FPGA Acceleration of the LFRic Weather and Climate Model in the EuroExa Project Using Vivado HLS

Mike Ashworth, Graham Riley, Andrew Attwood and John Mawer
Advanced Processor Technologies Group
School of Computer Science,
University of Manchester, United Kingdom
mike.ashworth.compsci@manchester.ac.uk

Project outline

Horizon 2020 FETHPC-01-2016:
Co-design of HPC systems and applications
EuroExa started 1st Sep 2017, runs for 3½ years
16 Partners, 8 countries, €20M
Builds on previous projects, esp. ExaNoDe, ExaNeSt, EcoScale

Aim: design, build, test and evaluate an Exascale prototype system
Architecture based on ARM CPUs with FPGA accelerators
Three testbed systems: #3 >100 Pflop/s
Low-power design goal to target realistic Exascale system
Architecture evolves in response to application requirements = co-design
Wide range of apps, incl. weather forecasting, lattice Boltzmann, multiphysics, astrophysics, astronomy data processing, quantum chemistry, life sciences and bioinformatics

@euroexa

euroexa.eu

Kick-off meeting 4th-5th Sep 2017, Barcelona

Motivation

- FPGAs offer large (OsOM) gains in performance/W
- Also gains in performance/{$£€฿}
- Major corporations are using FPGAs in datacentres for cloud services, analytics, communication, etc.
- H/W traditionally led by Xilinx (ARM CPU + FPGA single chip)
- Intel’s acquisition of Altera led to Heterogeneous Architecture Research Platform (HARP) (also single chip)
- Predictions: up to 30% of datacenter servers will have FPGAs by 2020

LFRic Weather and Climate Model

Brand new weather and climate model: LFRic

named after Lewis Fry Richardson (1881-1953)

- Dynamics from the GungHo project 2011-2015
- Scalability – globally uniform grid (no poles)
- Speed – maintain performance at high & low resolution and for high & low core counts
- Accuracy – need to maintain standing of the model
- Separation of Concerns – PSyClone generated layer for automated targeting of architectures
- Operational weather forecasts around 2022 – anniversary of Richardson (1922)

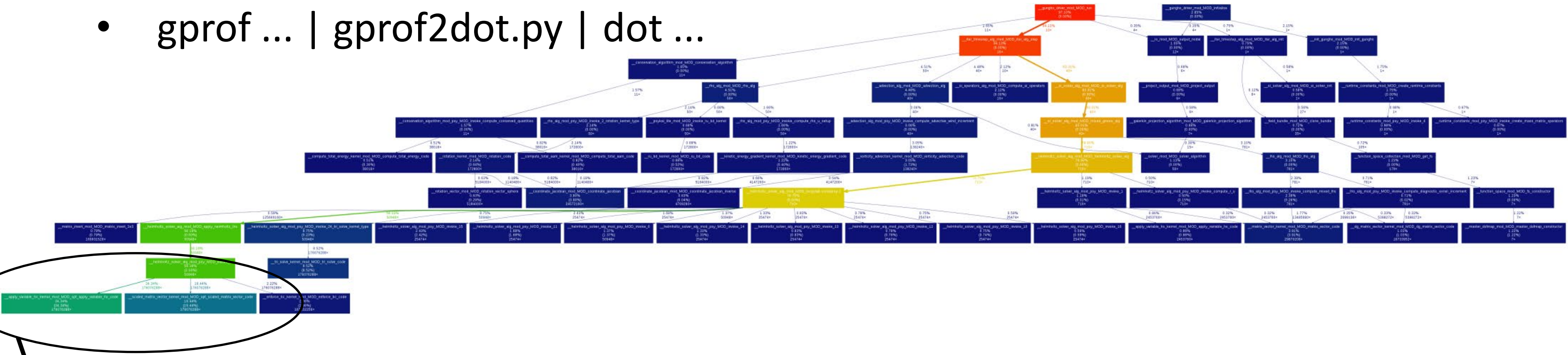
Globally
Uniform
Next
Generation
Highly
Optimized

工合

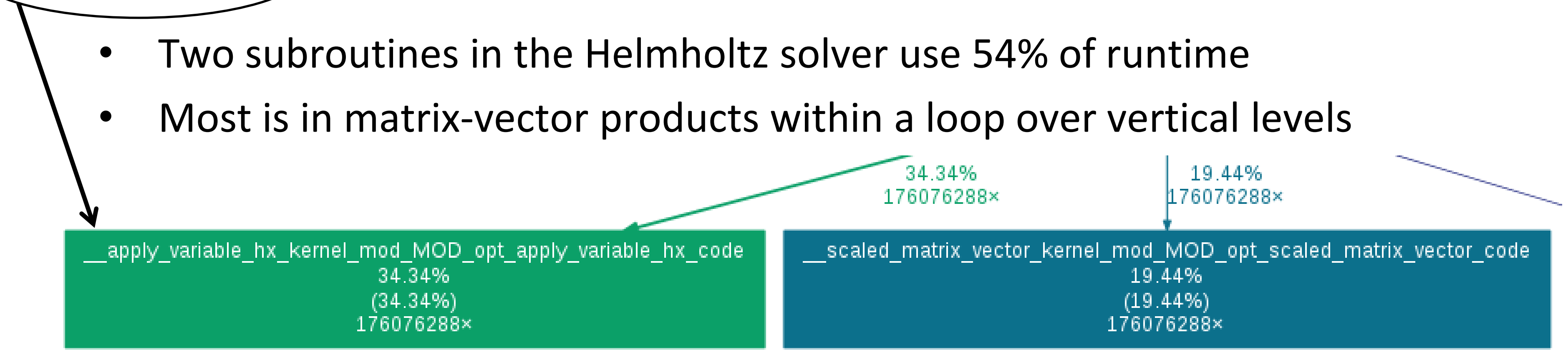
"Working together harmoniously"

LFRic profile & call graph

- Baroclinic performance benchmark case
- gprof ... | gprof2dot.py | dot ...



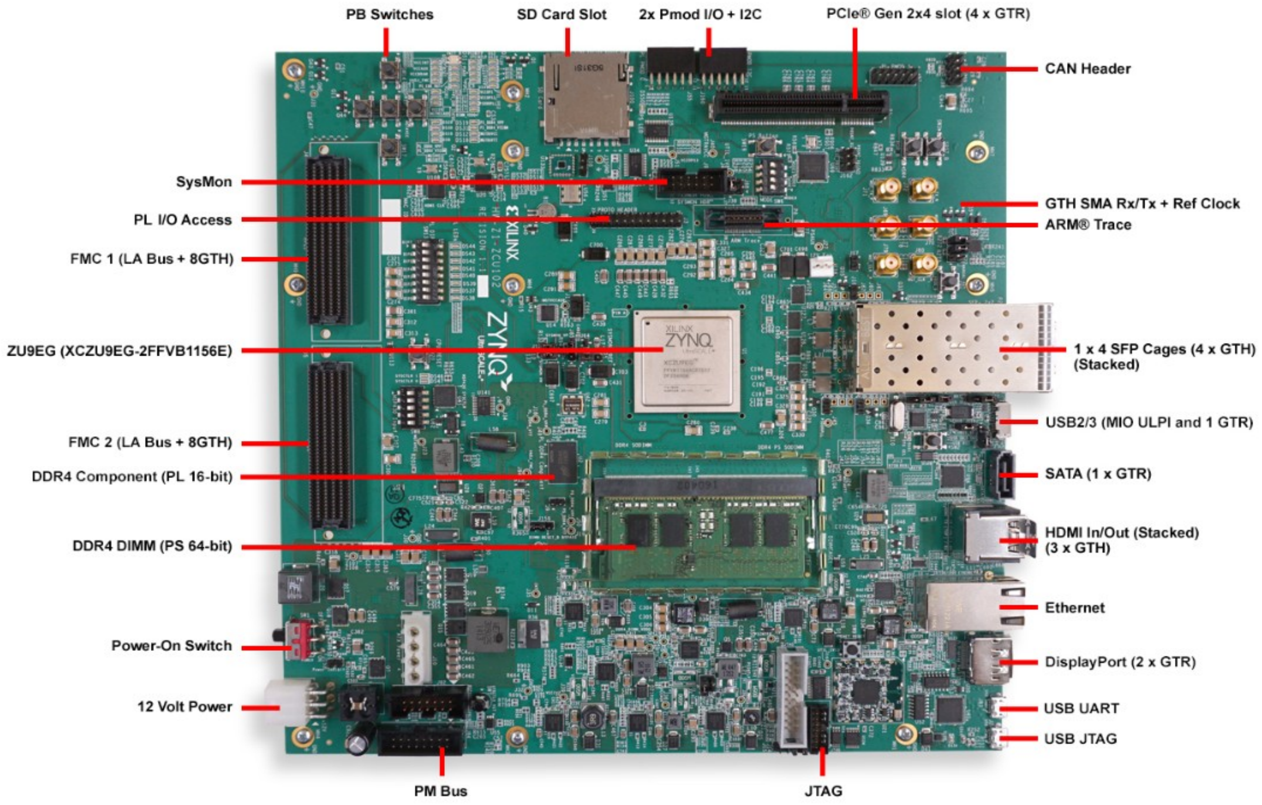
- Two subroutines in the Helmholtz solver use 54% of runtime
- Most is in matrix-vector products within a loop over vertical levels


