Digital design: The hardest thing I've ever done

Keith Bannister - Co-learnium - 17 June 2021

Or

The ASKAP/CRAFT Coherent upgrade

FRB2020

Keith Bannister - keith.bannister@csiro.au
@pleasefftme

With: Xinping Deng, Li Bang
On behalf of the CRAFT collaboration

Or

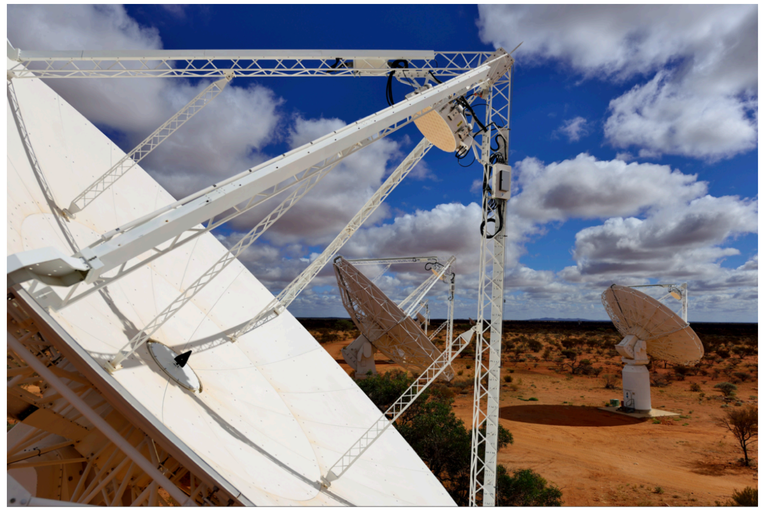
Localising an FRB/day by shoving 2 million Youtube viewers into a fridge

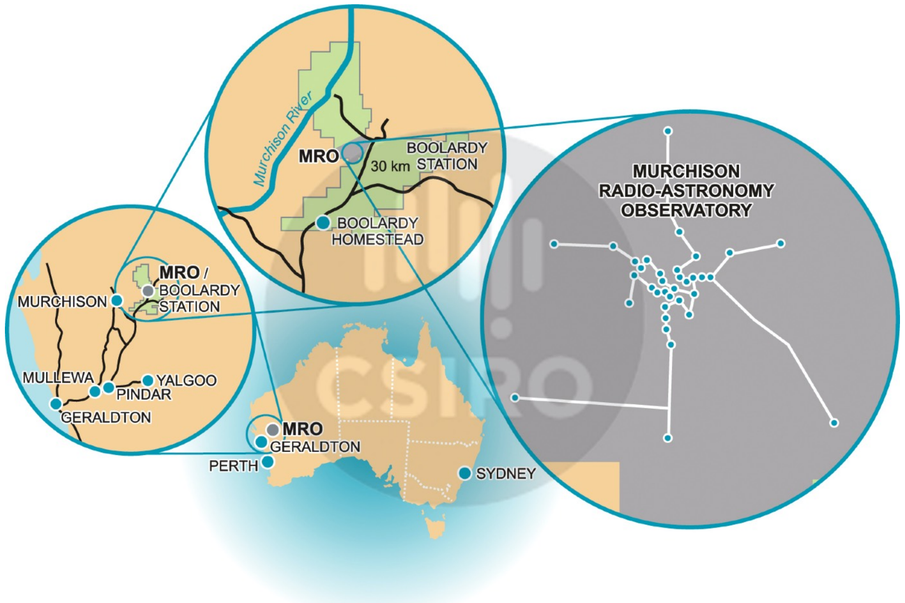
FRB2020
Keith Bannister - keith.bannister@csiro.au
@pleasefftme

With: Xinping Deng, Li Bang
On behalf of the CRAFT collaboration

ASKAP

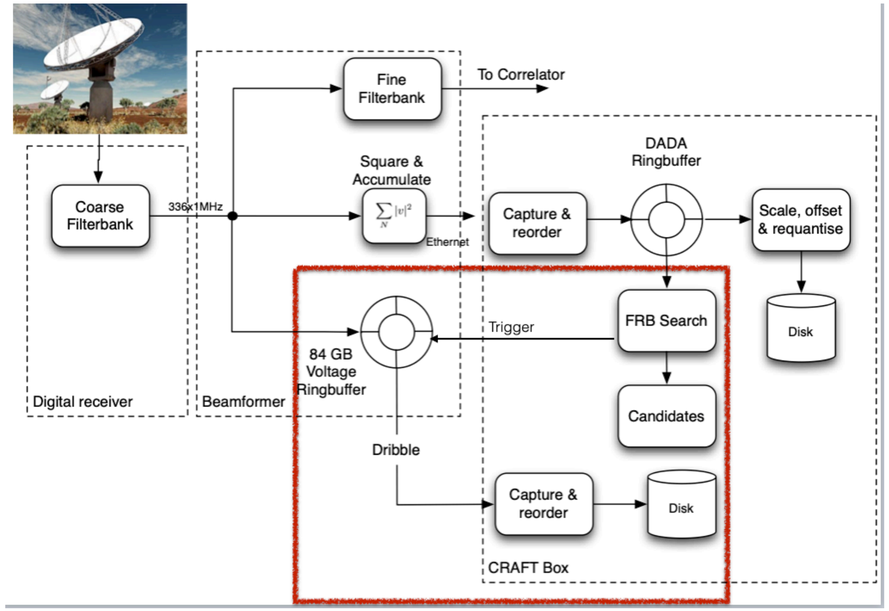
- 36 x 12m antennas
- Each antenna: 188 receivers
- Each: 36 beams = ~ 30 deg^2 per antenna
- Total: 1296 beams = 1000 deg^2 in Flyseye
- Tuning: 0.7-1.8 GHz
- 336 x 1 MHz channels
- Autocorrelations with ~1ms time resolution
- 6km max baseline = 6” synthesised beam at 1.4 GHz
- 7000 Receivers
- 20 000 Lasers
- 15 500km of fibre: Sydney to LA.
- 72 Tbits/sec off samplers = 10% of the internet
- 1.6 MW PV Solar Array - enough to power a small village
- 2.5 MWhr Lithium ion battery

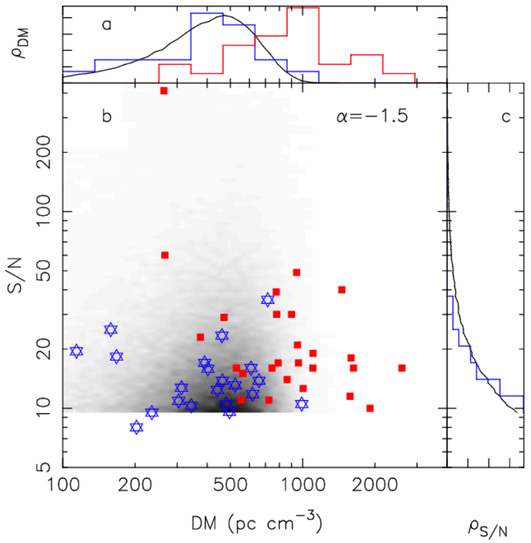


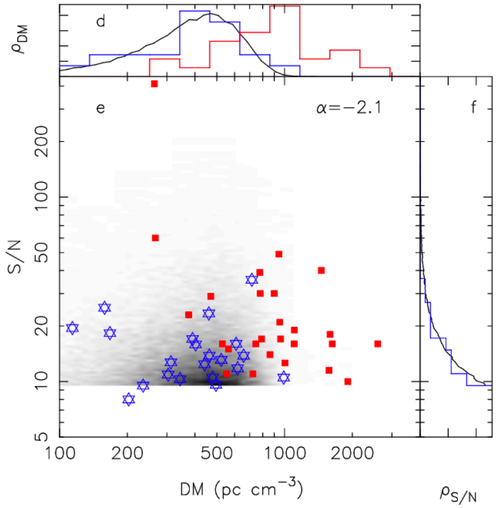


But: It isn't *fully* armed an operational

- Current processing: Incoherent sum sensitivity $\propto N^{1/2}$
- Proposed method - Fast imaging of visibilities: Fully coherent sensitivity $\propto N$
- i.e. *5x* more sensitive than current method (don't process outer 6 antennas)
- ~0.5-2 FRBs/day each with with ~arc second localisation (dependant on logN-logS)





