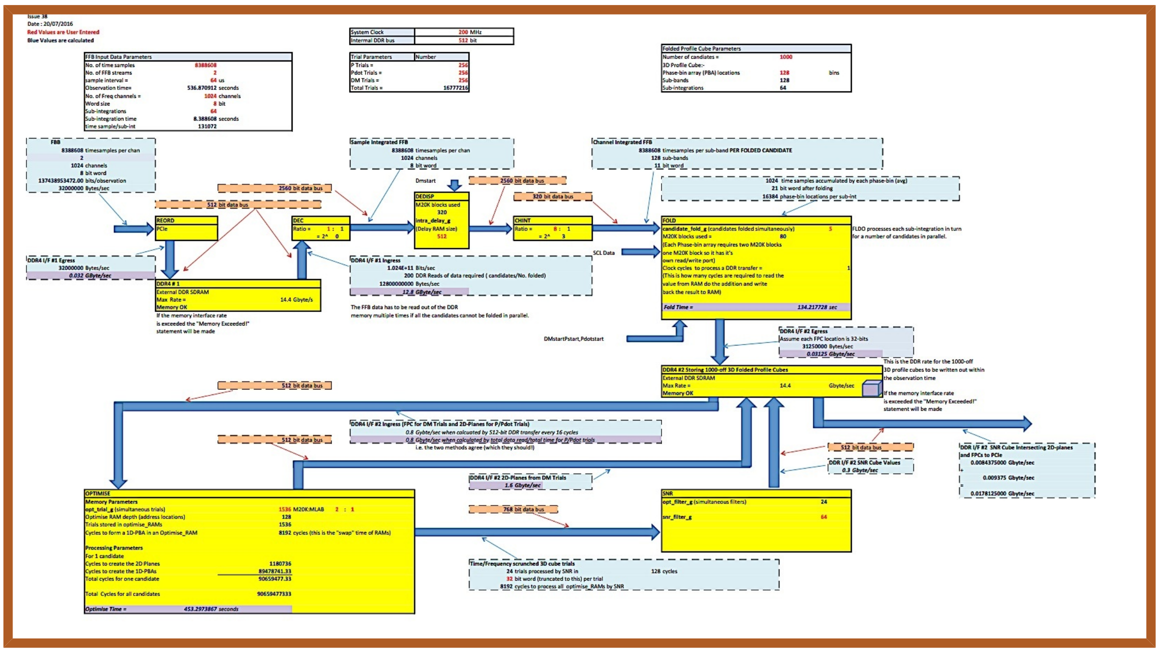

PCIe Interface FFB data 2^{23} time samples in each of 1,024 channels
FLDO
Decimation
De-Dispersion (between channels of a sub-band)
Channel Integration
Folding
Optimisation
Signal to Noise Measurementnt
PCIe Interface 256^3 results for each of 1,000 candidates
SCL:- •Decimate
SCL:- •DM_{start}
SCL:- •DM_{start} •P_{start} •$\dot{P}_{start}$
SCL:- •DM_{inc} •P_{inc} •$\dot{P}_{inc}$
DDR Memory #1 32 GiB (Gibi Bytes) Arranged as two 16 GiB banks.
DDR Memory #2 4 GiB (Gibi Bytes)

Figure 3: FLDO processing Flow showing possible design partition after folding stage

The study began by identifying all major modules and functions that are required to be developed in

the FPGA for the FLDO. All variants within the Arria-10 FPGA family were considered, along with their capabilities to accommodate the number of local buffers, signal processing blocks and portability with number of external DDR memory banks as needed in the design. As a part of this effort, a spreadsheet based FPGA resource and processing performance estimator (Figure 4) that works based on user-supplied parameters, was also developed. The internal, external memory capacity and external memory bandwidth required for the various processing options are also captured in this work.
Naming tags were assigned for each of these modules to trace them during development and improve testing logistics. A detailed internal architecture for each module was designed. shows one example from the optimiser module. Detailed estimates were made of the FPGA resources required to implement all major modules in an Arria-10 FPGA.
Detailed analysis was made of the timing to capture the data flow to FLDO from within the PSS. The timing diagrams shown in Figure 6 capture the PSS data flow up to FLDO. The control flow within the FLDO FPGA platform is captured as a swim-lane diagram shown in Figure 7.



Issue 38
Date : 20/07/2016
Red Values are User Entered
Blue Values are calculated

System Clock	200 MHz
Internal DDR bus	512 bit

Trial Parameters	Number
P Trials =	256
Pdot Trials =	256
DM Trials =	256
Total Trials =	16777216

FFB Input Data Parameters		
No. of time samples	8388608	
No. of FFB streams	2	
sample interval =	64	us
Observation time=	536.870912	seconds
No. of Freq channels =	1024	channels
Word size	8	bit
Sub-integrations	64	
Sub-integration time	8.388608	seconds
time sample/sub-int	131072	

Folded Profile Cube Parameters		
Number of candiates =	1000	
3D Profile Cube:-		
Phase-bin array (PBA) locations	128	bins
Sub-bands	128	
Sub-integrations	64	

FBB
8388608 timesamples per chan
2
1024 channels
8 bit word
137438953472.00 bits/observation
32000000 Bytes/sec

Sample Integrated FFB
8388608 timesamples per chan
1024 channels
8 bit word

Channel Integrated FFB
8388608 timesamples per sub-band PER FOLDED CANDIDATE
128 sub-bands
11 bit word

