Fixing the E3xx DMA Bottleneck

Implementing a High-Rate Heterogeneous FPGA/Processor Transport

EJ Kreinar

Lorin Metzger

Dan CaJacob

HawkEye 360

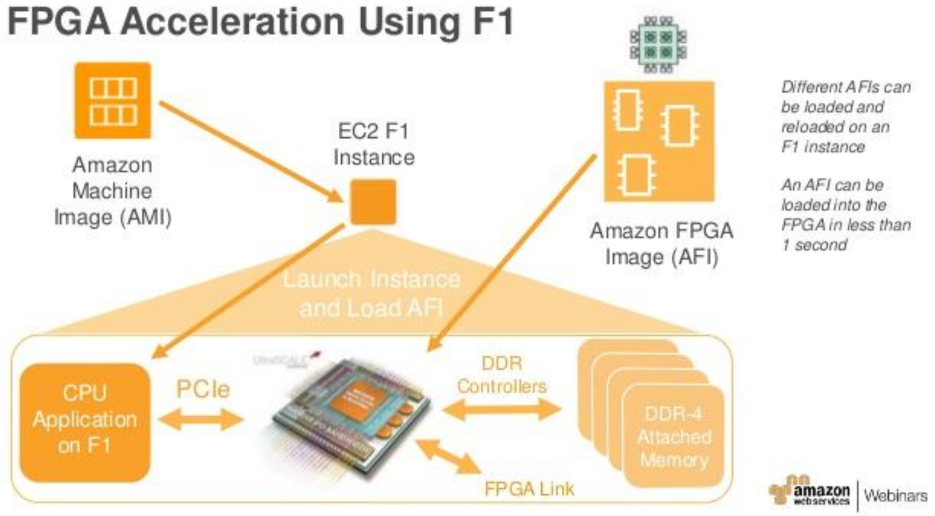
Topics

- Problem Introduction & Motivation
- Direct Memory Access (DMA) Background
- Technical Solution
  - FPGA
  - Kernel Driver
  - User-Space Software
- Results & Tradeoffs

Problem Introduction

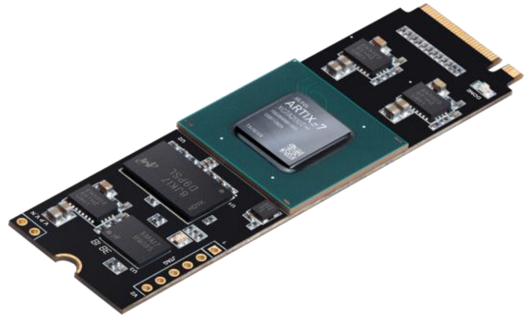
- Heterogeneous computing requires a data transfer
  - Data must be moved from one computing device to another
- This data transfer is often a bottleneck in system throughput and latency

NVIDIA GPU



FPGA Acceleration Using F1
Amazon Machine Image (AMI)
EC2 F1 Instance
Amazon FPGA Image (AFI)
Different AFIs can be loaded and reloaded on an F1 instance
An AFI can be loaded into the FPGA in less than 1 second
Launch Instance and Load AFI
CPU Application on F1
PCIe
DDR Controllers
DDR-4 Attached Memory
FPGA Link
amazon web services | Webinars

AWS F1 CPU + FPGA


"Aller" Artix 7 FPGA with M.2 Interface

Takeaway Point

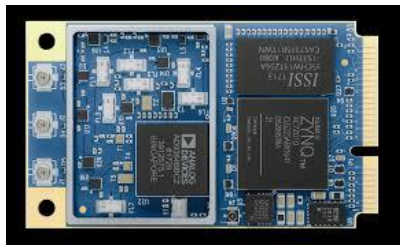
HawkEye 360

… for Software Defined Radio?

- Self-contained, low-SWAP Software Defined Radios...