34.34%
176076288×
19.44%
176076288×
__apply_variable_hx_kernel_mod_MOD_opt_apply_variable_hx_code
34.34%
(34.34%)
176076288×
__scaled_matrix_vector_kernel_mod_MOD_opt_scaled_matrix_vector_code
19.44%
(19.44%)
176076288×

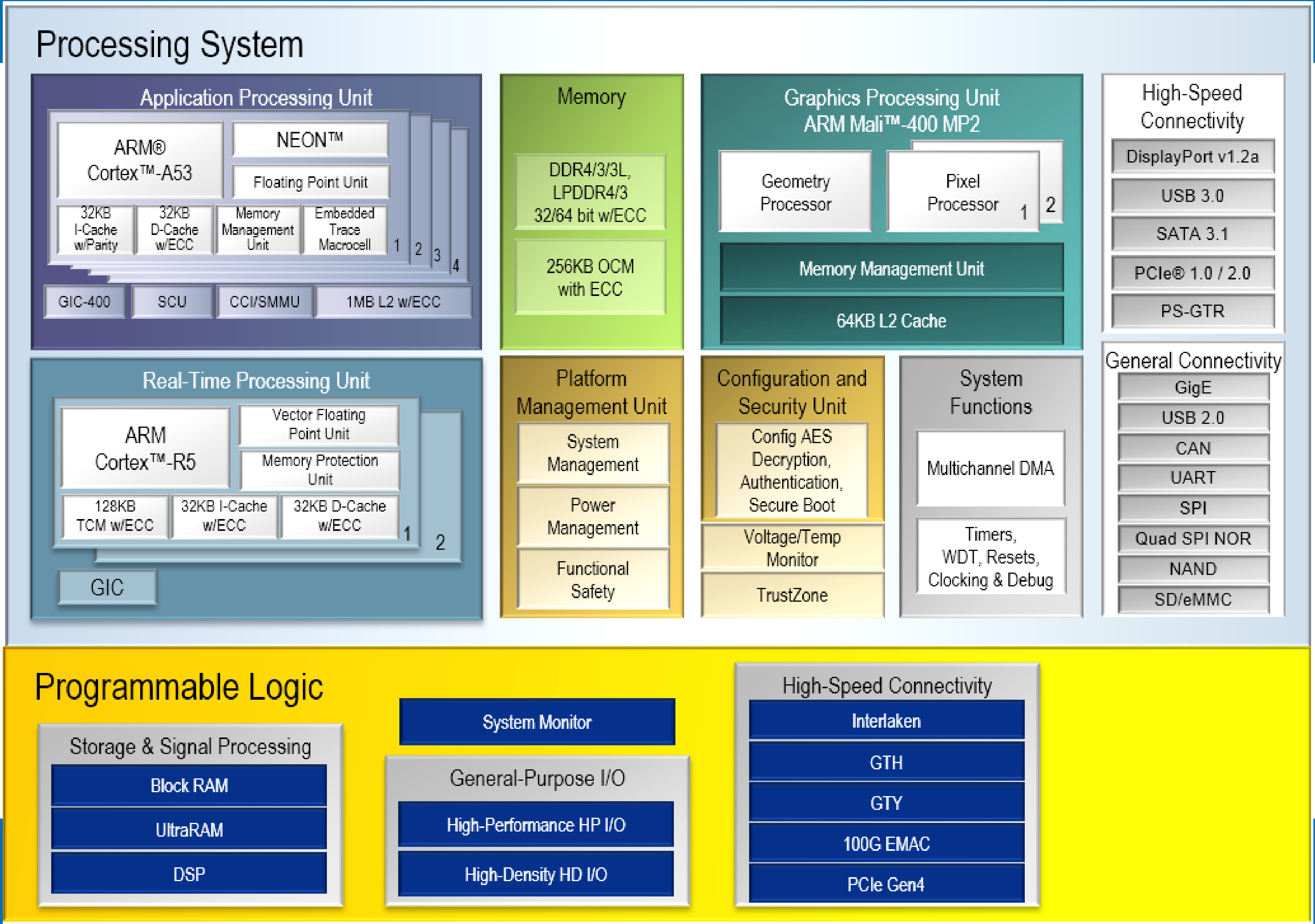
Zynq UltraScale+ ZCU102 Evaluation Platform

- ARM Cortex A53 quad-core CPU 1.2 GHz
- Dual-core Cortex-R5 real-time processor
- Mali-400 MP2 GPU
- Zynq UltraScale XCZU9EG-2FFVB1156 FPGA

System Logic Cells (K)	600
Memory (Mb)	32.1
DSP Slices	2,520
Maximum I/0 Pins	328



Zynq UltraScale+ MPSoC EG



Approach

- LFRic
  - Full application (Fortran)
  - Compact applications “mini-apps”
  - Kernels e.g. matrix-vector product (C)
- Performance on ARM CPU quad-core A53
- Use Vivado HLS to generate IP blocks for key kernels to run on the UltraScale+ FPGA
- Investigate performance optimizations

Range of Programming Models

1. C code with Xilinx Vivado HLS and Vivado Design Suite
2. OmpSs@FPGA directive-based (BSC)
3. MaxJ compiler for Maxeler systems
4. OpenCL code with Xilinx SDSoC
5. OpenStream (Uni Man)

- Options 2-5 being investigated by other members of the project

Starting code for Vivado HLS

```
#define NDF1 8
#define NDF2 6
#define NK 40
#define MVTYPE double
int matvec_8x6x40_vanilla (MVTYPE matrix[NK][NDF2][NDF1],
              MVTYPE x[NDF2][NK], MVTYPE lhs[NDF1][NK]) {
  int df,j,k;
  for (k=0;k<NK;k++) {
    for (df=0;df<NDF1;df++) {
      lhs[df][k] = 0.0;
      for (j=0;j<NDF2;j++) {
        lhs[df][k] = lhs[df][k]
          + x[j][k]*matrix[k][j][df];
      }
    }
  }
  return 0;
}
```

Notes:

- Data sizes hard-wired for HLS
- Vertical loop k is outer
- Vectors x and lhs are sequential in k (k-last in C)
- Matrix is not (k-first)
- Read-then-write dependence on lhs
- Flops = 2*NK*NDF1*NDF2 = 3840
- Mem refs = 2*flops = 7680 doubles

Optimizations in Vivado HLS

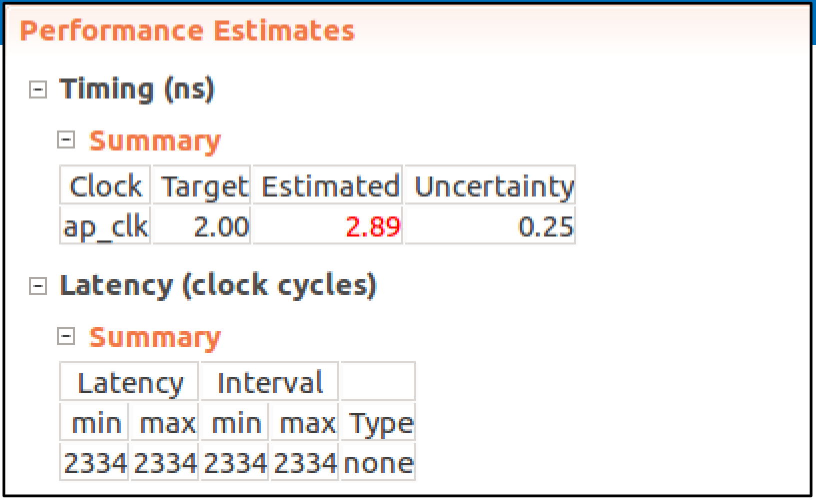
- Make k the inner loop (loop length 40, independent, sequential access)
- Transpose matrix to k-last to ensure sequential memory access
- HLS pragma to unroll inner loops on k (no benefit from hand unrolling)
- HLS pragma to pipeline outer loop on df
- HLS pragma for input and output arguments including
  - num_read_outstanding=8
  - max_read_burst_length=64
- Access input and output arguments by memcpy to local arrays to ensure streaming of loads/stores to/from BRAM (see later)