1024 time samples accumulated by each phase-bin (avg)
21 bit word after folding
16384 phase-bin locations per sub-int

REORD
PCIe

DEC
Ratio = 1 : 1
= 2^ 0

DEDISP
M20K blocks used
320
intra_delay_g
(Delay RAM size)
512

CHINT
Ratio = 8 : 1
= 2^ 3

FOLD
candidate_fold_g (candidates folded simultaneously) 5
M20K blocks used = 80
(Each Phase-bin array requires two M20K blocks
one M20K block so it has it's
own read/write port)
Clock cycles to process a DDR transfer = 1
(This is how many cycles are required to read the
value from RAM do the addition and write
back the result to RAM)
Fold Time = 134.217728 sec

FLDO processes each sub-integration in turn
for a number of candidates in parallel.

DDR4 I/F #1 Egress
32000000 Bytes/sec
0.032 GByte/sec

DDR4 # 1
External DDR SDRAM
Max Rate = 14.4 Gbyte/s
Memory OK

If the memory interface rate
is exceeded the "Memory Exceeded!"
statement will be made

DDR4 I/F #1 Ingress
1.024E+11 Bits/sec
200 DDR Reads of data required (candidates/No. folded)
12800000000 Bytes/sec
12.8 GByte/sec

The FFB data has to be read out of the DDR
memory multiple times if all the candidates cannot be folded in parallel.

DDR4 I/F #2 Egress
Assume each FPC location is 32-bits
31250000 Bytes/sec
0.03125 Gbyte/sec

DDR4 #2 Storing 1000-off 3D Folded Profile Cubes
External DDR SDRAM
Max Rate = 14.4 Gbyte/sec
Memory OK

This is the DDR rate for the 1000-off
3D profile cubes to be written out within
the observation time

If the memory interface rate
is exceeded the "Memory Exceeded!"
statement will be made

DDR4 I/F #2 Ingress (FPC for DM Trials and 2D-Planes for P/Pdot Trials)
0.8 Gybte/sec when calcuated by 512-bit DDR transfer every 16 cycles
0.8 Gbyte/sec when calculated by total data read/total time for P/Pdot trials
i.e. the two methods agree (which they should!)

DDR4 I/F #2 2D-Planes from DM Trials
1.6 Gbyte/sec

DDR I/F #2 SNR Cube Values
0.3 Gbyte/sec

DDR I/F #2 SNR Cube Intersecting 2D-planes
and FPCs to PCIe
0.0084375000 Gbyte/sec
+
0.009375 Gbyte/sec
=
0.0178125000 Gbyte/sec

OPTIMISE
Memory Parameters
opt_trial_g (simultaneous trials) 1536 M20K:MLAB 2 : 1
Optimise RAM depth (address locations) 128
Trials stored in optimise_RAMs 1536
Cycles to form a 1D-PBA in an Optimise_RAM 8192 cycles (this is the "swap" time of RAMs)

Processing Parameters
For 1 candidate
Cycles to create the 2D Planes 1180736
Cycles to create the 1D-PBAs 89478741.33
Total cycles for one candidate 90659477.33

Total Cycles for all candidates 90659477333

Optimise Time = 453.2973867 seconds

SNR
opt_filter_g (simultaneous filters) 24
snr_filter_g 64

Time/Frequency scrunched 3D cube trials
24 trials processed by SNR in 128 cycles
32 bit word (truncated to this) per trial
8192 cycles to process all optimise_RAMs by SNR

Figure 4: FLDO processing metrics spreadsheet showing the relation between the processing choices, memory-IO bandwidth, FPGA resources and Processing Time.