Ettus E310

Ettus E320



Epiq Sidekick Z2

Lime SDR

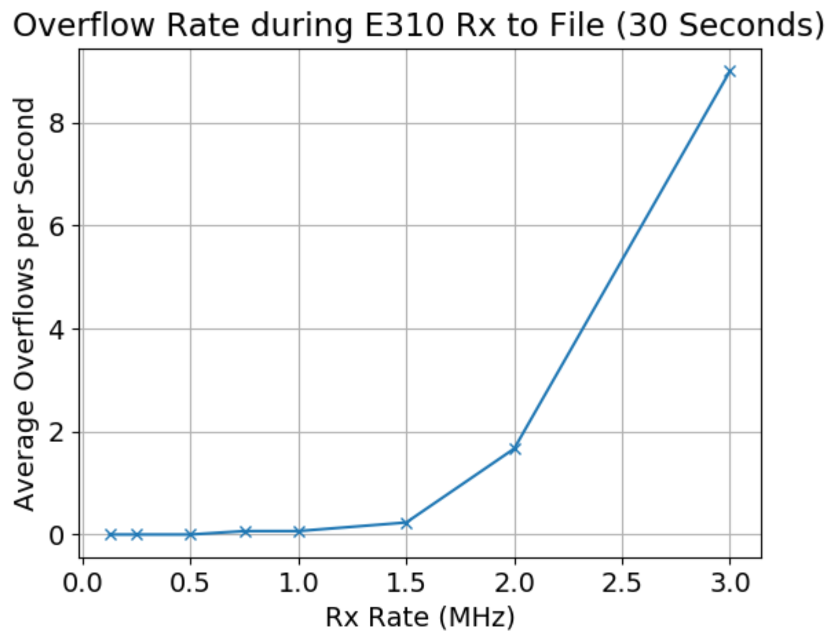
- Often using a Zynq "System on Chip": FPGA + ARM Processor

We'll focus on the Xilinx Zynq platform

HawkEye 360

Performance Metrics

- Example Device: **Ettus E310**
  - (No PS Load) Max continuous receive rate without overflows: **10 MHz**
  - (Writing to File) Max continuous receive rate without overflow: **~1 MHz**
- File Write Test:
  - 1-channel receive only. Increment rx sample rate
  - Collect continuously for 30 seconds
  - Write to disk in natural "wire format", ie interleaved short 16-bit ints



Some devices may NOT achieve theoretical limits

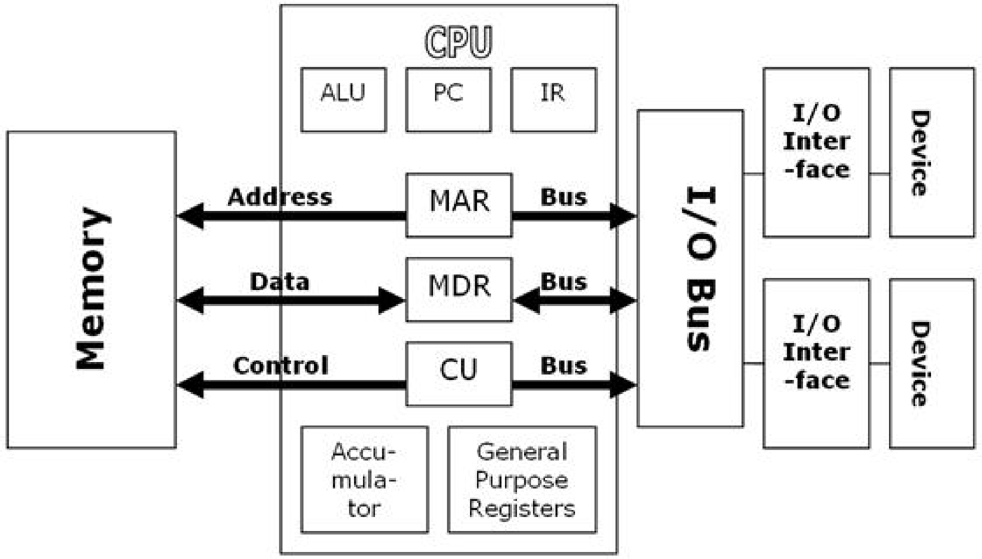
Making the Invisible Visible
Making the Invisible Visible
Making the Invisible Visible