Optimized code in Vivado HLS

```
#pragma HLS INTERFACE m_axi depth=128
  port=matrix offset=slave bundle=bram /
        num_read_outstanding=8 /
        num_write_outstanding=8 /
        max_read_burst_length=64 /
        max_write_burst_length=64
< pragmas for m_axi interfaces for x, lhs
  and s_axilite interface for return>

  int df,j,k;
  MVTYPE ml[NDF2][NK], xl[NDF2][NK],
  ll[NDF1][NK];
  memcpy (xl, x, NDF2*NK*sizeof(MVTYPE));
  for (df=0;df<NDF1;df++) {
#pragma HLS PIPELINE
     for (k=0;k<NK;k++) {
#pragma HLS UNROLL
        ll[df][k] = 0.0;
     }
     memcpy (ml, matrix+df*NDF2*NK, /
        NDF2*NK*sizeof(MVTYPE));
     for (j=0;j<NDF2;j++) {
        for (k=0;k<NK;k++) {
#pragma HLS UNROLL
            ll[df][k] = ll[df][k]+
  xl[j][k]*ml[j][k];
        }
     }
  }
  memcpy (lhs, ll,
  NDF1*NK*sizeof(MVTYPE));
```

Vivado HLS Performance Estimate



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	2.00	2.89	0.25

Latency (clock cycles)

Summary

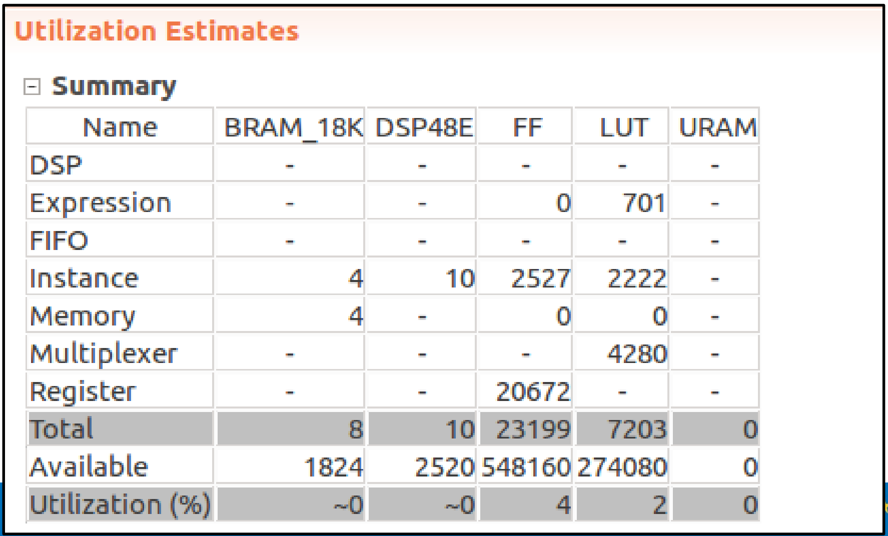
Latency		Interval		
min	max	min	max	Type
2334	2334	2334	2334	none

Utilization Estimate:

- Try to maximize performance while minimizing utilization
- Shows percentage of chip 'real-estate being utilized

Performance Estimate:

- Target 2ns clock: design validated at 2.89ns = 346 MHz
- 2334 cycles for 3840 flops = 1.65 flops/cycle
- Overlapped dmul with dadd
- Starting code was 69841 cycles

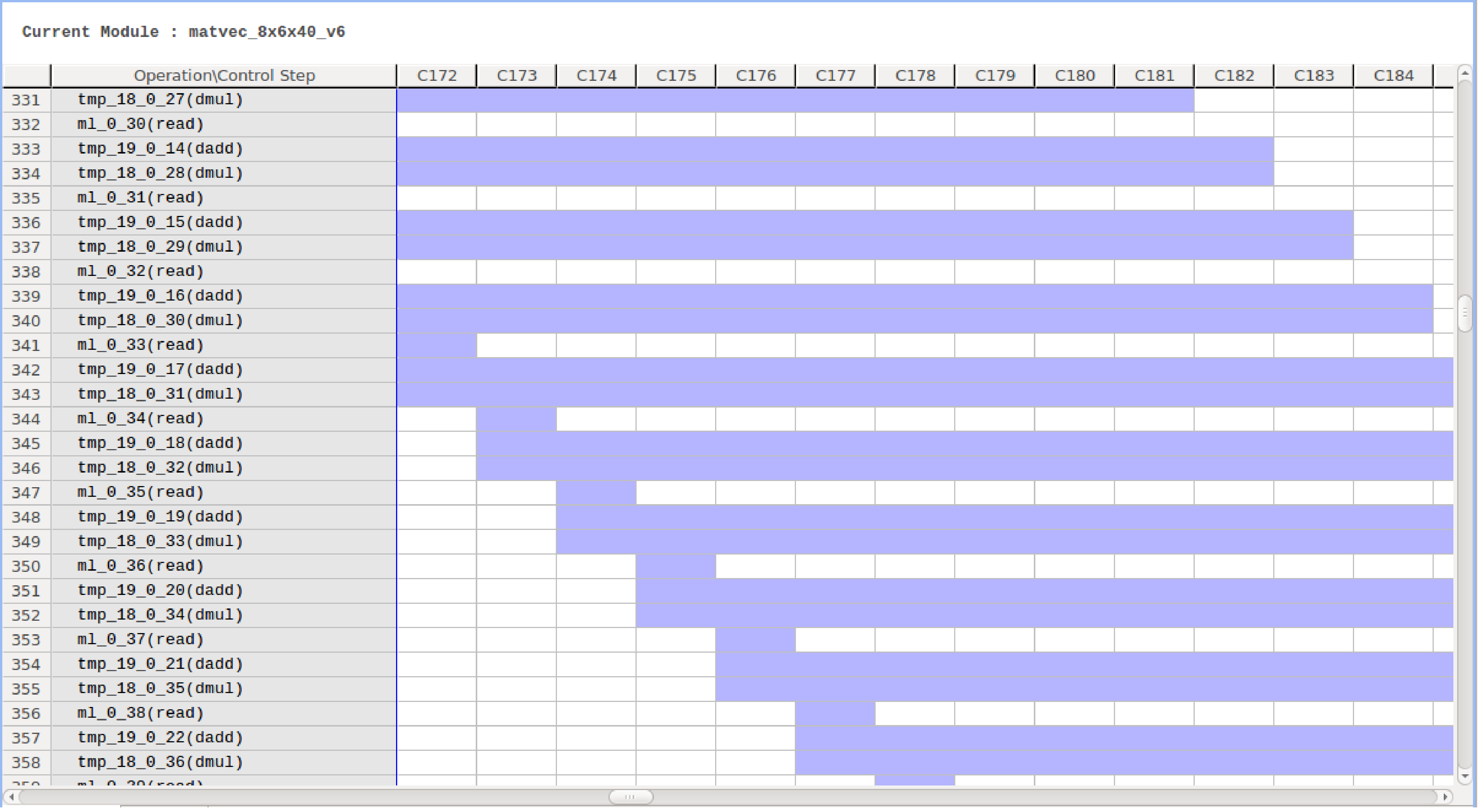


Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	701	-
FIFO	-	-	-	-	-
Instance	4	10	2527	2222	-
Memory	4	-	0	0	-
Multiplexer	-	-	-	4280	-
Register	-	-	20672	-	-
Total	8	10	23199	7203	0
Available	1824	2520	548160	274080	0
Utilization (%)	~0	~0	4	2	0