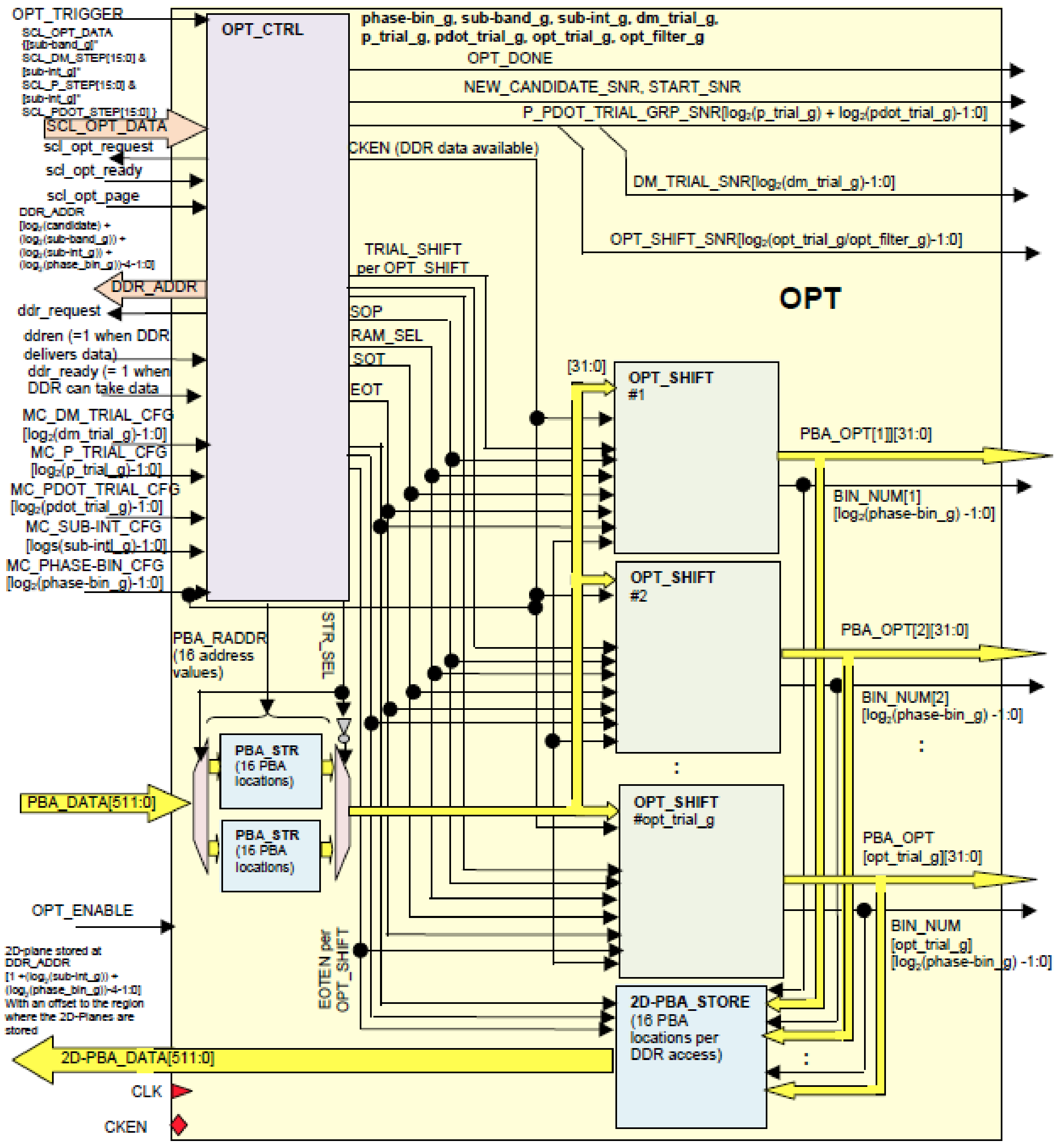

OPT_TRIGGER
SCL_OPT_DATA ([sub-band_g]* SCL_DM_STEP[15:0] & [sub-int_g]* SCL_P_STEP[15:0] & [sub-int_g]* SCL_PDOT_STEP[15:0] }
SCL_OPT_DATA
scl_opt_request
scl_opt_ready
scl_opt_page
DDR_ADDR [log2(candidate) + (log2(sub-band_g)) + (log2(sub-int_g)) + (log2(phase_bin_g))-4-1:0]
DDR_ADDR
ddr_request
ddren (=1 when DDR delivers data)
ddr_ready (= 1 when DDR can take data
MC_DM_TRIAL_CFG [log2(dm_trial_g)-1:0]
MC_P_TRIAL_CFG [log2(p_trial_g)-1:0]
MC_PDOT_TRIAL_CFG [log2(pdot_trial_g)-1:0]
MC_SUB-INT_CFG [logs(sub-int_g)-1:0]
MC_PHASE-BIN_CFG [log2(phase-bin_g)-1:0]
OPT_CTRL
phase-bin_g, sub-band_g, sub-int_g, dm_trial_g, p_trial_g, pdot_trial_g, opt_trial_g, opt_filter_g
OPT_DONE
NEW_CANDIDATE_SNR, START_SNR
P_PDOT_TRIAL_GRP_SNR[log2(p_trial_g) + log2(pdot_trial_g)-1:0]
CKEN (DDR data available)
DM_TRIAL_SNR[log2(dm_trial_g)-1:0]
OPT_SHIFT_SNR[log2(opt_trial_g/opt_filter_g)-1:0]
TRIAL_SHIFT per OPT_SHIFT
OPT
SOP
RAM_SEL
SOT
EOT
[31:0]
OPT_SHIFT #1
PBA_OPT[1][31:0]
BIN_NUM[1] [log2(phase-bin_g) -1:0]
OPT_SHIFT #2
PBA_OPT[2][31:0]
BIN_NUM[2] [log2(phase-bin_g) -1:0]
PBA_RADDR (16 address values)
STR_SEL
PBA_STR (16 PBA locations)
PBA_STR (16 PBA locations)
PBA_DATA[511:0]
OPT_SHIFT #opt_trial_g
PBA_OPT [opt_trial_g][31:0]
BIN_NUM [opt_trial_g] [log2(phase-bin_g) -1:0]
OPT_ENABLE
EOTEN per OPT_SHIFT
2D-plane stored at DDR_ADDR [1 +(log2(sub-int_g)) + (log2(phase_bin_g))-4-1:0] With an offset to the region where the 2D-Planes are stored
2D-PBA_STORE (16 PBA locations per DDR access)
2D-PBA_DATA[511:0]
CLK
CKEN

Figure 5: Optimisation Module - an example for the detailed design.