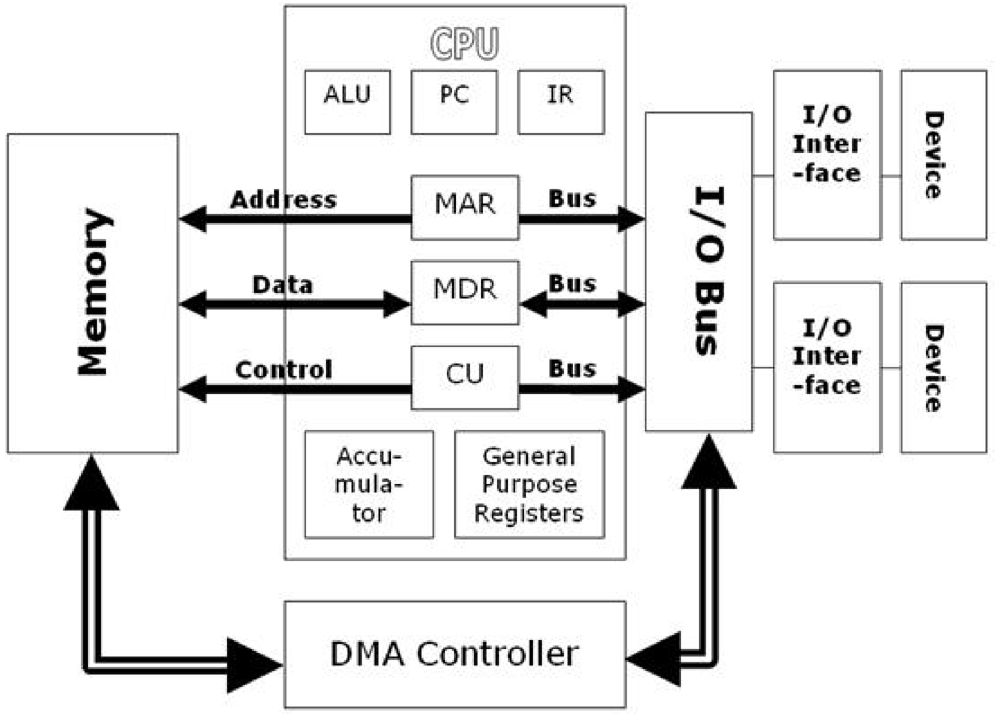
Direct Memory Access (DMA) Background

HawkEye 360

Direct Memory Access (DMA)

- Direct Memory Access is a feature that allows memory copy operations to happen *without* processor involvement



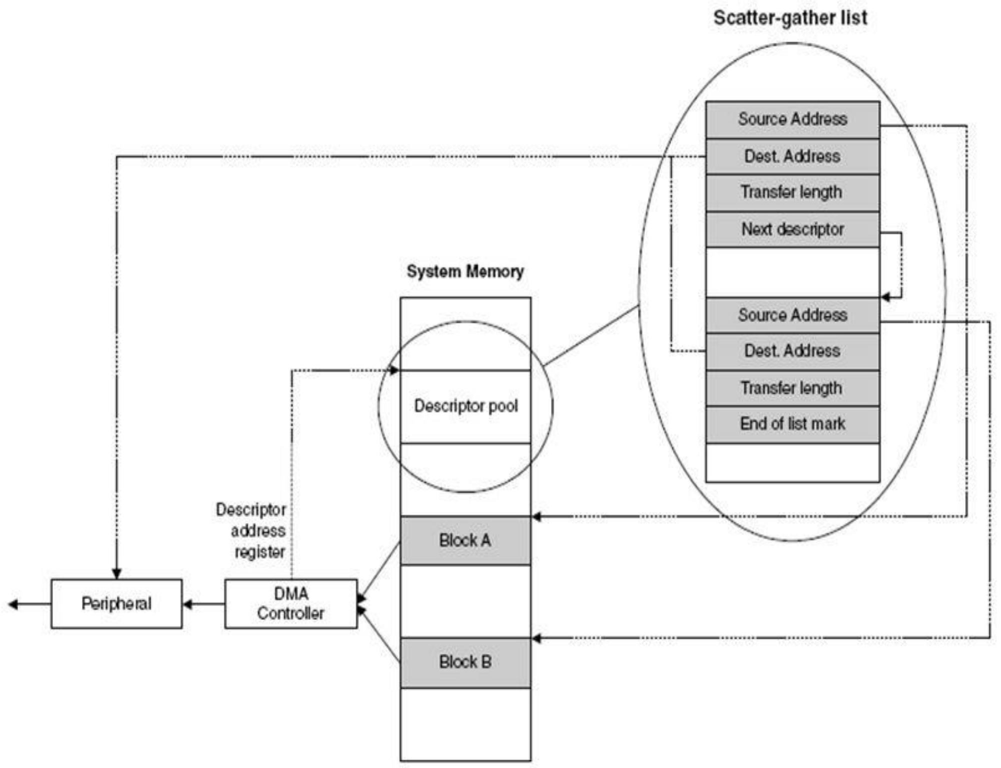


DMA generally requires use of a purpose-designed hardware architecture to perform the optimized copy

Scatter Gather

- Scatter-gather allows the transfer of data to and from multiple memory areas in a single DMA transaction
- It is equivalent to chaining together multiple simple DMA requests
  - The motivation is to off-load multiple input/output interrupt and data copy tasks from the CPU
  - Allows for more efficient memory use by kernel by with less potential memory fragmentation vs large continuous buffers.