Vivado HLS Performance Timeline



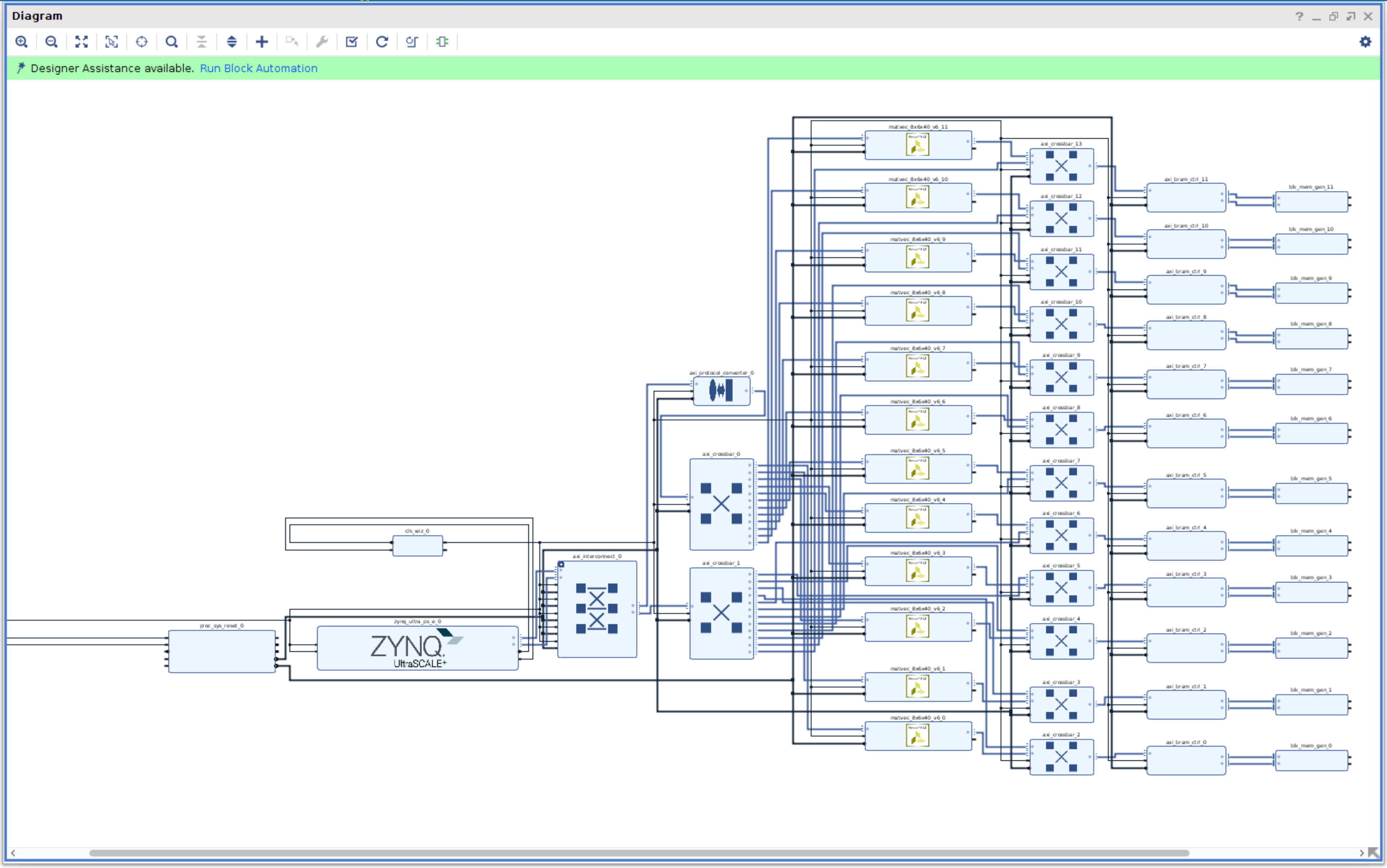
Current Module : matvec_8x6x40_v6

	Operation\Control Step	C172	C173	C174	C175	C176	C177	C178	C179	C180	C181	C182	C183	C184
331	tmp_18_0_27(dmul)	■	■	■	■	■	■	■	■	■	■			
332	ml_0_30(read)													
333	tmp_19_0_14(dadd)	■	■	■	■	■	■	■	■	■	■	■		
334	tmp_18_0_28(dmul)	■	■	■	■	■	■	■	■	■	■	■		
335	ml_0_31(read)													
336	tmp_19_0_15(dadd)	■	■	■	■	■	■	■	■	■	■	■	■	
337	tmp_18_0_29(dmul)	■	■	■	■	■	■	■	■	■	■	■	■	
338	ml_0_32(read)													
339	tmp_19_0_16(dadd)	■	■	■	■	■	■	■	■	■	■	■	■	■
340	tmp_18_0_30(dmul)	■	■	■	■	■	■	■	■	■	■	■	■	■
341	ml_0_33(read)	■												
342	tmp_19_0_17(dadd)	■	■	■	■	■	■	■	■	■	■	■	■	■
343	tmp_18_0_31(dmul)	■	■	■	■	■	■	■	■	■	■	■	■	■
344	ml_0_34(read)		■											
345	tmp_19_0_18(dadd)		■	■	■	■	■	■	■	■	■	■	■	■
346	tmp_18_0_32(dmul)		■	■	■	■	■	■	■	■	■	■	■	■
347	ml_0_35(read)			■										
348	tmp_19_0_19(dadd)			■	■	■	■	■	■	■	■	■	■	■
349	tmp_18_0_33(dmul)			■	■	■	■	■	■	■	■	■	■	■
350	ml_0_36(read)				■									
351	tmp_19_0_20(dadd)				■	■	■	■	■	■	■	■	■	■
352	tmp_18_0_34(dmul)				■	■	■	■	■	■	■	■	■	■
353	ml_0_37(read)					■								
354	tmp_19_0_21(dadd)					■	■	■	■	■	■	■	■	■
355	tmp_18_0_35(dmul)					■	■	■	■	■	■	■	■	■
356	ml_0_38(read)						■							
357	tmp_19_0_22(dadd)						■	■	■	■	■	■	■	■
358	tmp_18_0_36(dmul)						■	■	■	■	■	■	■	■