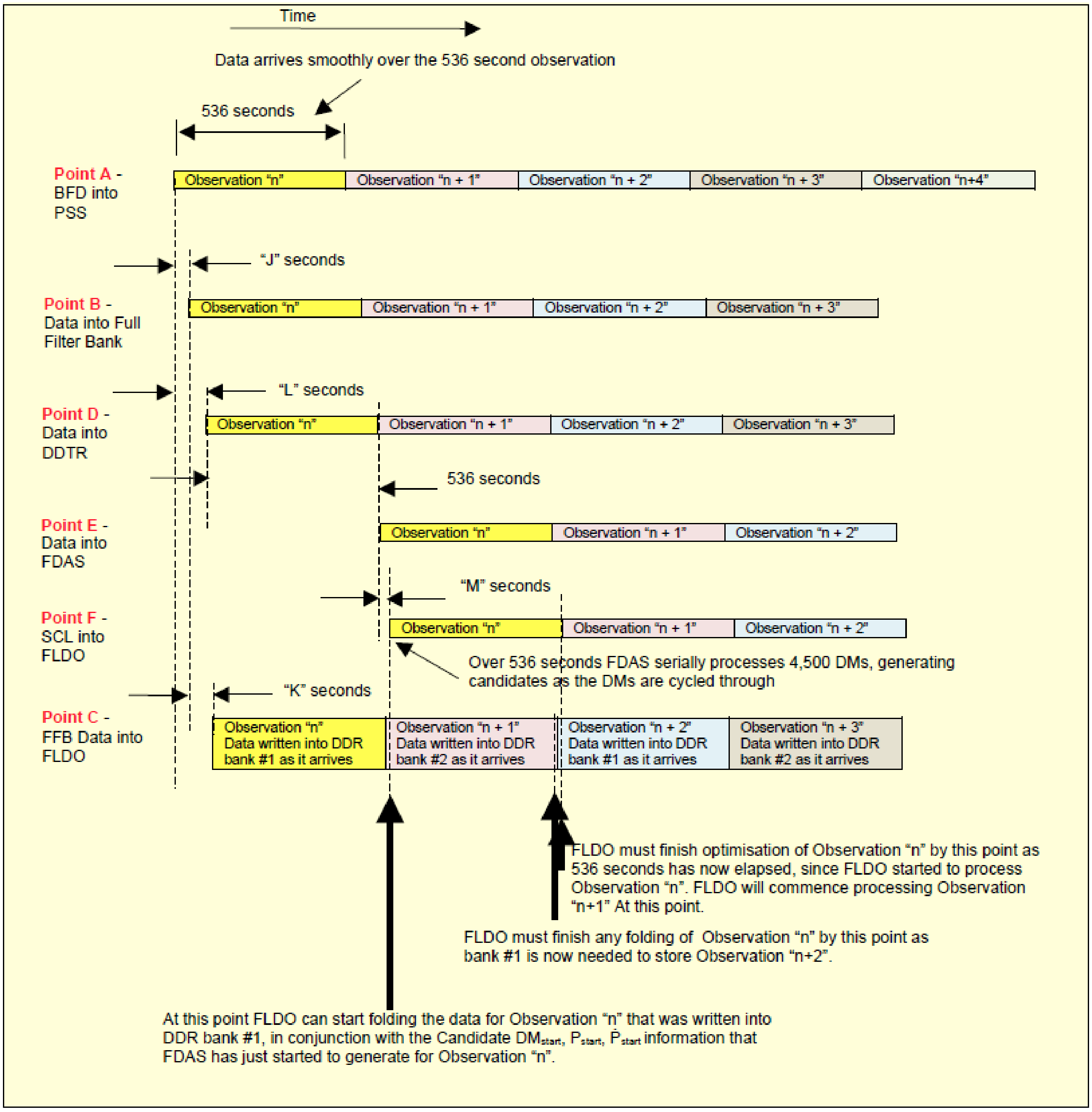

Time
Data arrives smoothly over the 536 second observation
536 seconds
Point A - BFD into PSS
Observation "n"
Observation "n + 1"
Observation "n + 2"
Observation "n + 3"
Observation "n+4"
"J" seconds
Point B - Data into Full Filter Bank
"L" seconds
Point D - Data into DDTR
536 seconds
Point E - Data into FDAS
"M" seconds
Point F - SCL into FLDO
Over 536 seconds FDAS serially processes 4,500 DMs, generating candidates as the DMs are cycled through
"K" seconds
Point C - FFB Data into FLDO
Observation "n" Data written into DDR bank #1 as it arrives
Observation "n + 1" Data written into DDR bank #2 as it arrives
Observation "n + 2" Data written into DDR bank #1 as it arrives
Observation "n + 3" Data written into DDR bank #2 as it arrives
FLDO must finish optimisation of Observation "n" by this point as 536 seconds has now elapsed, since FLDO started to process Observation "n". FLDO will commence processing Observation "n+1" At this point.
FLDO must finish any folding of Observation "n" by this point as bank #1 is now needed to store Observation "n+2".
At this point FLDO can start folding the data for Observation "n" that was written into DDR bank #1, in conjunction with the Candidate DMstart, Pstart, Ṗstart information that FDAS has just started to generate for Observation "n".

Figure 6: PSS Pipeline Timing Diagram for Data to FLDO. The timing diagram captures the data flow within PSS before every new set of data that arrives at FLDO for processing.