Scatter Gather DMA Mode

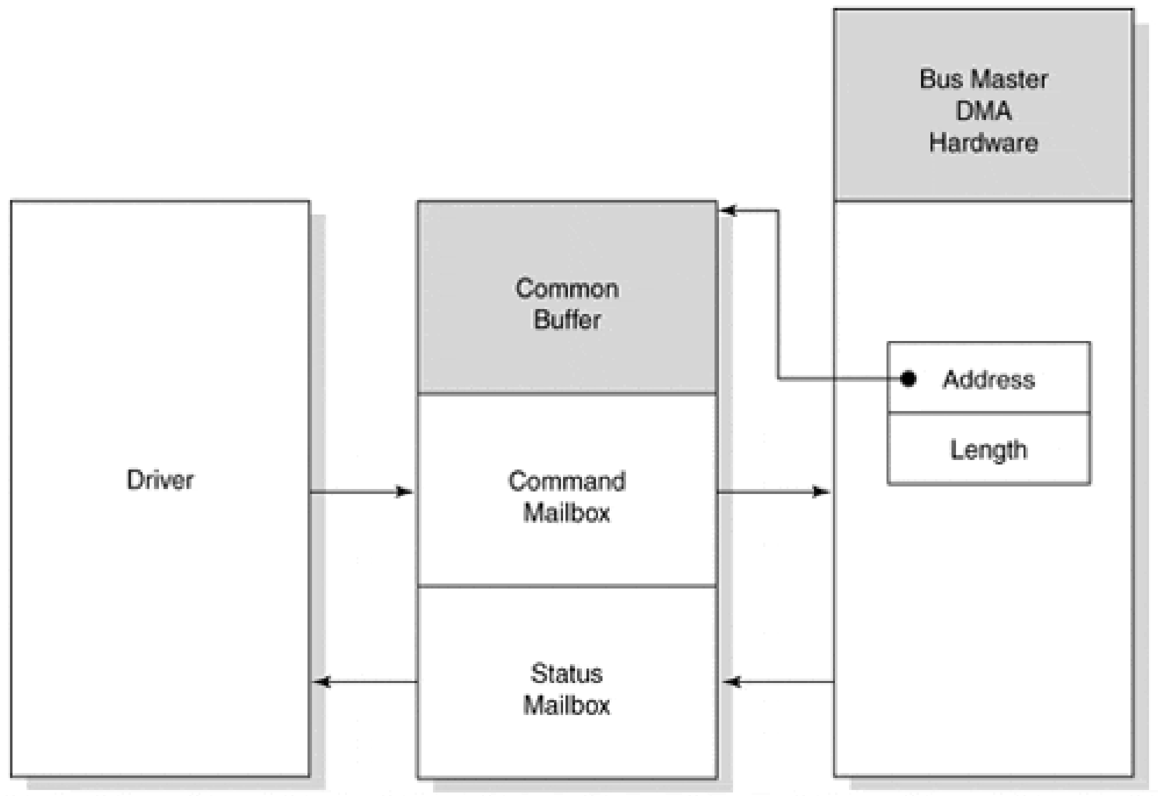


Source: Symbian OS Internals/13. Peripheral Support

FPGA controls buffer swapping with minimal PS interaction

Direct Register Mode

- Direct Register Mode (non-scatter gather) provides a configuration for doing simple DMA transfers
- Transfers are initiated by accessing the DMACR, the Source or Destination Address, and the Length registers
- When the transfer is completed, an IRQ register asserts for the associated channel and if enabled generates an interrupt output signal