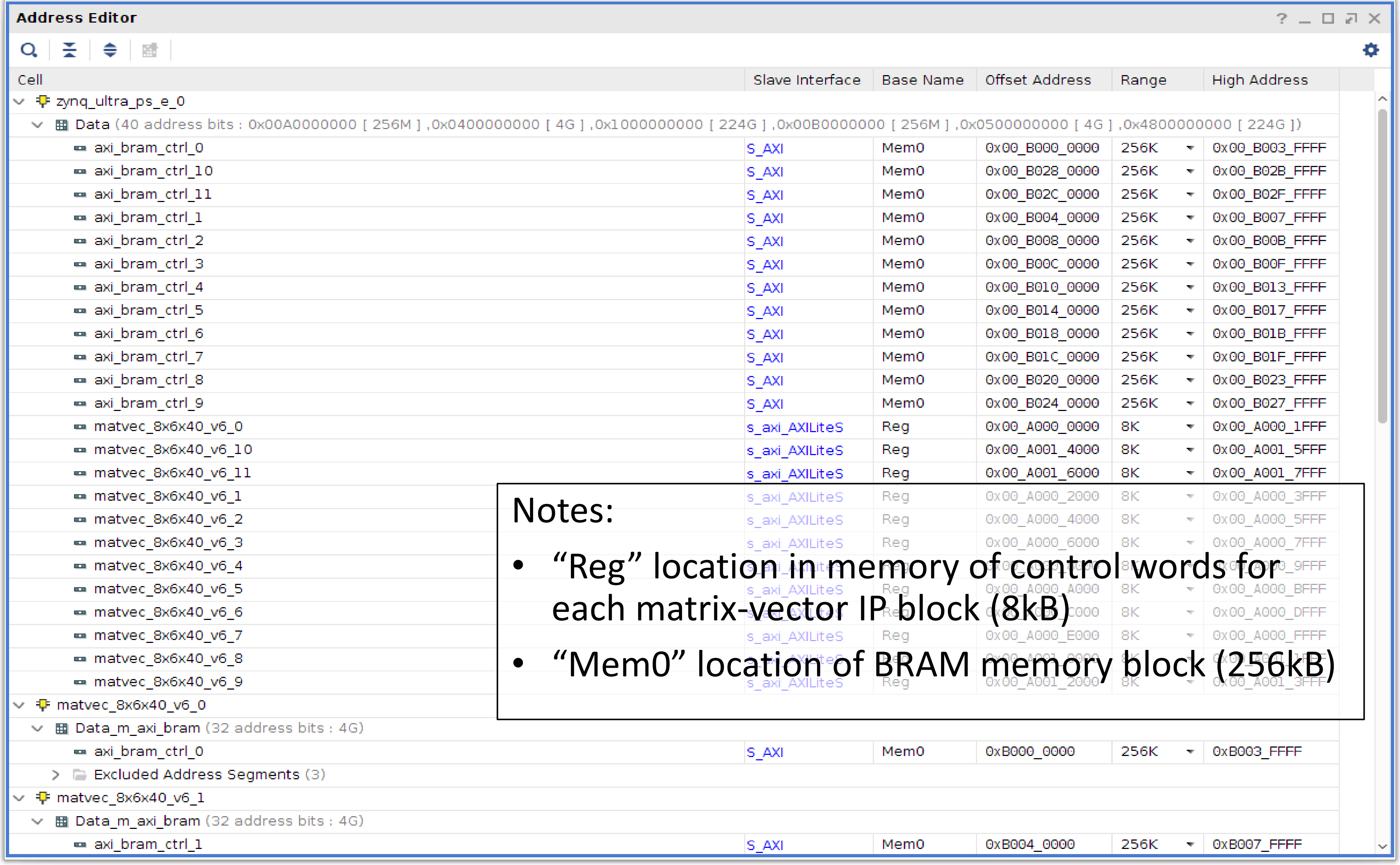
Vivado Design Studio

- Design built from the following IP blocks:
  - 12x matrix-vector multiply IP blocks from HLS
  - 12x Block Memory Generators for BRAM memory
  - 12x AXI BRAM Controller
  - 14x AXI Crossbar blocks for switching
  - 1x AXI Interconnect blocks for switching
  - 1x AXI Protocol Converter
  - 1x ZynQ PS block for interface with ARM
  - 1x Clocking Wizard to multiply system clock
  - 1x Processor System Reset
- Connect blocks manually or with automated assistance
- Address mapping manually or automated using TCL scripts
  - Address editor determines size of BRAM blocks (12x 256kB)
- Vivado SDK reveals “register” assignment for HLS blocks

Design with 12 Matrix-Vector Blocks



Vivado DS Address Editor



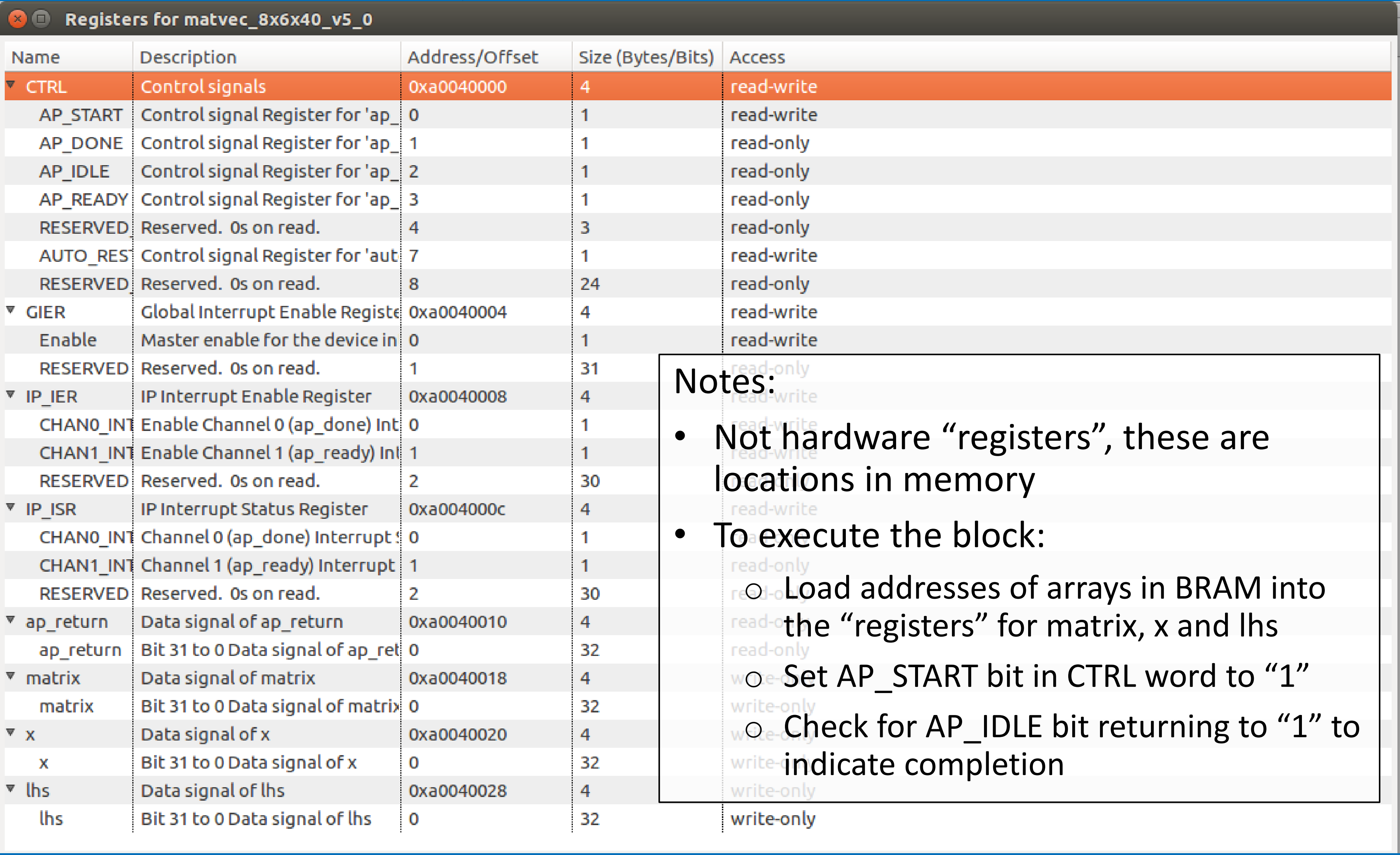
Address Editor

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
zynq_ultra_ps_e_0					
Data (40 address bits : 0x00A0000000 [256M] ,0x0400000000 [4G] ,0x1000000000 [224G] ,0x00B0000000 [256M] ,0x0500000000 [4G] ,0x4800000000 [224G])					
axi_bram_ctrl_0	S_AXI	Mem0	0x00_B000_0000	256K	0x00_B003_FFFF
axi_bram_ctrl_10	S_AXI	Mem0	0x00_B028_0000	256K	0x00_B02B_FFFF
axi_bram_ctrl_11	S_AXI	Mem0	0x00_B02C_0000	256K	0x00_B02F_FFFF
axi_bram_ctrl_1	S_AXI	Mem0	0x00_B004_0000	256K	0x00_B007_FFFF
axi_bram_ctrl_2	S_AXI	Mem0	0x00_B008_0000	256K	0x00_B00B_FFFF
axi_bram_ctrl_3	S_AXI	Mem0	0x00_B00C_0000	256K	0x00_B00F_FFFF
axi_bram_ctrl_4	S_AXI	Mem0	0x00_B010_0000	256K	0x00_B013_FFFF
axi_bram_ctrl_5	S_AXI	Mem0	0x00_B014_0000	256K	0x00_B017_FFFF
axi_bram_ctrl_6	S_AXI	Mem0	0x00_B018_0000	256K	0x00_B01B_FFFF
axi_bram_ctrl_7	S_AXI	Mem0	0x00_B01C_0000	256K	0x00_B01F_FFFF
axi_bram_ctrl_8	S_AXI	Mem0	0x00_B020_0000	256K	0x00_B023_FFFF
axi_bram_ctrl_9	S_AXI	Mem0	0x00_B024_0000	256K	0x00_B027_FFFF
matvec_8x6x40_v6_0	s_axi_AXILiteS	Reg	0x00_A000_0000	8K	0x00_A000_1FFF
matvec_8x6x40_v6_10	s_axi_AXILiteS	Reg	0x00_A001_4000	8K	0x00_A001_5FFF
matvec_8x6x40_v6_11	s_axi_AXILiteS	Reg	0x00_A001_6000	8K	0x00_A001_7FFF
matvec_8x6x40_v6_1	s_axi_AXILiteS	Reg	0x00_A000_2000	8K	0x00_A000_3FFF
matvec_8x6x40_v6_2	s_axi_AXILiteS	Reg	0x00_A000_4000	8K	0x00_A000_5FFF
matvec_8x6x40_v6_3	s_axi_AXILiteS	Reg	0x00_A000_6000	8K	0x00_A000_7FFF
matvec_8x6x40_v6_4	[illegible]	[illegible]	[illegible]	8K	[illegible]
matvec_8x6x40_v6_5	s_axi_AXILiteS	Reg	0x00_A000_A000	8K	0x00_A000_BFFF
matvec_8x6x40_v6_6	[illegible]	[illegible]	[illegible]	8K	0x00_A000_DFFF
matvec_8x6x40_v6_7	s_axi_AXILiteS	Reg	0x00_A000_E000	8K	0x00_A000_FFFF
matvec_8x6x40_v6_8	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
matvec_8x6x40_v6_9	s_axi_AXILiteS	Reg	0x00_A001_2000	8K	0x00_A001_3FFF
matvec_8x6x40_v6_0					
Data_m_axi_bram (32 address bits : 4G)					
axi_bram_ctrl_0	S_AXI	Mem0	0xB000_0000	256K	0xB003_FFFF
Excluded Address Segments (3)					
matvec_8x6x40_v6_1					
Data_m_axi_bram (32 address bits : 4G)					
axi_bram_ctrl_1	S_AXI	Mem0	0xB004_0000	256K	0xB007_FFFF