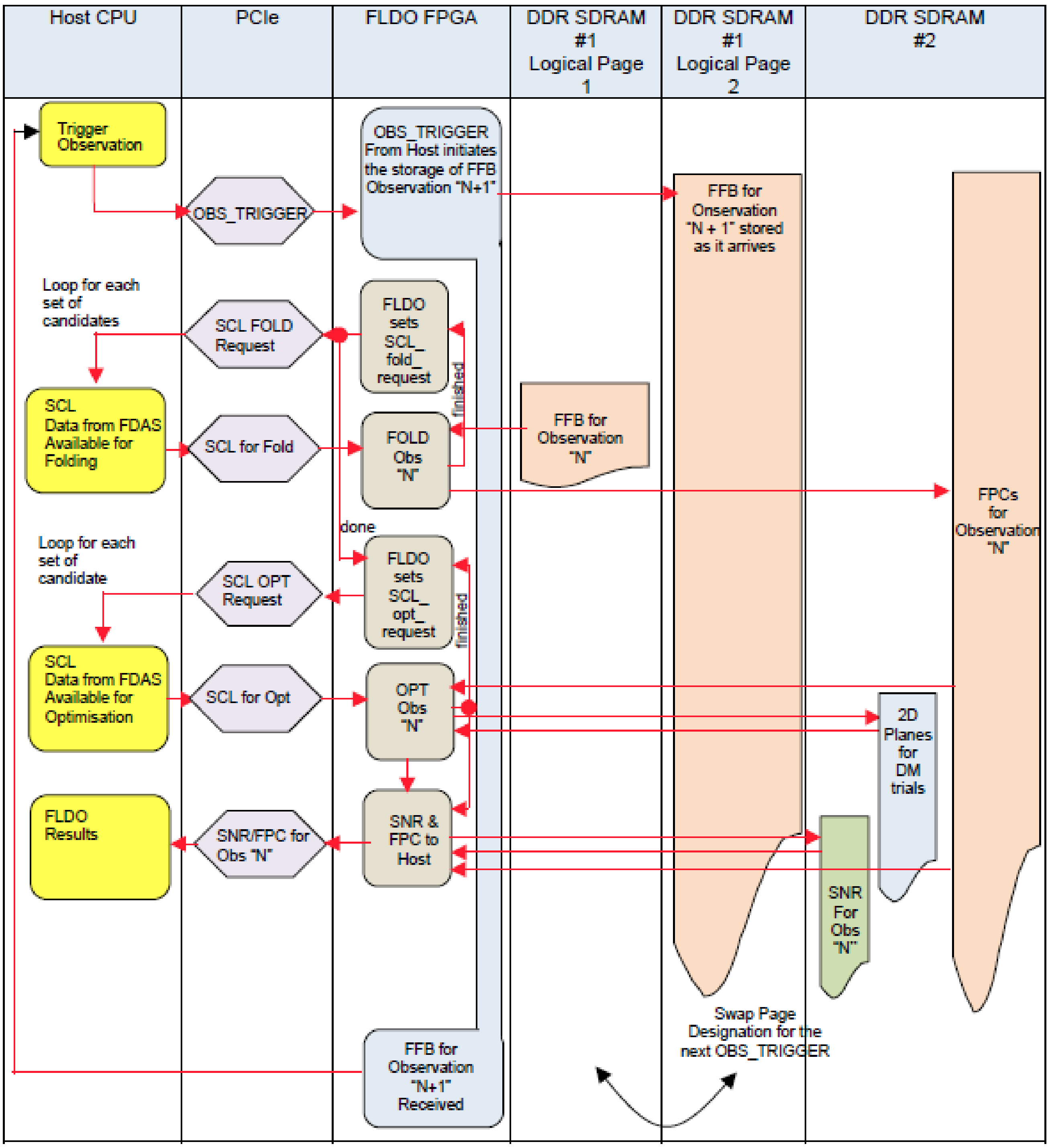

Host CPU
PCIe
FLDO FPGA
DDR SDRAM #1 Logical Page 1
DDR SDRAM #1 Logical Page 2
DDR SDRAM #2
Trigger Observation
OBS_TRIGGER
OBS_TRIGGER From Host initiates the storage of FFB Observation "N+1"
FFB for Onservation "N + 1" stored as it arrives
Loop for each set of candidates
SCL FOLD Request
FLDO sets SCL_ fold_ request
finished
SCL Data from FDAS Available for Folding
SCL for Fold
FOLD Obs "N"
FFB for Observation "N"
FPCs for Observation "N"
done
Loop for each set of candidate
SCL OPT Request
FLDO sets SCL_ opt_ request
finished
SCL Data from FDAS Available for Optimisation
SCL for Opt
OPT Obs "N"
2D Planes for DM trials
FLDO Results
SNR/FPC for Obs "N"
SNR & FPC to Host
SNR For Obs "N"
Swap Page Designation for the next OBS_TRIGGER
FFB for Observation "N+1" Received

Figure 7: FLDO operation swim-lane diagram. The sequence of control flow within FLDO FPGA platform is captured in this diagram.

Based on the metrics developed from this work, the expected performance for the FLDO, resource utilisation and the power performance is presented here. For this analysis, it is assumed that the FLDO is implemented as a single image. For the power analysis, Altera's early power estimator tools were used. Based on this study, early estimates were made for the power performance of the future Startix 10 FPGAs.

4 GPU

The current approach to the software implementation on GPU of candidate folding is a C-CUDA module. A prototype, named *Origami,* has been developed and tested. The results are presented in Section 1.