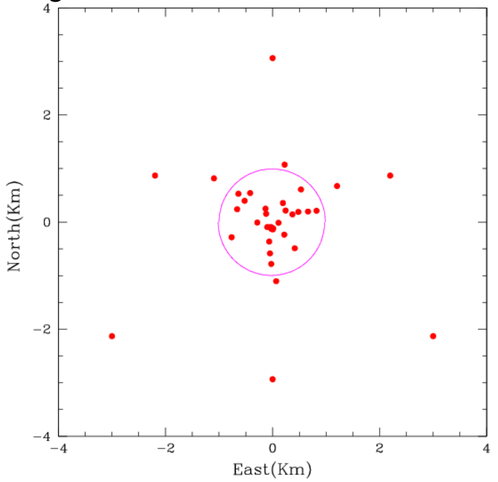


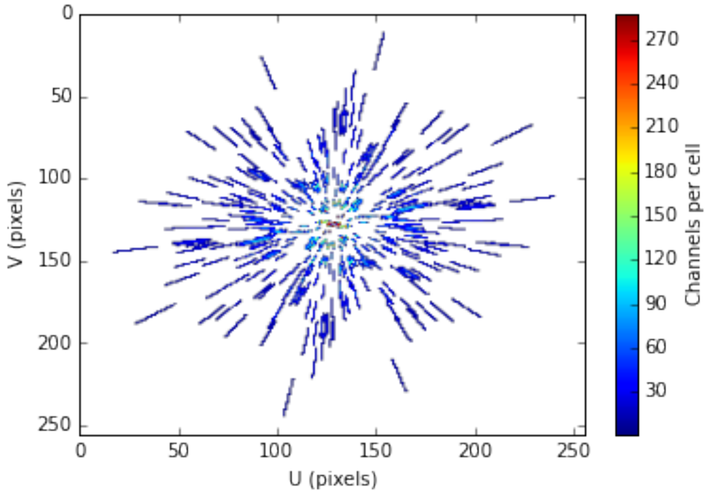
Shannon+17 Supplementary

ASKAP Fast Imaging

- Typically we'll use 30 antennas within 2 km diameter - each with 288 channels, 1ms integrations.
- Discrete sampling of UV plane = Npix x Npix = 256x256 cell grid.
- The positions of the baselines are essentially static for ~30 seconds (Earth rotates slowly).
- Every millisecond we get visibilities for 30 antennas = 436 baselines x 288 channels ~ 130k measurements
- Many of the channels fall in the same cell in the grid. We'll average those (in a preprocessing step - the FDMT) by a factor of ~25x
- There are only ~6500 non-zero points in the UV plane i.e. ***the UV plane is 90% zeros (!)***

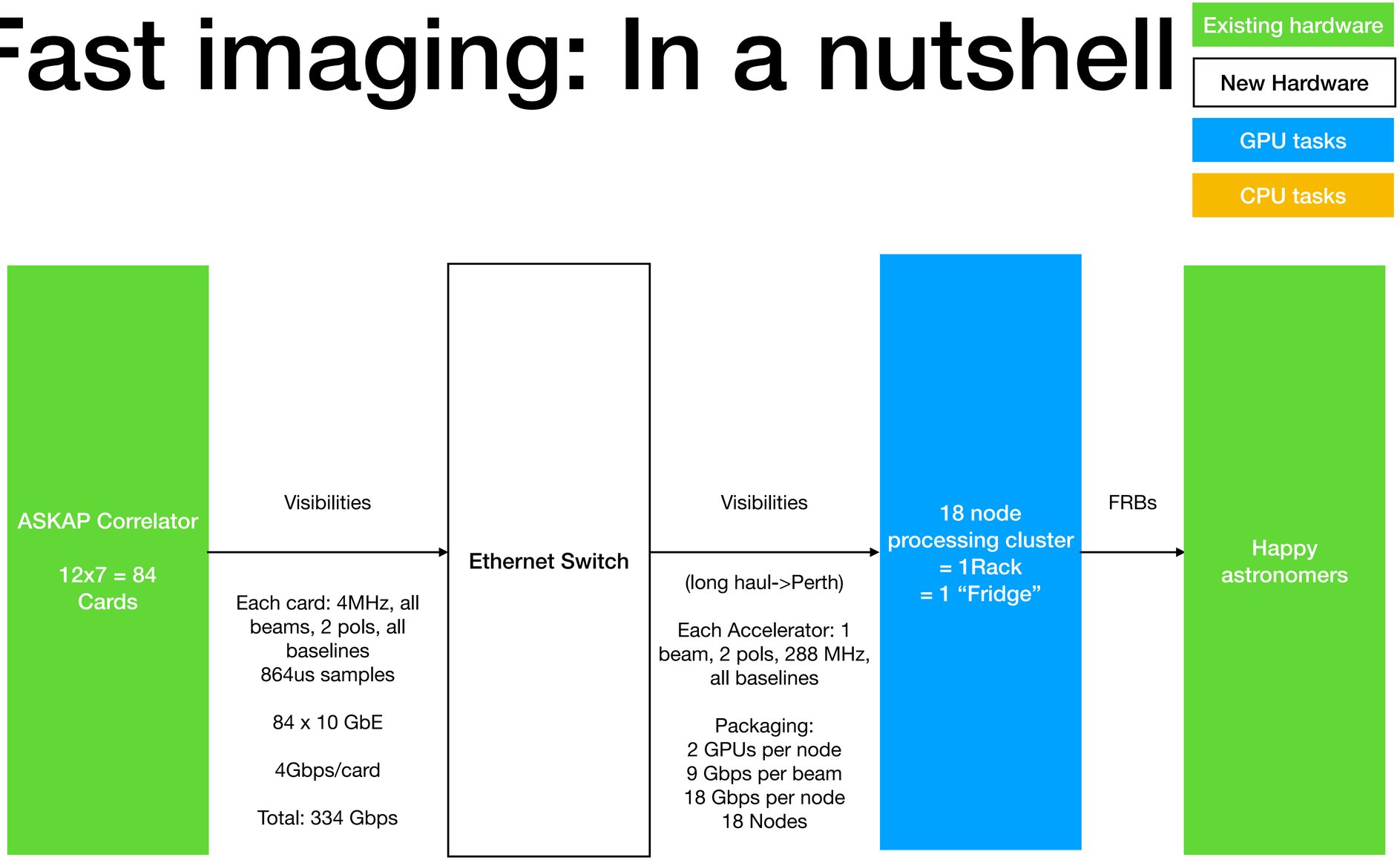
ASKAP array configuration = circle has radius=1km ~ 30 antennas