Notes:

- “Reg” location in memory of control words for each matrix-vector IP block (8kB)
- “Mem0” location of BRAM memory block (256kB)

Vivado SDK Registers



Registers for matvec_8x6x40_v5_0

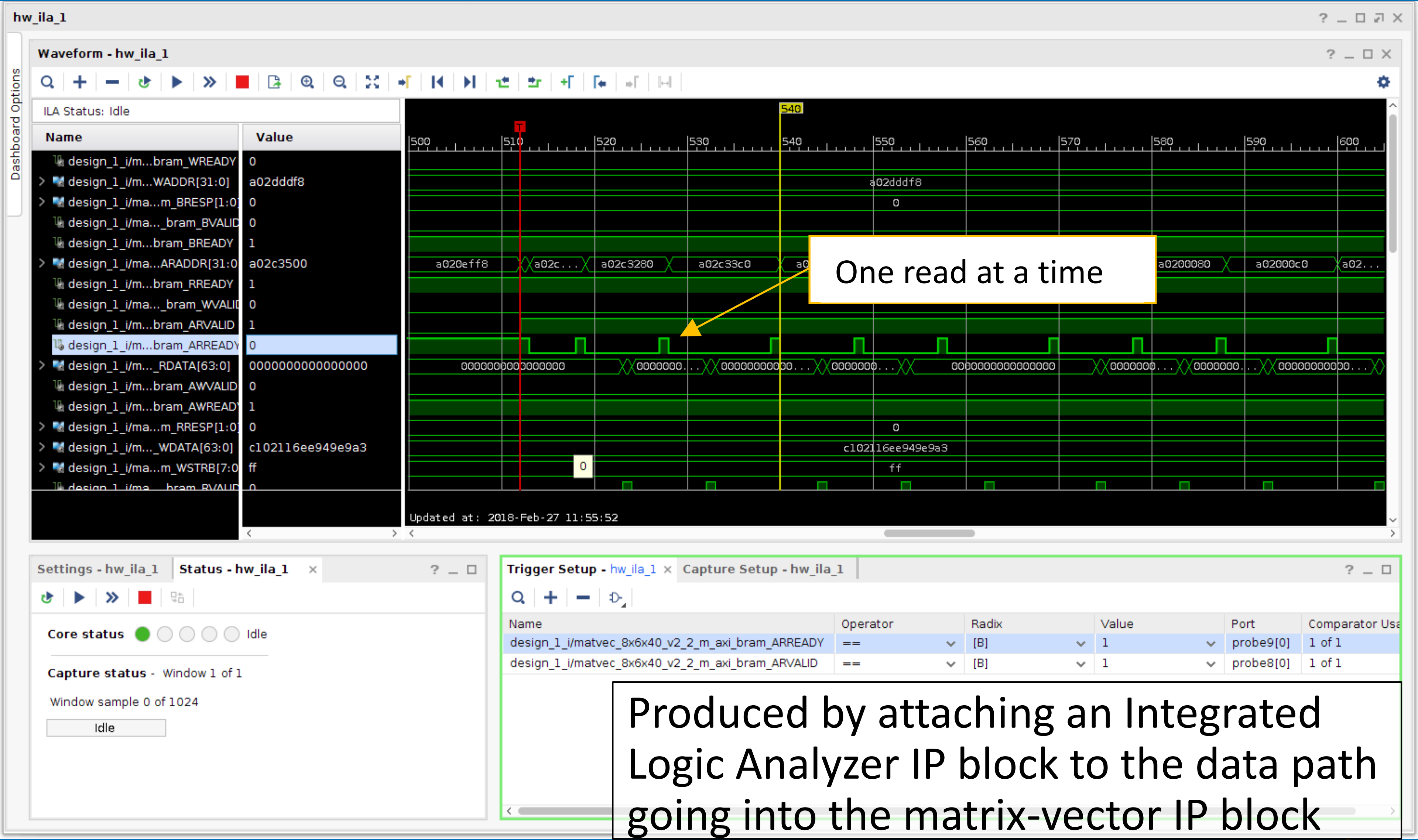
Name	Description	Address/Offset	Size (Bytes/Bits)	Access
CTRL	Control signals	0xa0040000	4	read-write
AP_START	Control signal Register for 'ap_	0	1	read-write
AP_DONE	Control signal Register for 'ap_	1	1	read-only
AP_IDLE	Control signal Register for 'ap_	2	1	read-only
AP_READY	Control signal Register for 'ap_	3	1	read-only
RESERVED	Reserved. 0s on read.	4	3	read-only
AUTO_RES	Control signal Register for 'aut	7	1	read-write
RESERVED	Reserved. 0s on read.	8	24	read-only
GIER	Global Interrupt Enable Registe	0xa0040004	4	read-write
Enable	Master enable for the device in	0	1	read-write
RESERVED	Reserved. 0s on read.	1	31	read-only
IP_IER	IP Interrupt Enable Register	0xa0040008	4	read-write
CHAN0_INT	Enable Channel 0 (ap_done) Int	0	1	read-write
CHAN1_INT	Enable Channel 1 (ap_ready) Int	1	1	read-write
RESERVED	Reserved. 0s on read.	2	30	read-only
IP_ISR	IP Interrupt Status Register	0xa004000c	4	read-write
CHAN0_INT	Channel 0 (ap_done) Interrupt	0	1	read-only
CHAN1_INT	Channel 1 (ap_ready) Interrupt	1	1	read-only
RESERVED	Reserved. 0s on read.	2	30	read-only
ap_return	Data signal of ap_return	0xa0040010	4	read-only
ap_return	Bit 31 to 0 Data signal of ap_ret	0	32	read-only
matrix	Data signal of matrix	0xa0040018	4	write-only
matrix	Bit 31 to 0 Data signal of matrix	0	32	write-only
x	Data signal of x	0xa0040020	4	write-only
x	Bit 31 to 0 Data signal of x	0	32	write-only
lhs	Data signal of lhs	0xa0040028	4	write-only
lhs	Bit 31 to 0 Data signal of lhs	0	32	write-only

Notes:

- Not hardware "registers", these are locations in memory
- To execute the block:
  - Load addresses of arrays in BRAM into the "registers" for matrix, x and lhs
  - Set AP_START bit in CTRL word to "1"
  - Check for AP_IDLE bit returning to "1" to indicate completion

EUROEXA FUNDED BY THE EUROPEAN UNION

Optimized Data Streaming - Before



hw_ila_1

Waveform - hw_ila_1

ILA Status: Idle

Name	Value
design_1_i/m...bram_WREADY	0
design_1_i/m...WADDR[31:0]	a02dddf8
design_1_i/ma...m_BRESP[1:0]	0
design_1_i/ma..._bram_BVALID	0
design_1_i/m...bram_BREADY	1
design_1_i/ma...ARADDR[31:0]	a02c3500
design_1_i/m...bram_RREADY	1
design_1_i/ma..._bram_WVALID	0
design_1_i/m...bram_ARVALID	1
design_1_i/m...bram_ARREADY	0
design_1_i/m..._RDATA[63:0]	0000000000000000
design_1_i/m...bram_AWVALID	0
design_1_i/m...bram_AWREADY	1
design_1_i/ma...m_RRESP[1:0]	0
design_1_i/m..._WDATA[63:0]	c102116ee949e9a3
design_1_i/ma...m_WSTRB[7:0]	ff
design_1_i/ma..._bram_RVALID	0