4.1 GPU Implementation Summary

The present *Origami* implementation consists of a four-phase process, executed in sequence for each pulsar candidate.
First, data is read from source (can be a disk file, a memory area or a network stream) and split into a number of sub-integrations (default 64).
Data is then transposed and converted from 8-bit unsigned integers to 32 bit float numbers. If the candidate period is longer than a specific value, data is also pre-binned. The pre-binning operation consists of summing adjacent time samples. The number of time samples to sum is always a power of 2, so subsequent pre-binning can benefit from the previous computation.
The next phase is folding. For each sample, the phase relative to the candidate period is computed and corrected for DM and acceleration. Each sample is placed in the appropriate phase bin or proportionally split between two successive phase bins. An array of weights takes care of the different numbers of samples added to each candidate phase bins. For each candidate, the algorithm produces a matrix of profiles (up to 128 bins long), one for each sub-integration and group of frequencies (default 64x64).
The fourth phase is the optimisation of candidate parameters, by means of a grid optimization search. The optimization result is then normalized and forwarded to SDP for further processing.

4.1.1 Phase 1 - Read and Split

The first phase of FLDO consists of getting data and splitting the input stream in a small number of parts (sub integration of parameter) in order to perform optimisation in the fourth phase. Data input are assumed to consist of a continuous stream of 8-bit numbers. Such values are the Stokes-I, and are ordered first in frequency (fastest index) and then in time. During development of the PSS observation, input values are assumed as 64 µs of time resolution, 4096 frequency channels, and 536 s integration time. These values will result in a 34 GB data chunk. The code can cope with different value sets inside the ranges allowed for real operations. The *Origami* implementation is able to process the maximum specified data length of 1800 s (145 GB).

4.1.2 Phase 2 – Transpose, conversion and Pre-binning

4.1.2.1 Transpose and conversion

In phase 2, data are transposed (corner turning) and converted from 8-bit unsigned integers to 32 bit float numbers. The execution of the corner-turning of each block of input data facilitates coalesced reads for GPU and it results in better execution performance.
Input data is processed in blocks of 64 sub-integrations. Each sub-integration is then divided in 64 sub-bands. These are the default values, but the folding program can be modified using the argument options.

4.1.2.2 Pre-Binning

The pre-binning procedure consists of summing adjacent time samples. The number of time samples to sum is a power of 2: in the prototype the maximum value is 16 but there is no hardwired limit. The pre-binning reduces the size of data to be processed and also makes it possible to process

pulsars with longer spin periods.
The pulsar candidate input list is pre-sorted by ascending periods. The candidates are divided in to groups. In Table 2 assuming the default sampling rate of 64 μs, two possible strategies are shown. The *Fastest* method strategy gives the fastest execution times, at the expense of final time resolution, while the *Conservative* method gives the maximum resolution compatible with the hardware limits. Maximizing the S/N seems to favour the *Conservative* approach.

Table 2: Candidate Pre-binning Strategies

Candidate Period - Fastest	*Candidate Period - Conservative*	*Re-bin Factor*
$P < 0.001$ sec	$P < 0.004$ sec	**1**
$0.001 < P < 0.002$ sec	$0.004 < P < 0.008$ sec	**2**
$0.002 < P < 0.004$ sec	$0.008 < P < 0.016$ sec	**4**
$0.004 < P < 0.008$ sec	$0.016 < P < 0.032$ sec	**8**
$0.008 < P < 0.063$ sec	$0.032 < P < 0.063$ sec	**16**

The current implementation of the *Origami* program uses a maximum re-binning factor of 16, so the FLDO can detect pulsars with periods up to 63 ms. There is no hard maximum for the folding factor. The present limit is only as a temporary provision, to limit the complexity of the set-up code during the design phase. Adding higher pre-bin, pulsar periods can be ramped up to seconds without problems. Also, due to the very small number of samples involved in this module, the execution time is insignificant and doesn't affect the overall processing.

4.1.3 Phase 3 – Coherent Folding

The folding algorithm consists of a synchronous summation of input data in order to improve the S/N.
The folding process gives a representation of the distribution of the data as a function of the phase relative to the pulse period. Folding is an operation where each sample is attributed to a reduced set of phase bins. The natural number of phase bins is calculated using the formula (the factor 2 comes from the Nyquist rule):

$$n_{phases} = 2 * P/t_{sampling}$$

The current implementation in GPU of the FLDO module works with a maximum of 126 bins[1]: higher values limit the GPU resources with an impact on the number of threads running in parallel. With this limit on the number of the phases and a sampling time of $t_{sampling} = 64$ μs, the FLDO module could only handle pulsars with periods up to 4.1 ms without re-binning.
For each block, folding produces two arrays of n_{phases} elements: the first with the coherent sum of the data (intensity profile) and the second with the weights of each phase (weights profile).
FLDO implements two different coherent summing approaches. The first one will consist of a synchronous summation with phase split, see Figure 8.
The second summation approach doesn't split input data, but each sample is summed in the bin which corresponds to its central phase (Figure 9). This approach is 20-30% faster, but sometimes it implies a small resolution loss.
At the end, the whole process results in 64 x 64 intensity profiles and 64 x 64 weight profiles.

[1] We use 126 and not 128 as a maximum to avoid the necessity to check for 'end of buffer' condition. This saves the use of a costly modulo function. Phase bins 127 and 128 are respectively added to phase 0 and 1 at the end of computation.