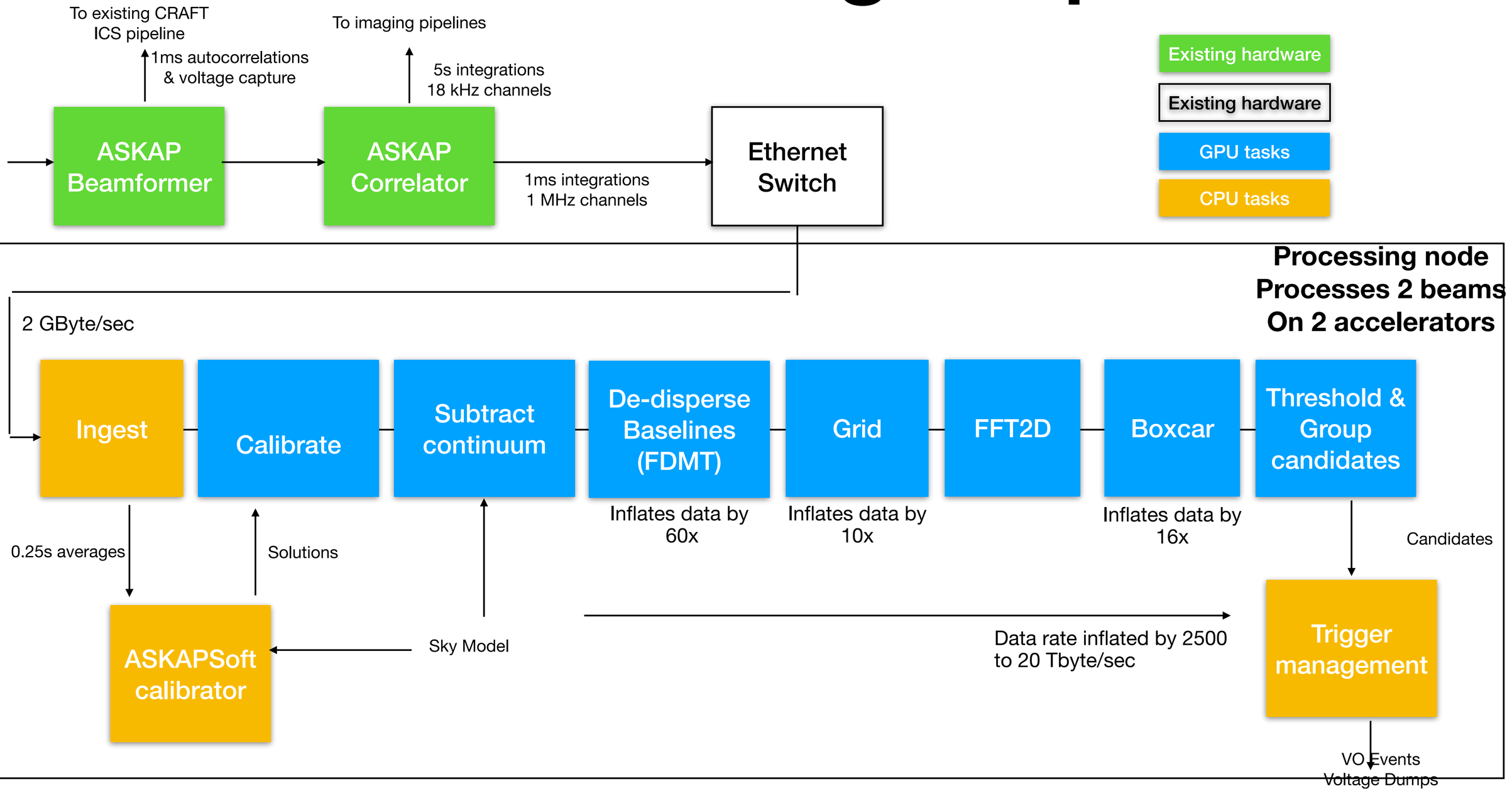


Gridded UV plane - coloured by number of channels per cell
22 antennas in 1km radius.

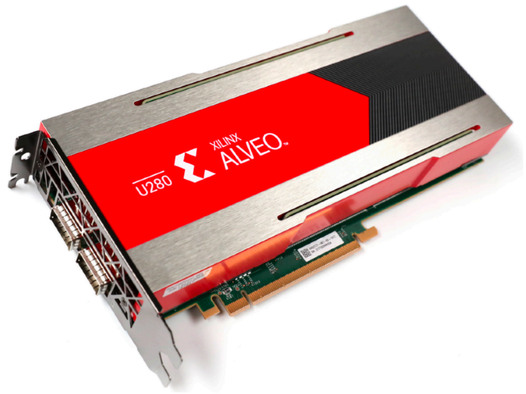
Fast imaging: In a nutshell



Processing Steps



GPU vs FPGA Smackdown



	NVIDIA V100 GPU	Xilinx Alveo U280	Xilinx Alveo U50
Cost	~$15k AUD	~$10k AUD	$3k AUD
Memory	16/32 GB	8GB HBM + 32GB DDR	8GB HBM
“Memory Bandwidth”	900 GB/sec	460 GB/Sec	316 GB/sec
“Computing”	“100 TFlops”	24.5 Tops (int8)	(less than U280)
L1 Cache	96KB x 84 = 8MB	41 MB(!)	28 MB
L1 cache rate	24 TB/sec	30 TB/sec	24 TB/sec
Power	300W	225 W	75W
Programming	CUDA :-)	HLS :-(	HLS :-(

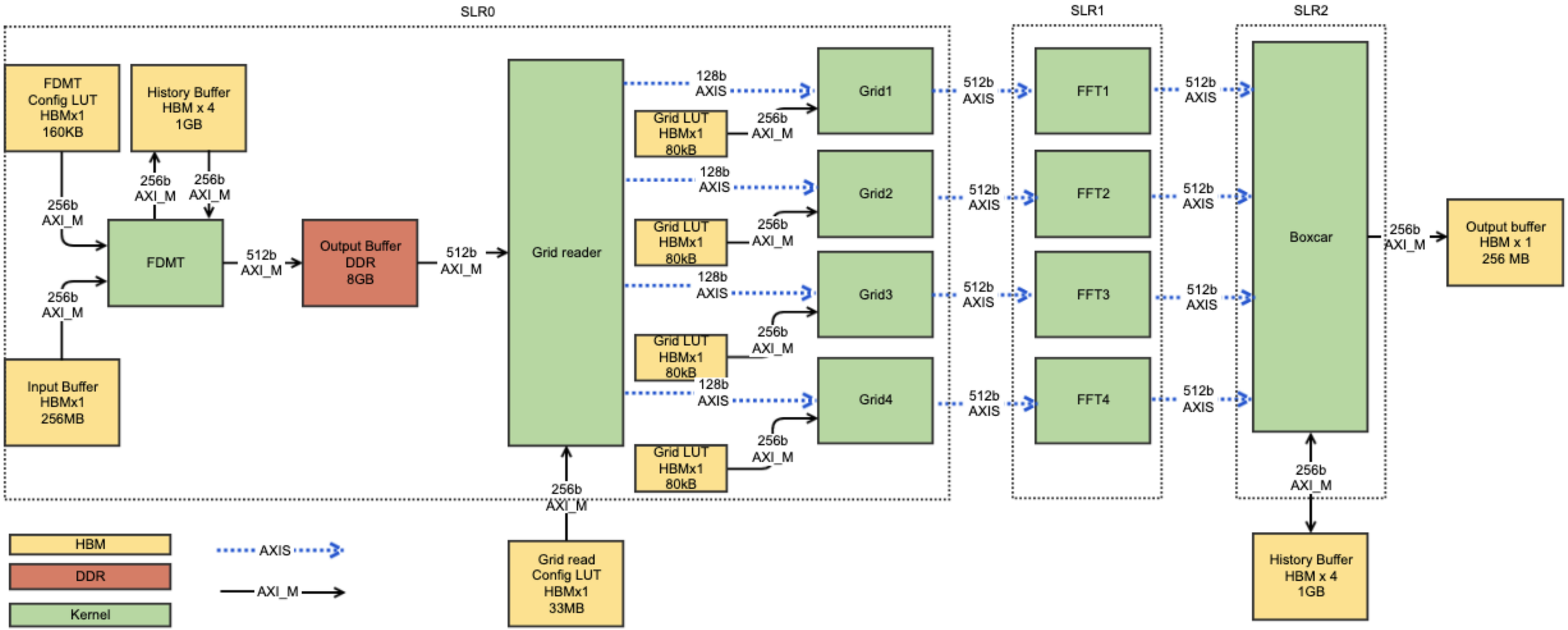
Designing hardware

- Usually done in VHDL/Verilog - which is thought to be difficult
- New Thing: HLS - you can write in a software-language = EASY!
- BUT: You're still designing hardware - it turns out, that's the thing that's hard. The language is secondary.
- Things you have to think about:
  - Consumption of different types of on-chip resource: LUTs, BRAM, URAM, SRL
  - Way in which off-chip memory is accessed: Data width, burst sizes, read vs write, outstanding reads/writes
  - Routing: How different functions are connected - both the data path ***and the control path*** *(in HLS the control path can be hidden from you a bit).*
  - Timing issues - how the architecture is implemented on the chip affects how fast it will run.