One read at a time

Updated at: 2018-Feb-27 11:55:52

Settings - hw_ila_1 | Status - hw_ila_1

Core status ● Idle

Capture status - Window 1 of 1

Window sample 0 of 1024

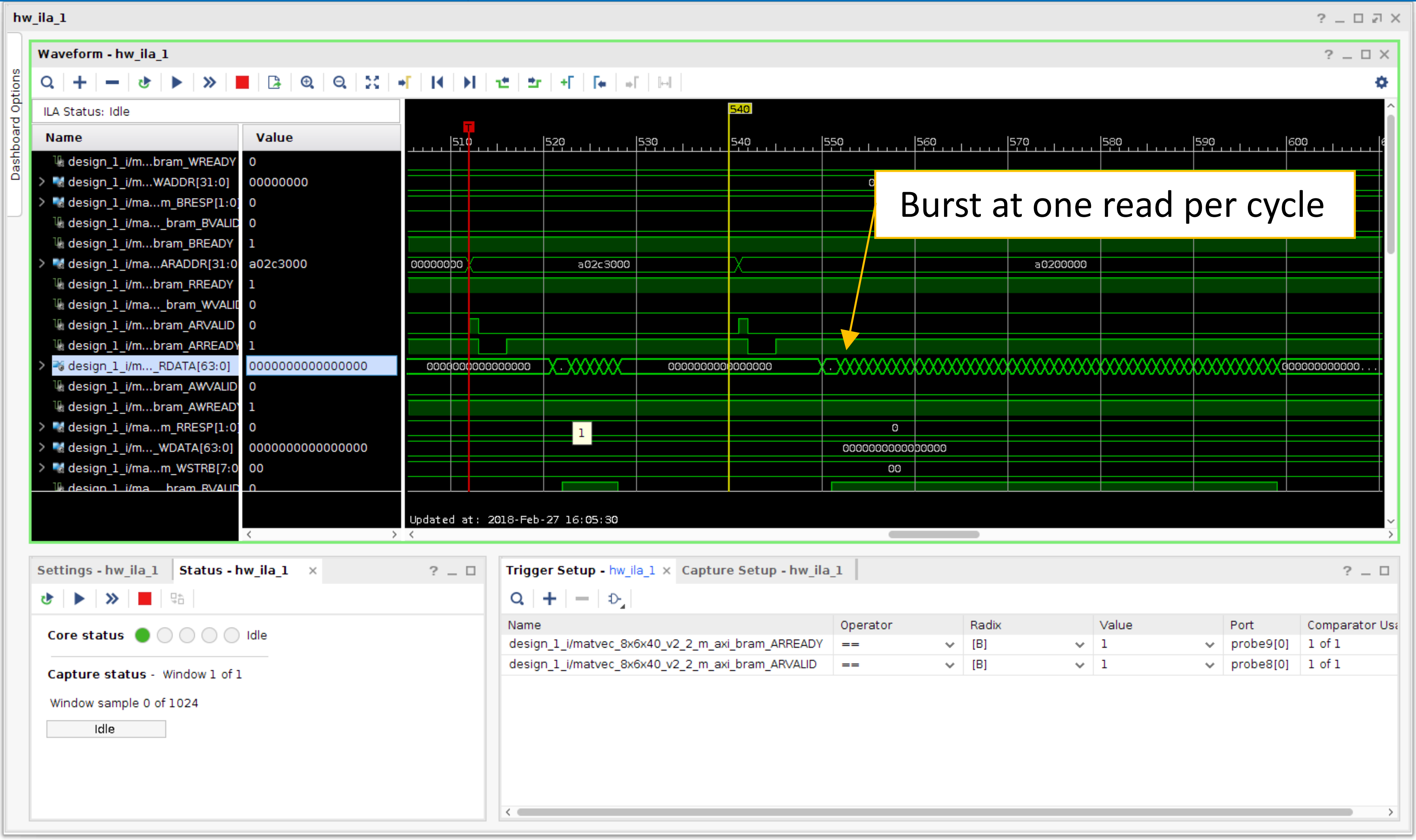
Idle

Trigger Setup - hw_ila_1 | Capture Setup - hw_ila_1

Name	Operator	Radix	Value	Port	Comparator Usa
design_1_i/matvec_8x6x40_v2_2_m_axi_bram_ARREADY	==	[B]	1	probe9[0]	1 of 1
design_1_i/matvec_8x6x40_v2_2_m_axi_bram_ARVALID	==	[B]	1	probe8[0]	1 of 1

Produced by attaching an Integrated Logic Analyzer IP block to the data path going into the matrix-vector IP block

Optimized Data Streaming - After



hw_ila_1

Waveform - hw_ila_1

ILA Status: Idle

Name	Value
design_1_i/m...bram_WREADY	0
design_1_i/m...WADDR[31:0]	00000000
design_1_i/ma...m_BRESP[1:0]	0
design_1_i/ma..._bram_BVALID	0
design_1_i/m...bram_BREADY	1
design_1_i/ma...ARADDR[31:0	a02c3000
design_1_i/m...bram_RREADY	1
design_1_i/ma..._bram_WVALID	0
design_1_i/m...bram_ARVALID	0
design_1_i/m...bram_ARREADY	1
design_1_i/m..._RDATA[63:0]	0000000000000000
design_1_i/m...bram_AWVALID	0
design_1_i/m...bram_AWREADY	1
design_1_i/ma...m_RRESP[1:0]	0
design_1_i/m..._WDATA[63:0]	0000000000000000
design_1_i/ma...m_WSTRB[7:0	00
design_1_i/ma..._bram_RVALID	0