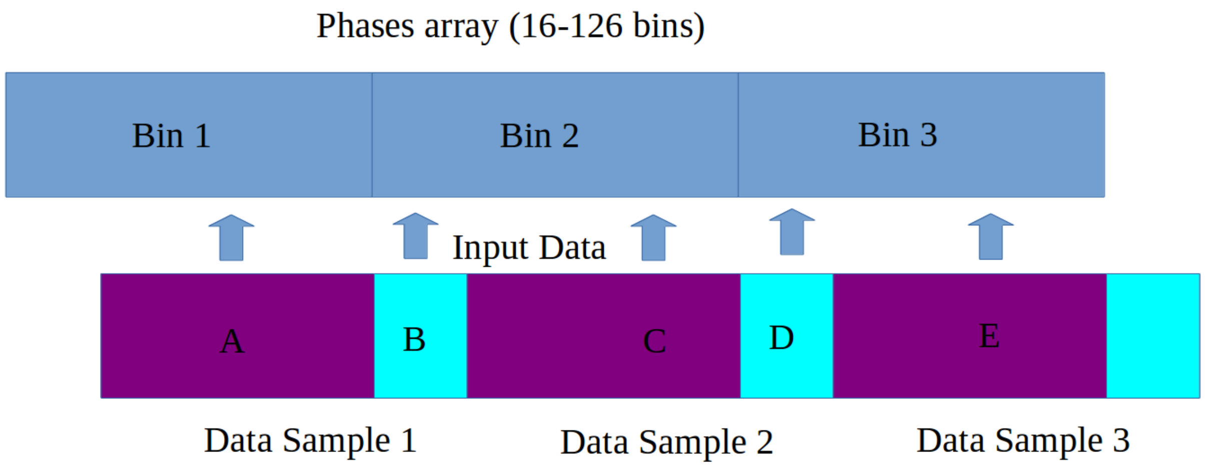


Figure 8: Synchronous summation with phase split: input data is split according to its phase relative to the pulsar candidate phase array. In the above figure, Data Sample portion A will be summed up to Bin1, Data Sample portions B and C will be summed up to Bin2

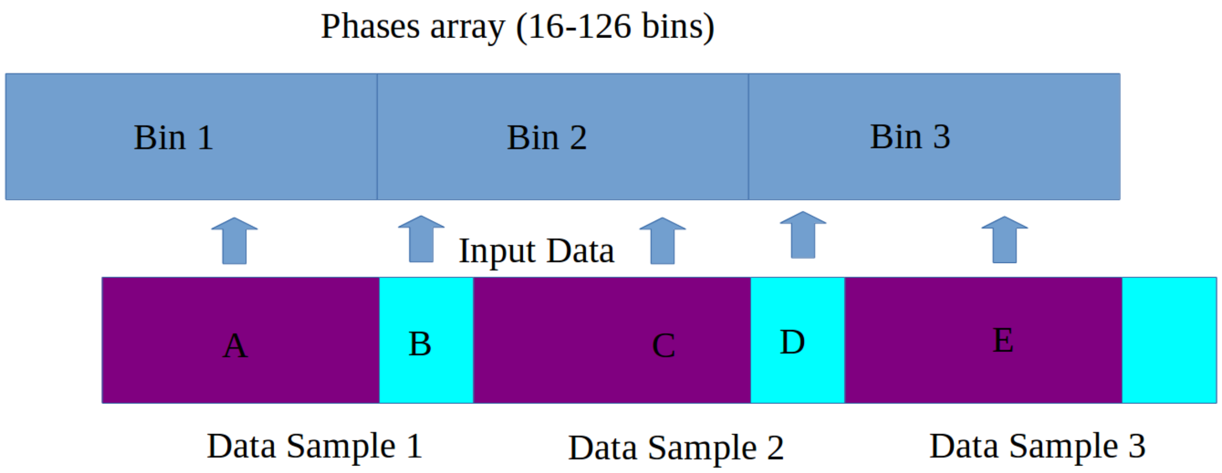


Figure 9: Synchronous summation without phase split: input data is summed to the phase bin corresponding to its central phase. In the above figure, Data Sample A will be summed up to Bin1, Data Sample B will be summed up to Bin2, Data Sample C will be summed up to Bin3

4.1.4 Phase 4 – Normalization and Optimization

The last steps are the normalization and the optimization phases.

4.1.4.1 Normalization

Each intensity profile is divided by the corresponding weight profile, correcting for the non-constant number of samples added to each phase bin. This portion has a negligible impact on the total execution time. Then the value of the mean of the current measure is computed[2] and subtracted from the normalized profile. This portion also has a negligible impact on the total execution time (0.5 %).

2
We used a customized version of Cuda::Thrust library

4.1.4.2 Optimization

In the last step of FLDO, the candidate parameters are optimized. The optimization procedure computes the effects of small perturbations on candidate parameters on the final S/N. The procedure selects the perturbed parameter set with higher S/N. This optimization is performed by a simple grid search, well suited for a parallel engine.
Some efforts have been devoted to the possible use of an “amoeba style” optimization approach. The results were not good, probably because of the 'clumpiness' of the numerical function.
The optimization phase, initially a small portion of FLDO total time, in the last iterations has assumed a larger and larger fraction of total time.

4.1.4.3 Reduction

The normalized profiles are summed along frequencies and sub-integrations to produce the reduced profiles, which are the result of FLDO computation for this prototype.