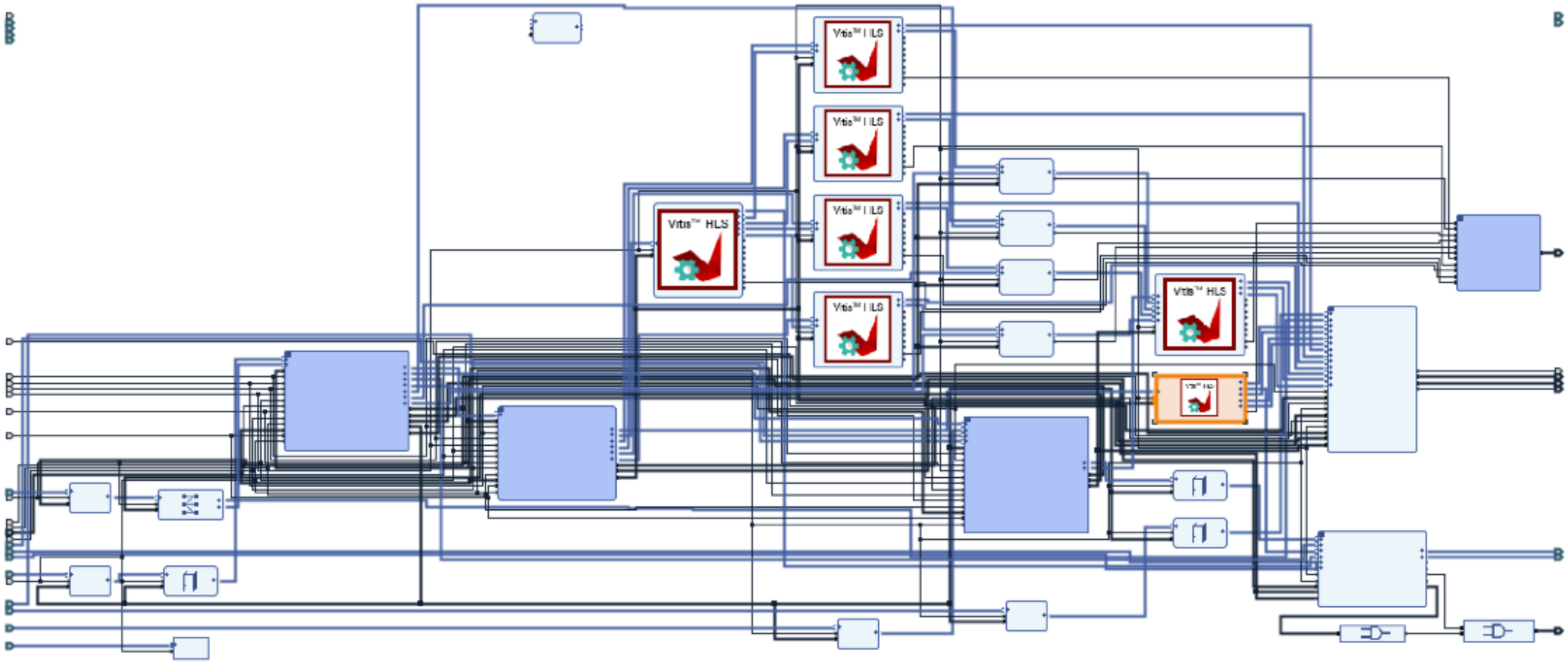
Challenges we've had

- Not being digital engineers *and* not realising it's a problem
- Massive learning curve
- First project *and* pushing the limits of what's possible.
- Bugs/limitations in the tools
- Working around bugs/limitations in the hardware