Updated at: 2018-Feb-27 16:05:30

Burst at one read per cycle

Settings - hw_ila_1 | Status - hw_ila_1

Core status ● Idle

Capture status - Window 1 of 1

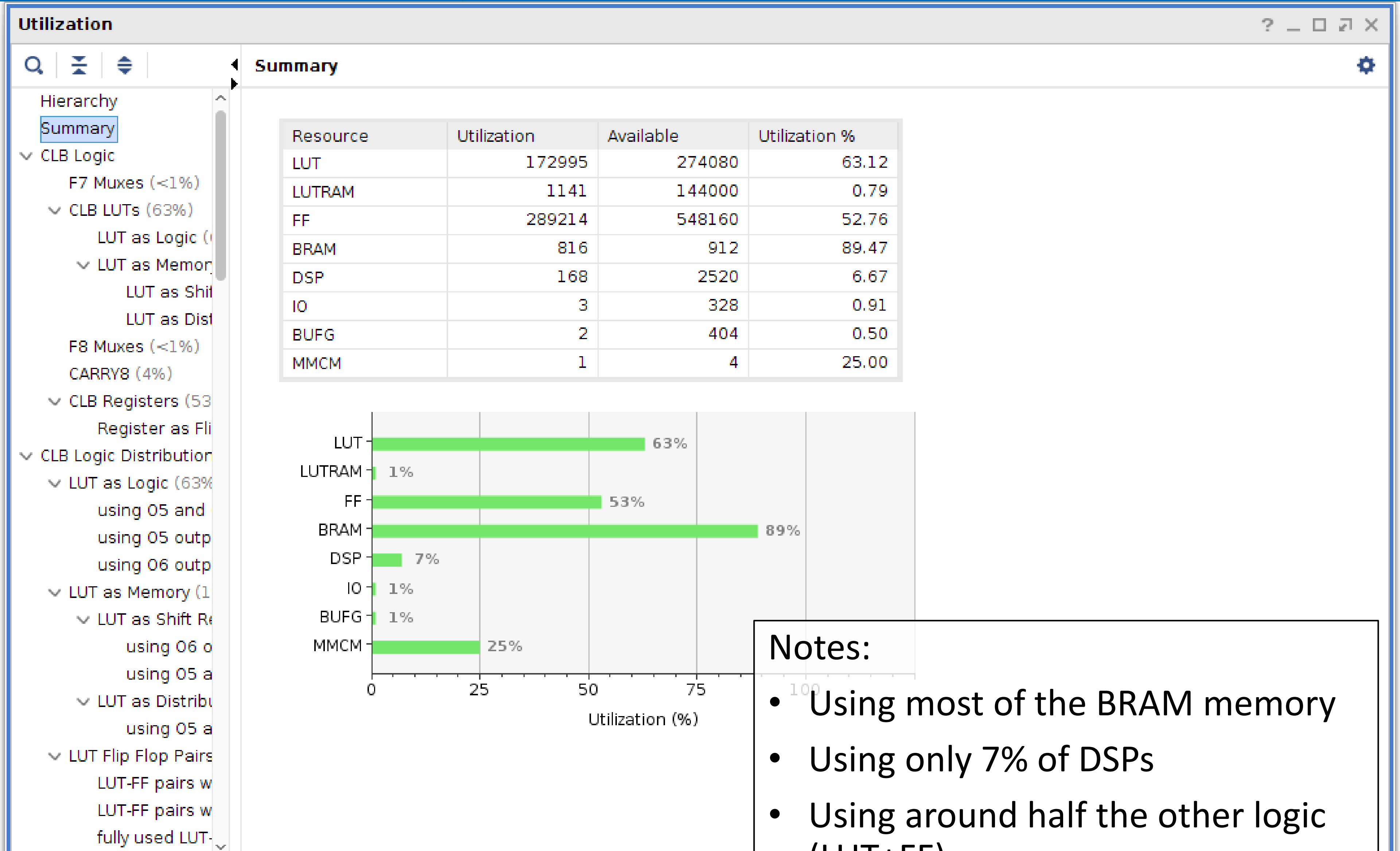
Window sample 0 of 1024

Idle

Trigger Setup - hw_ila_1 | Capture Setup - hw_ila_1

Name	Operator	Radix	Value	Port	Comparator Usa
design_1_i/matvec_8x6x40_v2_2_m_axi_bram_ARREADY	==	[B]	1	probe9[0]	1 of 1
design_1_i/matvec_8x6x40_v2_2_m_axi_bram_ARVALID	==	[B]	1	probe8[0]	1 of 1

Vivado DS Resource Utilization



Resource	Utilization	Available	Utilization %
LUT	172995	274080	63.12
LUTRAM	1141	144000	0.79
FF	289214	548160	52.76
BRAM	816	912	89.47
DSP	168	2520	6.67
IO	3	328	0.91
BUFG	2	404	0.50
MMCM	1	4	25.00

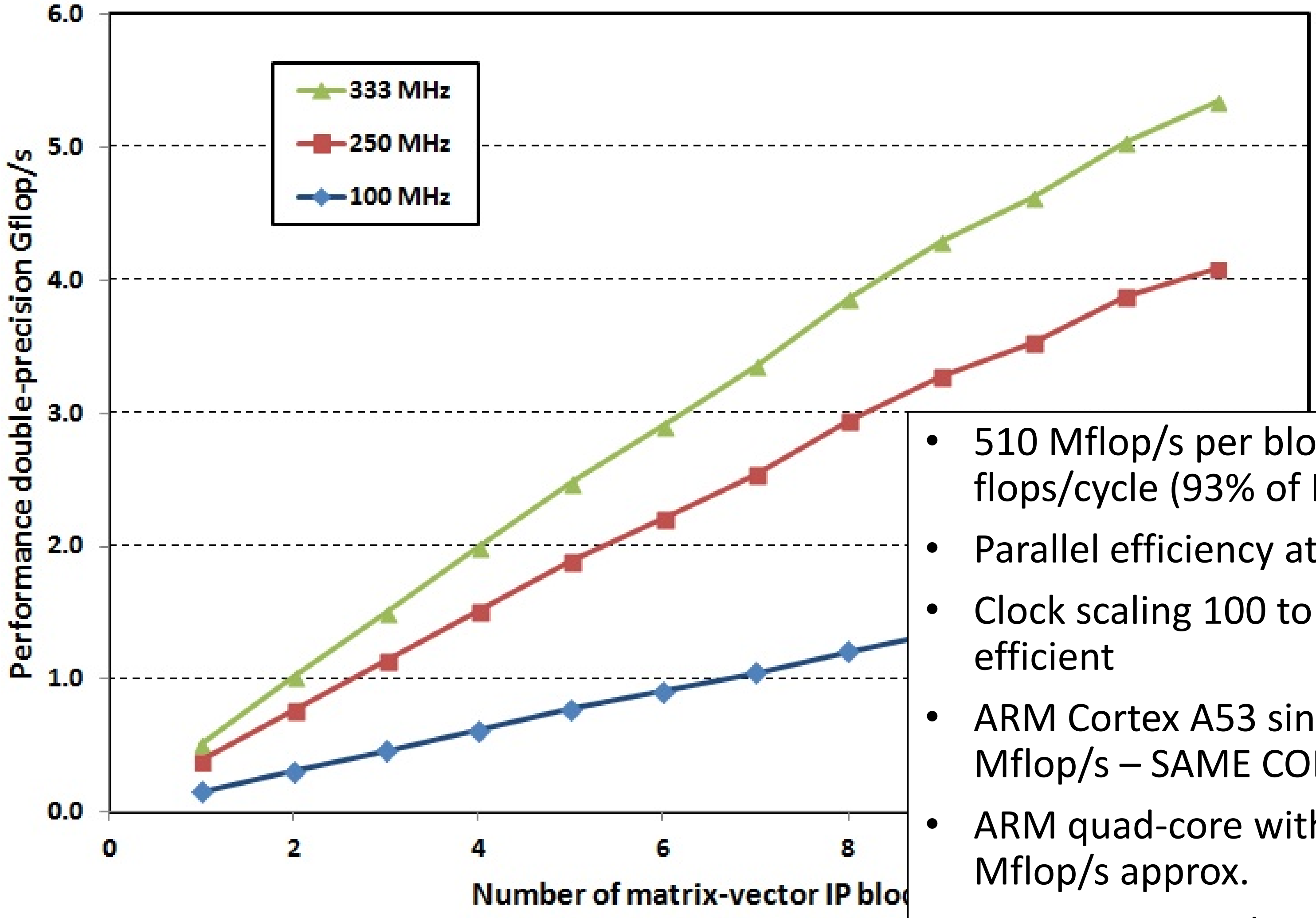
Notes:

- Using most of the BRAM memory
- Using only 7% of DSPs
- Using around half the other logic

ARM driver code

- Setup a two devices /dev/uio0 and /dev/uio1 – two ports on the ZynQ block
- Use mmap to map the FPGA memory into user space
- Assign pointers for each data array to location in user space
- Control loop to divide up the work into 12 “chunks” which will fit into the FPGA BRAM memory (maximum 12 x 256kB = 3MB) (13 columns in this LFRic model)
- For each chunk:
  - Assign work to one of the matrix-vector blocks
  - Copy input data into BRAM
  - Set the control word “registers” for the block
  - Start the block by setting AP_START
  - Wait for block to finish by watching AP_IDLE (opportunity for overlap)
  - Copy output data from BRAM
- In practice we fill 3MB BRAM, then run all 12 matrix-vector blocks, then copy output data back and repeat
- Check correctness and time the code

Results for 12 blocks



- 510 Mflop/s per block => 1.53 flops/cycle (93% of HLS estimate)
- Parallel efficiency at 12 IP blocks 87%
- Clock scaling 100 to 333 MHz is 94% efficient
- ARM Cortex A53 single core 227 Mflop/s – SAME CODE
- ARM quad-core with OpenMP 800 Mflop/s approx.
- FPGA:ARM quad-core speed-up: 6.7x

Summary

We have

- Used Vivado HLS to develop a matrix-vector kernel which runs on the UltraScale+ FPGA at 5.3 double precision Gflop/s

Issues

- Timing constraints in the Vivado design prevent larger numbers of blocks and higher clock speeds
- Therefore cannot exploit all the FPGA logic for this algorithm

Future work

- Generate an IP block and driver for the LFRic code: apply_variable_hx_kernel_code (HLS done; 1.75 flops/cycle)
- Exploit MPI within LFRic to run across multiple nodes and multiple FPGAs (done trivially with the matrix-vector kernel)
- How many other kernels can we port to the FPGAs?
- Can we link kernels to avoid data transfer?
- When do we need to reconfigure? At what cost?
- Future hardware: now ZU9, VU9 (early 2019) and HBM

Many thanks
Please connect at
@euroexa or euroexa.eu

Mike Ashworth, Graham Riley, Andrew Attwood and John Mawer
Advanced Processor Technologies Group
School of Computer Science,
University of Manchester, United Kingdom
mike.ashworth.compsci@manchester.ac.uk