5 Results

This section reports the current performance of FLDO prototypes. Also, some historical trends are presented.

5.1 FPGA Performance

The main results obtained from this design effort are:

1. **An FPGA based design for the FLDO**: This is an implementable detailed design documentation and guideline derived from the basic FLDO requirement and best technology choices available at this time [AD33]

2.

Result: The development and design are described in AD33.

3. **FLDO processing metrics**: This is a spreadsheet model of the FLDO FPGA architecture that calculates processing time and FPGA resource utilisation based on user entered FLDO input parameters and choices. [AD33: Appendix D]

   Result: The design is split into two logical blocks, namely folding and optimisation. Based on the design that we made, we find that an FPGA based folding process would take 134.22 seconds and optimisation would take 456.80 seconds. The optimisation process is expected to start once first few folded cubes are produced from the folding process. Due to the expected overlapped operation between the two processes, the entire FLDO consisting of folding and optimisation for 1000 candidates across two beams can be completed in the real time of 536.7s that is considered for this design.

4. **Memory size and bandwidth requirement metrics:** We have developed a set of equations that calculate the external DDR storage space for input data, results, internal buffer size for any intermediate results and the internal memory bandwidths that are required to meet the FLDO specifications. The user parameters are input data size, number of buffering stages, memory bus clock speed, internal bus width and expected bandwidth efficiency. Parameters such as bus width comes from the design choice, and clock speed and bandwidth efficiency mainly come from the memory technology choice [AD33: Appendix E]

   Result: Results from this work, especially helped us to understand the constraints imposed on our design for the FPGA size and platform memory bank architecture, thus allowed us to optimally arrive at an implementable design for the prototype purpose.

5. **Power estimation**: We developed spreadsheets that translate design information into power estimates for a given FPGA part. The user parameters are: a detailed FPGA resource list that is usually auto generated from another spread-sheet (derived after a trial synthesis using Quartus), clock speed, expected data toggling rates, ambient conditions, cooling choices such as heat-sink size and air-flow capacity and FPGA part details [AD33: Appendix F]

   Result: For one set of optimal design parameters that we used, the estimated FLDO FPGA nominal power consumption is about 26 Watts and a silicon-junction temperature is 63.0 °C for an Arria-10 GX1150 hosting folding and optimisation processing on a single FPGA image. The data toggle rate can affect the power consumption over a range from about 15 W to 38 W as observed from the simulations. Besides this result, our study also captures effects of changes to airflow and ambient temperature to power and silicon junction temperature. We have also extended this work to study the anticipated power performance for the future technology based FPGAs namely Startix 10 using the early power estimate tools made available to use through University of Manchester-Altera-Covnetics NDA arrangements. The power performance obtained for Stratix 10 FPGA appears attractive and the details are presented in the Covnetics document [AD33: Appendix F].

6. **Future upgrades and scalability:** The basic design can be extended to support more candidates or to process additional beams when higher capability FPGAs and high bandwidth and capacity memories are available during the implementation stage in future.

5.2 Conclusions from the FPGA prototyping work

The FLDO design was developed by working with industry partner Covnetics [AD33], and the design details are captured with high accuracy that can easily be translated for any implementation and validation purposes. Careful attention has been paid to the testing and verification procedures that will be required later in the project. Over all, this effort towards prototyping of FLDO using FPGA technology resulted in a design that we think is readily realizable with high level of confidence.

5.3 GPU performances

In Table 3the GPU execution n times and the relative S/N for FLDO are shown using different folding algorithms. Candidate periods are distributed uniformly up to 49 ms. The assumed folding approach is always conservative (see Table 3) on phase length choice (maximizing S/N). All values are relative to an Nvidia GTX1080 GPU. For these simulations 128 candidates, 128 phases, 64 sub integrations and 64 sub bands are assumed.

Table 3: GPU Execution Times

Number of measures **Number of channels**	**Split coherent folding conservative**		**Non-Split coherent folding conservative**	
	Folding time & Optimization	**S/N**	**Folding time & Optimization**	**S/N**
2^{23} (536s) 4096	56s + 37s	337	47s + 37s	350
2^{22} (268s) 4096	28s + 37s	213	23s + 37s	226
2^{21}(134s) 4096	14s + 37s	171	12s + 37s	174
2^{23} (536s) 1024	18s + 37s	360	15s + 37s	350
2^{22} (268s) 1024	9s + 37s	222	7s + 37s	231
2^{21}(134s) 1024	5s + 37s	179	4s + 37s	184

From all measured timings, some indications can be inferred. Most trends are linear, but there should be differentiation between 'linear increase' (something of the form y=ax+const) and 'linear proportionality' (something of the form y=ax). Moreover, due to the relative small number of trials compared to the total parameter space, second order effects cannot be ruled out while changing more than one parameter at time.
Main trends (derived from a larger set of computations):

1. Folding time is proportional to integration time, and increases linearly with the numbers of candidates.
2. Folding time is proportional to the number of frequency channels, while optimization time has little or no dependence on it.
3. Optimization times have no significant dependence on integration time, and are proportional to the number of candidates.
4. Split phase is consistently faster by a factor of 20-25%, with no loss of S/N ratio.
5. Optimization time is proportional to the number of points explored. S/N seems to increase with the decrease of the spacing of the points explored, but it becomes less likely to get to the maximum.