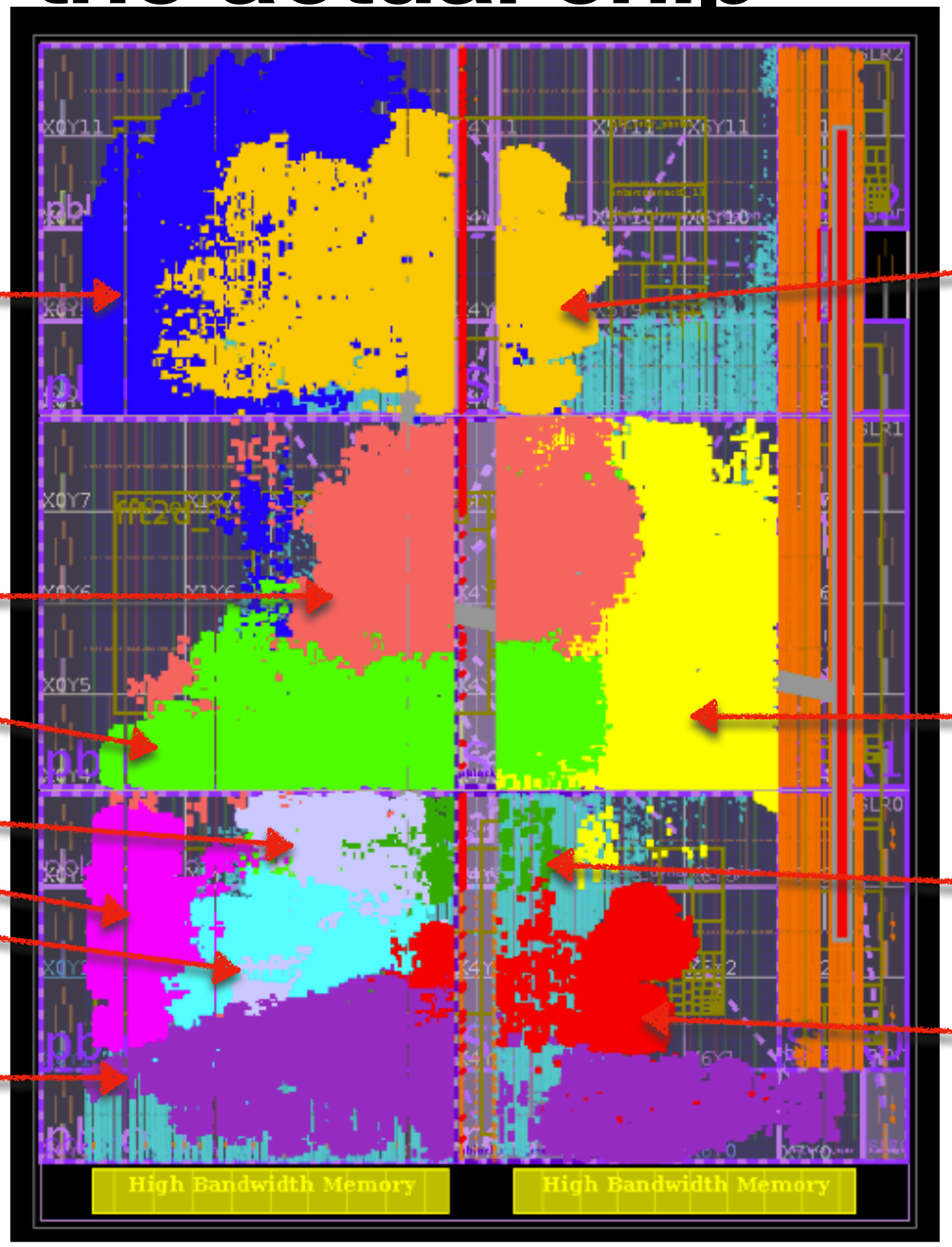
Block diagram when I draw it



Block diagram generated by the tool



Placement on the actual chip

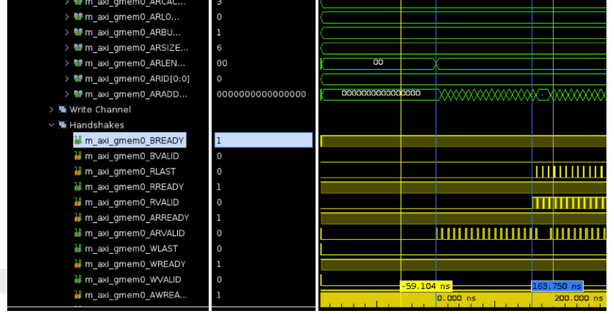


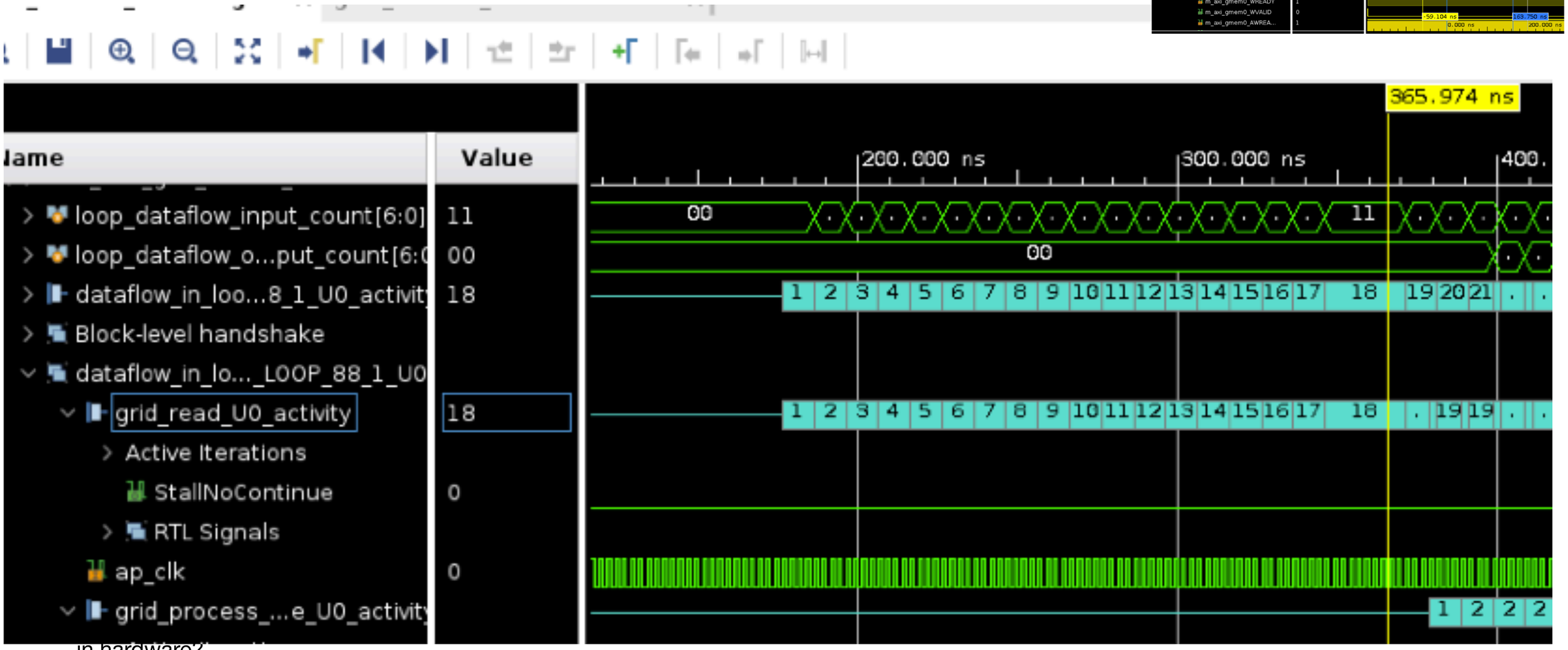
- fdmt_tunable_c32_1 (pfm_dynamic_fdmt_tu
- fft2d_1 (pfm_dynamic_fft2d_1_0)
- fft2d_2 (pfm_dynamic_fft2d_2_0)
- fft2d_3 (pfm_dynamic_fft2d_3_0)
- fft2d_4 (pfm_dynamic_fft2d_4_0)
- hmss_0 (pfm_dynamic_hmss_0_0)
- init_cal_combine_mss (pfm_dynamic_init_ca
- init_combine_mss (pfm_dynamic_init_combi
- interrupt_concat (pfm_dynamic_interrupt_c
- krnl_boxc_4cu_1 (pfm_dynamic_krnl_boxc_4
- krnl_grid_4cu_1 (pfm_dynamic_krnl_grid_4cu
- krnl_grid_4cu_2 (pfm_dynamic_krnl_grid_4cu
- krnl_grid_4cu_3 (pfm_dynamic_krnl_grid_4cu
- krnl_grid_4cu_4 (pfm_dynamic_krnl_grid_4cu
- krnl_sync_stream_uv_4cu_1 (pfm_dynamic_k

```
sp=fdmt_tunable_c32_1.pin:HBM[9]
sp=fdmt_tunable_c32_1.pout:DDR[0]
sp=fdmt_tunable_c32_1.histin:HBM[10:13]
sp=fdmt_tunable_c32_1.histout:HBM[10:13]
sp=fdmt_tunable_c32_1.pconfig:HBM[14]

sp=krnl_sync_stream_uv_4cu_1.in:DDR[0]

sp=krnl_grid_4cu_1.lut:HBM[0]
sp=krnl_grid_4cu_2.lut:HBM[1]
sp=krnl_grid_4cu_3.lut:HBM[2]
sp=krnl_grid_4cu_4.lut:HBM[3]

sp=krnl_boxc_4cu_1.history_in:HBM[4:7]
sp=krnl_boxc_4cu_1.history_out:HBM[4:7]
sp=krnl_boxc_4cu_1.cand:HBM[8]

slr=fdmt_tunable_c32_1:SLR0

slr=krnl_sync_stream_uv_4cu_1:SLR0

slr=krnl_grid_4cu_1:SLR0
slr=krnl_grid_4cu_2:SLR0
slr=krnl_grid_4cu_3:SLR0
slr=krnl_grid_4cu_4:SLR0

slr=fft2d_1:SLR1
slr=fft2d_2:SLR1
slr=fft2d_3:SLR1
slr=fft2d_4:SLR1

slr=krnl_boxc_4cu_1:SLR2
```

|0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15| |16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31|

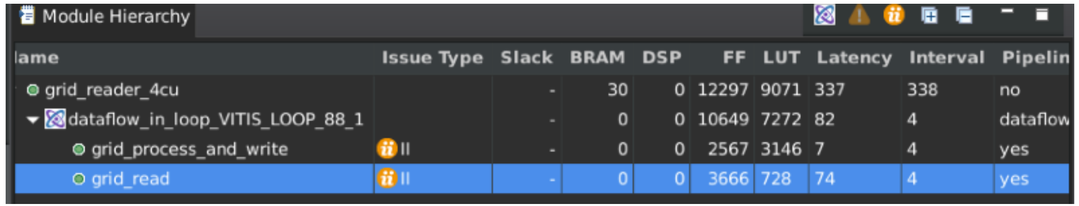
Debugging waveforms





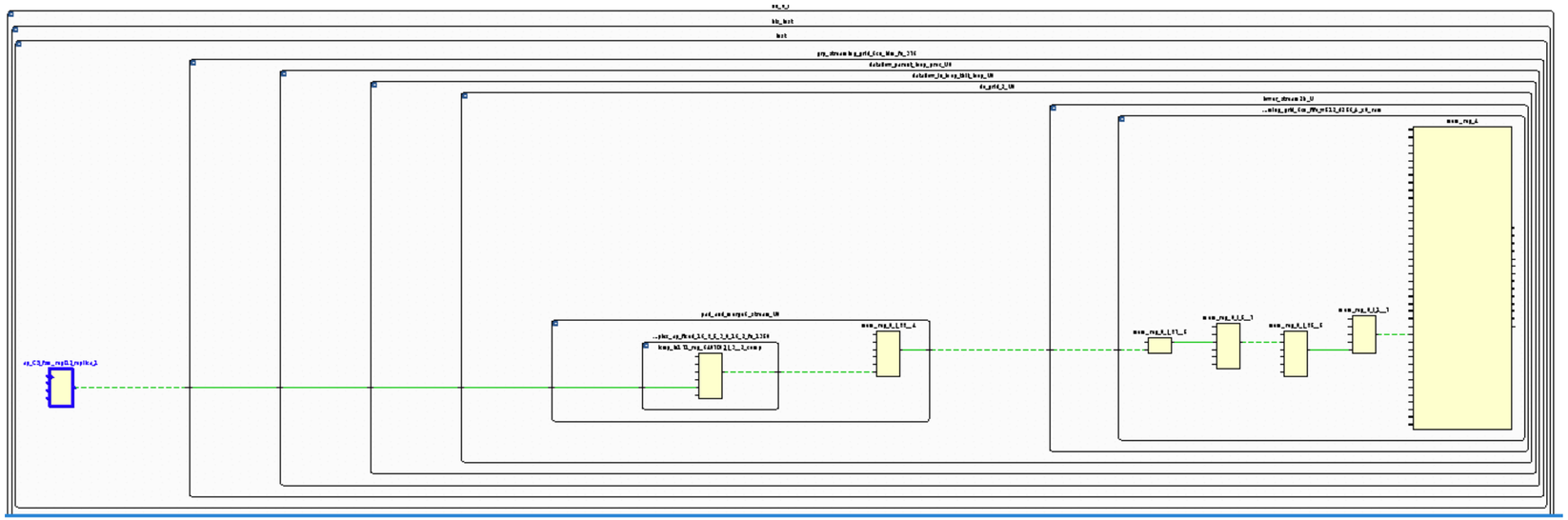
in hardware?

- See git 8b4136a3c6b501384f72a622b251b4692820f234



Trying to meet timing of grid kernel

Control logic in large Vitis kernels is slowing design. Might change to smaller kernels



Path 1 - bd_0_wrapper_timing_summary_routed | Schematic | Schematic

Summary

Name	Path 1
Slack	-0.855ns
Source	bd_0_i/hls_inst/inst/ap_CS_fsm_reg[1]_replica_1/C (rising edge-
Destination	bd_0_i/hls_inst/inst/grp_streaming_grid_6cu_idm_fu_376/dataflo
Path Group	ap_clk
Path Type	Setup (Max at Slow Process Corner)
Requirement	2.500ns (ap_clk rise@2.500ns - ap_clk rise@0.000ns)
Data Path Delay	2.987ns (logic 0.364ns (12.186%) route 2.623ns (87.814%))
Logic Levels	6 (LUT2=1 LUT5=1 LUT6=4)
Clock Path Skew	0.009ns
Clock Un...rtainty	0.035ns

Source Clock Path

Delay Type	Incr (ns)	Path ...	Location	...	Netlist Re
(clock ap_clk rise edge)	(r) 0.000	0.000			
	(r) 0.000	0.000			ap_clk
net (fo=150453, unset)	0.030	0.030			bd_0_i
FDRE			Site: SL...X114Y143	...	bd_0_i

Data Path

Delay Type	Incr (ns)	Path ...	Location	...
FDRE (Prop_HFF_SLICEM_C_Q)	(f) 0.079	0.109	Site: SLI..._X114Y143	...
net (fo=2, routed)	0.758	0.867		
LUT6 (Prop_A6LUT_SLICEM_I4_O)	(r) 0.038	0.905	Site: SLICE_X95Y183	...
net (fo=5, routed)	0.147	1.052		
LUT6 (Prop_C6LUT_SLICEM_I5_O)	(f) 0.037	1.089	Site: SLICE_X95Y187	...
net (fo=11, routed)	0.453	1.542		
LUT2 (Prop_F6LUT_SLICEL_I1_O)	(r) 0.035	1.577	Site: SLI..._X112Y188	...
net (fo=1, routed)	0.039	1.616		
LUT6 (Prop_H6LUT_SLICEL_I2_O)	(r) 0.036	1.652	Site: SLI..._X112Y188	...
net (fo=10, routed)	0.237	1.889		
LUT6 (Prop_A6LUT_SLICEL_I1_O)	(r) 0.050	1.939	Site: SLI..._X113Y190	...
net (fo=1, routed)	0.141	2.080		
LUT5 (Prop_B6LUT_SLICEM_I4_O)	(r) 0.089	2.169	Site: SLI..._X114Y192	...
net (fo=9, routed)	0.848	3.017		
RAMB36E2			Site: RAMB36_X8Y39	...
Arrival Time		3.017		

Example code - and hardware block

```
template <int iterno>
void fdmt_process_iteration_ctloop(
        fdmt_stream& in_stream,
        fdmt_stream& out_stream,
        fdmt_config_entry_t config_table[NCIN/2],
        FdmtRingFifos<iterno>& fifos)
{
    constexpr int ncout = MyConfig::nchan_out_for_iter(iterno);
    // Need to mark these stable so do_iteration() can overlap with stream_into_buffer()
#pragma HLS STABLE variable=config_table
#pragma HLS STABLE variable=fifos

    ct_loop:
    for(int ct = 0; ct < ncout*NT; ct++) {
#pragma HLS DATAFLOW // DATAFLOW with PIPO

        fdmt_complex_t buffer[2][MyConfig::ndm_in_for_iter(iterno)];
#pragma HLS STREAM variable=buffer depth=2 off
        fdmt_process_stream_into_buffer<iterno>(buffer, in_stream);
        fdmt_process_do_iteration<iterno>(ct,
                buffer,
                out_stream,
                config_table,
                fifos);
    }
}
```

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	fdmt_process_do_iteration<0>	return value
ap_rst	in	1	ap_ctrl_hs	fdmt_process_do_iteration<0>	return value
ap_start	in	1	ap_ctrl_hs	fdmt_process_do_iteration<0>	return value
ap_done	out	1	ap_ctrl_hs	fdmt_process_do_iteration<0>	return value
ap_continue	in	1	ap_ctrl_hs	fdmt_process_do_iteration<0>	return value
ap_idle	out	1	ap_ctrl_hs	fdmt_process_do_iteration<0>	return value
ap_ready	out	1	ap_ctrl_hs	fdmt_process_do_iteration<0>	return value
s1_V_din	out	32	ap_fifo	s1_V	pointer
s1_V_full_n	in	1	ap_fifo	s1_V	pointer
s1_V_write	out	1	ap_fifo	s1_V	pointer
configs_address0	out	4	ap_stable	configs	array
configs_ce0	out	1	ap_stable	configs	array
configs_q0	in	32	ap_stable	configs	array
ct_dout	in	10	ap_fifo	ct	pointer
ct_empty_n	in	1	ap_fifo	ct	pointer
ct_read	out	1	ap_fifo	ct	pointer
buffer_r_address0	out	3	ap_memory	buffer_r	array
buffer_r_ce0	out	1	ap_memory	buffer_r	array
buffer_r_q0	in	32	ap_memory	buffer_r	array
buffer_r_address1	out	3	ap_memory	buffer_r	array
buffer_r_ce1	out	1	ap_memory	buffer_r	array
buffer_r_q1	in	32	ap_memory	buffer_r	array
fifos0_m_process_buffer_data_V_address0	out	8	ap_memory	fifos0_m_process_buffer_data_V	array
fifos0_m_process_buffer_data_V_ce0	out	1	ap_memory	fifos0_m_process_buffer_data_V	array
fifos0_m_process_buffer_data_V_we0	out	1	ap_memory	fifos0_m_process_buffer_data_V	array
fifos0_m_process_buffer_data_V_d0	out	32	ap_memory	fifos0_m_process_buffer_data_V	array
fifos0_m_process_buffer_data_V_address1	out	8	ap_memory	fifos0_m_process_buffer_data_V	array
fifos0_m_process_buffer_data_V_ce1	out	1	ap_memory	fifos0_m_process_buffer_data_V	array
fifos0_m_process_buffer_data_V_q1	in	32	ap_memory	fifos0_m_process_buffer_data_V	array

FIFOS are marked as STABLE in code

But in the synthesis report the FIFOS are ap_memory (unlike the configuration table)

If I do a different build with a different top function

Synthesis report

General Information

Date: Wed Jun 16 10:20:56 2021
Version: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020)
Project: fdmt_tunable
Solution: solution1 (Vitis Kernel Flow Target)
Product family: virtexuplus
Target device: xcu280-fsvh2892-2L-e

Timing Estimate

Target	Estimated	Uncertainty
2.50 ns	3.557 ns	0.50 ns

Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
fdmt_tunable_c32		-1.56	-	-	-	-	-	dataflow	54	8	23128	35792	8
fdmt_run_with_config		-1.56	-	-	-	-	-	dataflow	4	8	18263	29441	8
fdmt_process_nbank		-	-	-	-	-	-	no	4	8	15035	25963	8
fdmt_read		-	6217	1.554e4	-	6217	-	no	0	0	1132	989	0
fdmt_write	Timing Violation	-1.56	6218	2.212e4	-	6218	-	no	0	0	521	777	0
fdmt_read_history		-	8217	2.054e4	-	8217	-	no	0	0	358	601	0
fdmt_write_history		-	8729	2.182e4	-	8729	-	no	0	0	386	387	0
fdmt_run_with_config_entry290		-	0	0.0	-	0	-	no	0	0	3	110	0
fdmt_get_config_from_hbm296		-	320	800.000	-	320	-	no	0	0	150	508	0

HW Interfaces

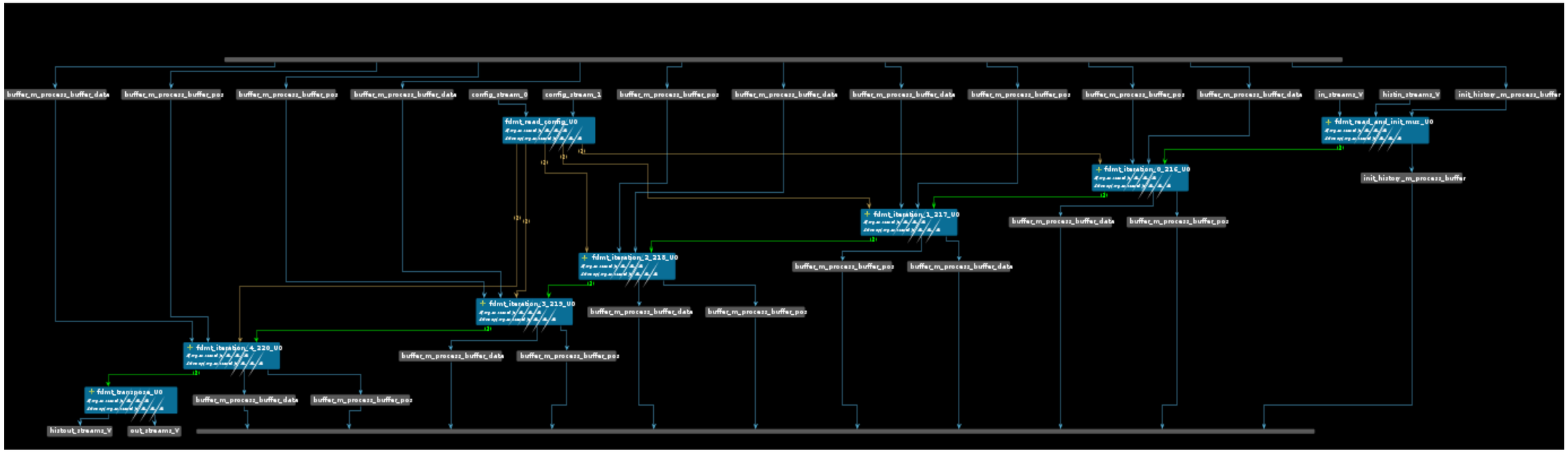
M_AXI

Interface	Data Width (SW->HW)	Address Width	Latency	Offset	Offset Interfaces	Register	Max Widen Bitwidth	Max Read Burst Length	Max Write Burst Length	Num Read Outstanc
m_axi_gmem0	64 -> 512	64	64	slave	s_axi_control	0	512	16	16	
m_axi_gmem1	64 -> 64	64	64	slave	s_axi_control	0	512	16	16	
m_axi_gmem2	64 -> 128	64	64	slave	s_axi_control	0	512	16	16	
m_axi_gmem3	64 -> 64	64	64	slave	s_axi_control	0	512	16	16	
m_axi_gmem4	64 -> 64	64	64	slave	s_axi_control	0	512	16	16	

Depth of modules

Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count
Select to add a body text box.		-1.56	-	-	-	-	-
fdmt_run_with_config		-1.56	-	-	-	-	-
fdmt_process_nbank		-	-	-	-	-	-
process_in_loop		-	-	-	-	-	-
process_r		-	-	-	-	-	-
fdmt_iteration_0_216		-	-	-	-	-	-
fdmt_process_iteration_0_s		-	2379	5.947e3	-	2379	-
fdmt_process_iteration_tloop_allchan_0_s		-	2378	5.945e3	-	2378	-
dataflow_in_loop_t_loop		-	168	420.000	-	96	-
fdmt_process_do_iteration_allchan_0_s		-	70	175.000	-	64	-
shift_4		-	0	0.0	-	1	-
cout_loop_d_loop		-	69	172.000	7	1	64
fdmt_process_stream_into_buffer_allchan_0_s		-	97	242.000	-	96	-
c_loop_d_loop		-	96	240.000	2	1	96
t_loop		-	2377	5.942e3	2377	-	24
fdmt_load_fifos_0_s		-	-	-	-	-	-
fdmt_iteration_2_218		-	-	-	-	-	-
fdmt_iteration_3_219		-	-	-	-	-	-
fdmt_iteration_4_220		-	-	-	-	-	-
fdmt_iteration_1_217		-	-	-	-	-	-
fdmt_read_and_init_mux		-	3370	8.425e3	-	3370	-
fdmt_transpose		-	1022	2.555e3	-	1022	-
fdmt_read_config		-	96	240.000	-	96	-
urest_loop		-	-	-	-	-	8
fdmt_read		-	6217	1.554e4	-	6217	-
urest_loop_t_loop_c_loop		-	6145	1.536e4	3	1	6144
fdmt_write	Timing Violation	-1.56	6218	2.212e4	-	6218	-
fdmt_write_ntout		-	74	185.000	-	3	-
urest_loop_tblk_loop_d_loop	Timing Violation	-	6216	2.211e4	76	3	2048
fdmt_read_history		-	8217	2.054e4	-	8217	-
urest_loop_h_loop		-	8145	2.036e4	3	1	8144
fdmt_write_history		-	8729	2.182e4	-	8729	-
fdmt_run_with_config_entry290		-	0	0.0	-	0	-
fdmt_get_config_from_hbm296		-	320	800.000	-	320	-

Dataflow View



Design rationale

Imaging pipeline

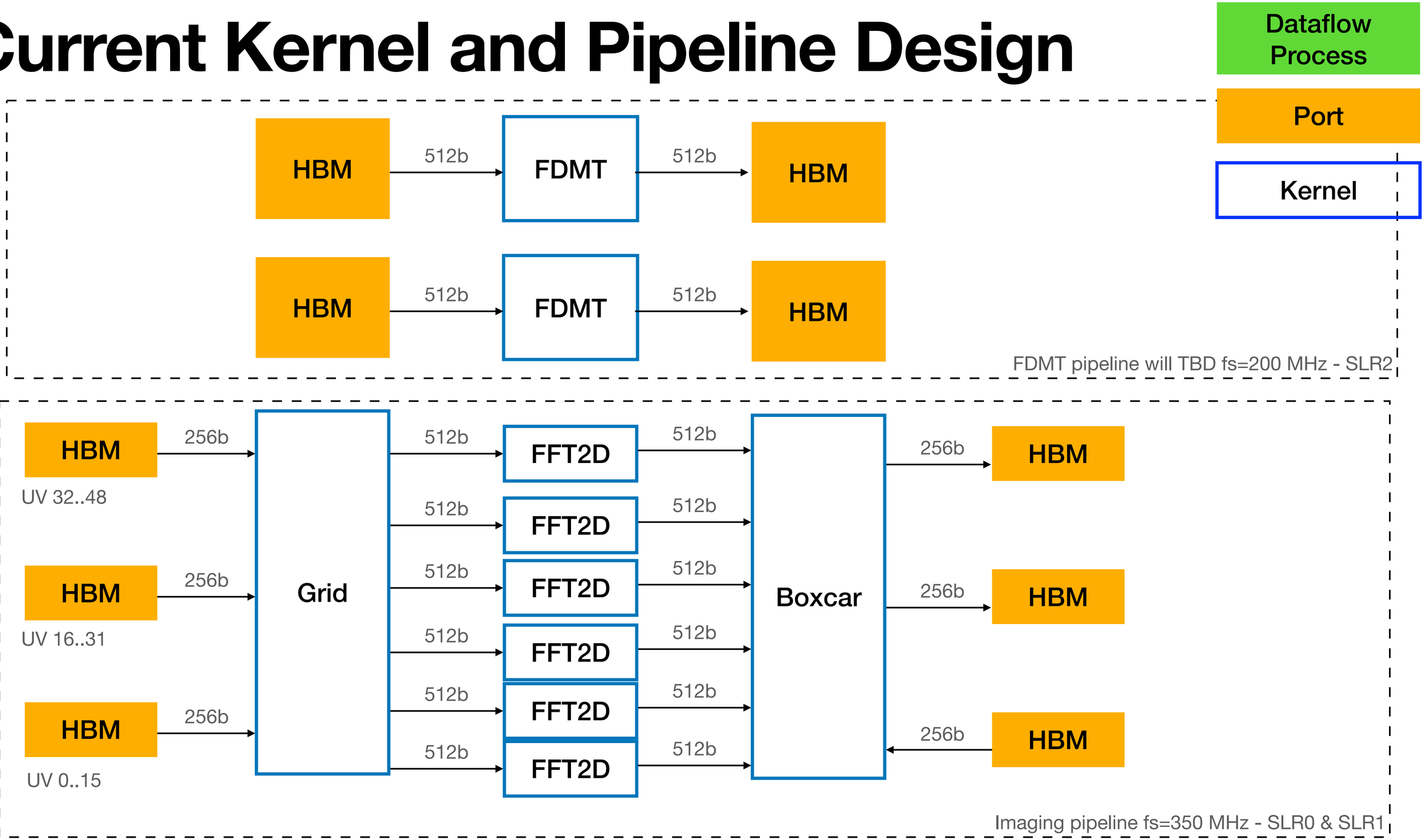
- Original goal: 1 Million FFTs/second
- Xilinx-supplied FFTs
- Block floating point FFT2D (March 2020) SSR=8.
- Fixed-point FFT2D: (Oct 2020)
  - Fmax=**400** MHz, **SSR**=16, **16%/SLR LUTS**
- Requirement can be achieved with 6 CU @ 350 MHz.
- At 3 CU/SLR, it leaves 50% of the SLR0 and SLR1 for grid/boxcar kernels.
- FDMT is by itself on SLR2

FFT Compute units required to achieve 1 M FFTs/sec

NCU	LUTs (% of SLR)	Fmax at 100% efficiency (MHz)
1	16	2048
2	32	1024
3	48	682
4	64	512
5	80	410
6	96	341
8	128	256
10	160	204

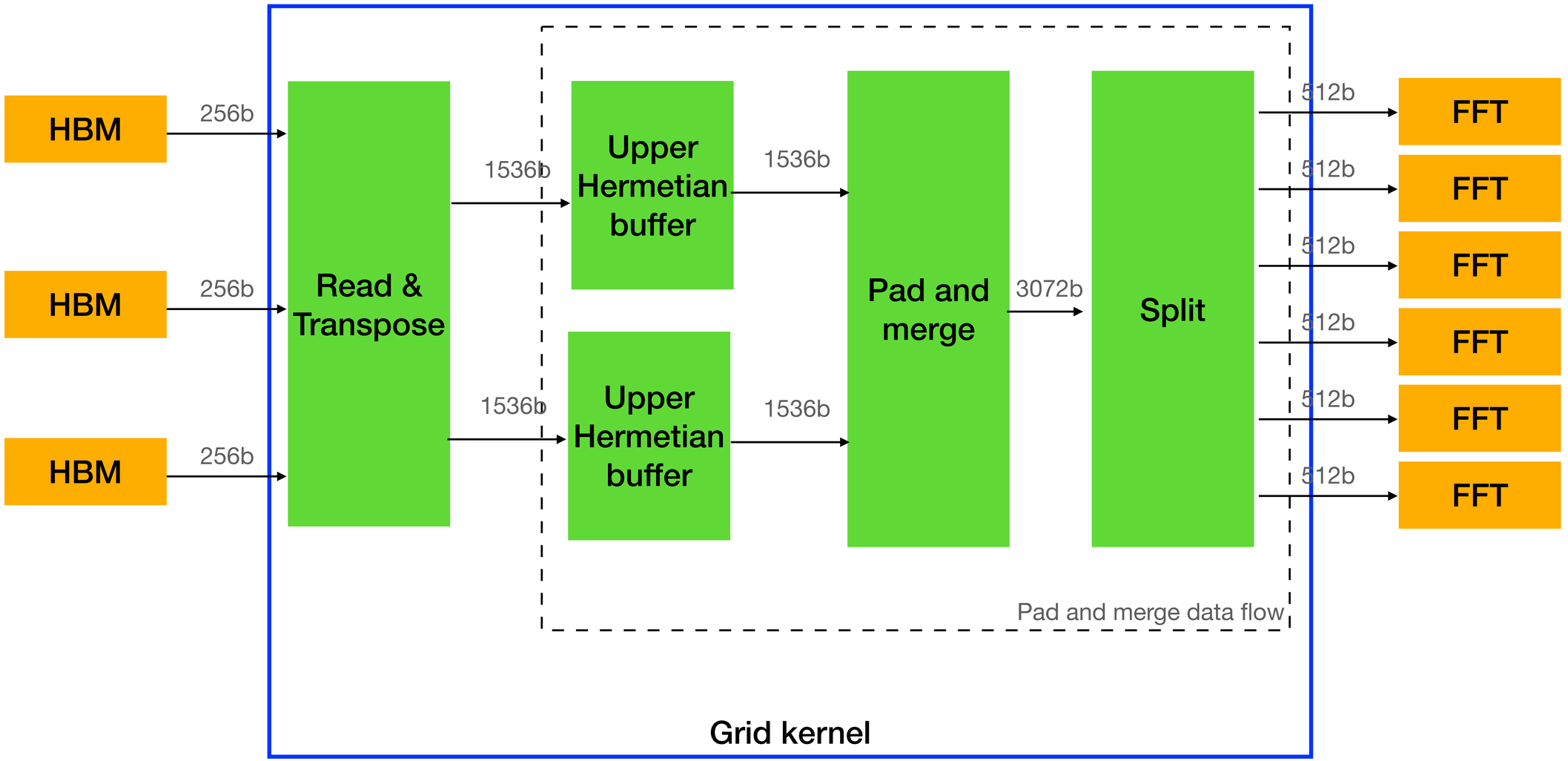
More block diagrams we tried

Current Kernel and Pipeline Design

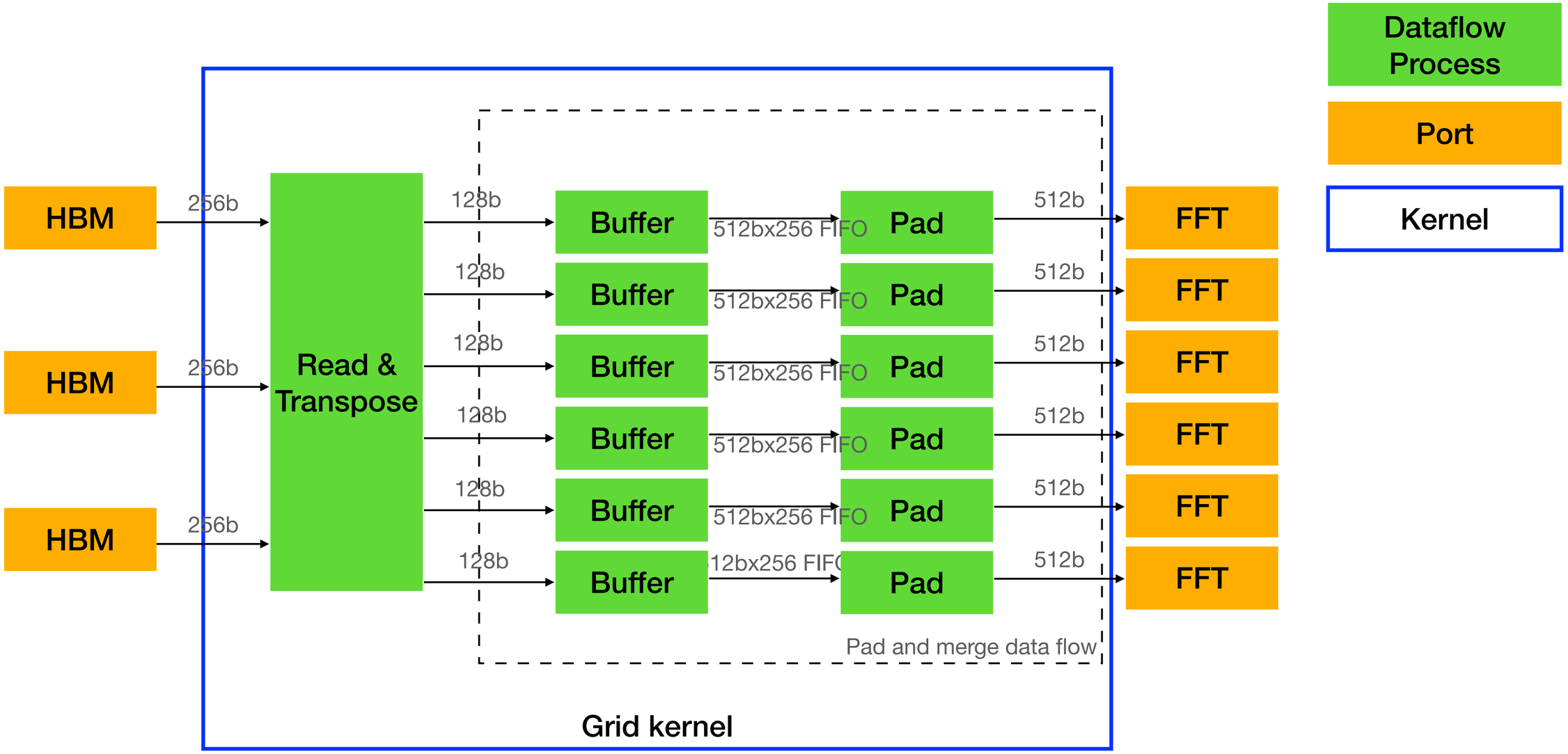


Original grid - slow (250 MHz)

Unroll the NFFT in the data path. Makes very wide busses. (Looking at it now, I'm a bit embarrassed we tried this)



Unroll NFFT as 6 independent Dataflow processes. They are independent after the buffers. fs=340 MHz Speed now limited by the control path - (i.e. ap_start from he whole kernel needs to get to all the buffers in 1 clk). Produces all FFT inputs in one kernel.



Proposed: Subdivided grid kernel - improved timing through duplicating control path?

Have one read & transpose kernel that feeds 6 buffer and pad kernels.
The tricky bit is you have to split the metadata too. But maybe this is a faster way to do things than a big data flow.