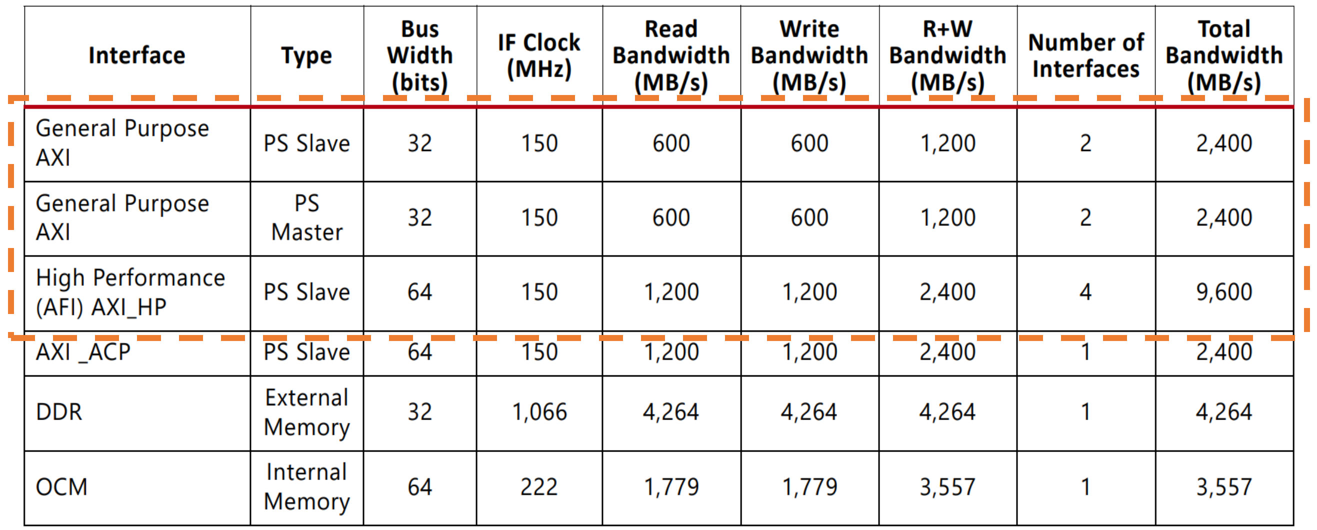


PS required to service IRQs and manipulate DMA buffer

DMA on the Zynq Architecture

- FPGA fabric is allocated on the system's AXI4 crossbar
- FPGA IP interacts with the DDR Memory Controller through the memory interconnects

Table 22-2: **Theoretical Bandwidth of PS-PL and PS Memory Interfaces**



Interface	Type	Bus Width (bits)	IF Clock (MHz)	Read Bandwidth (MB/s)	Write Bandwidth (MB/s)	R+W Bandwidth (MB/s)	Number of Interfaces	Total Bandwidth (MB/s)
General Purpose AXI	PS Slave	32	150	600	600	1,200	2	2,400
General Purpose AXI	PS Master	32	150	600	600	1,200	2	2,400
High Performance (AFI) AXI_HP	PS Slave	64	150	1,200	1,200	2,400	4	9,600
AXI _ACP	PS Slave	64	150	1,200	1,200	2,400	1	2,400
DDR	External Memory	32	1,066	4,264	4,264	4,264	1	4,264
OCM	Internal Memory	64	222	1,779	1,779	3,557	1	3,557

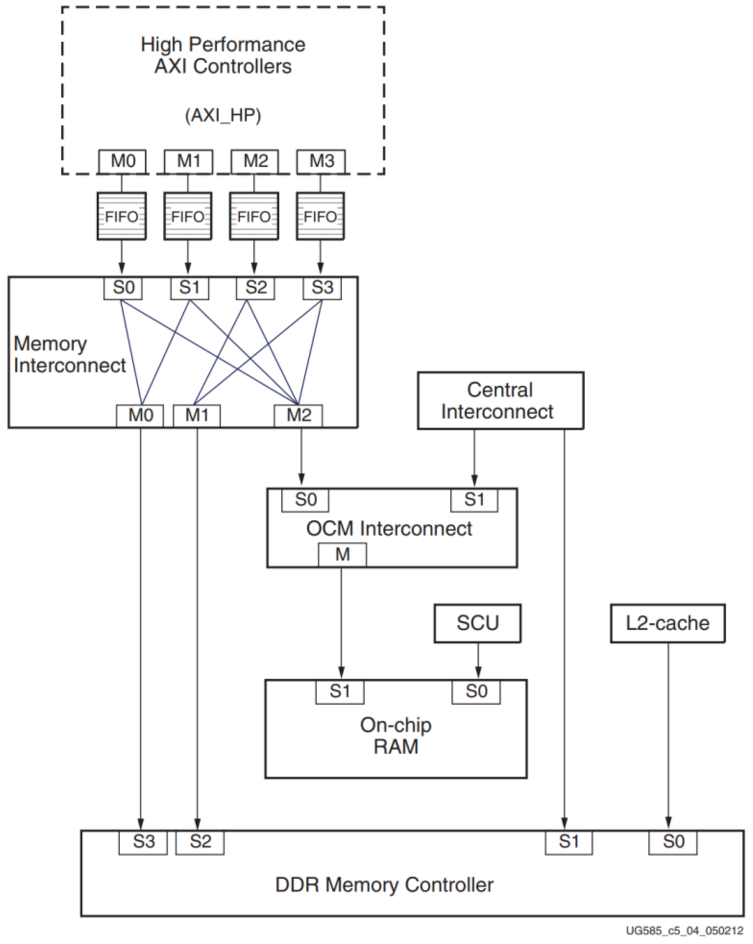
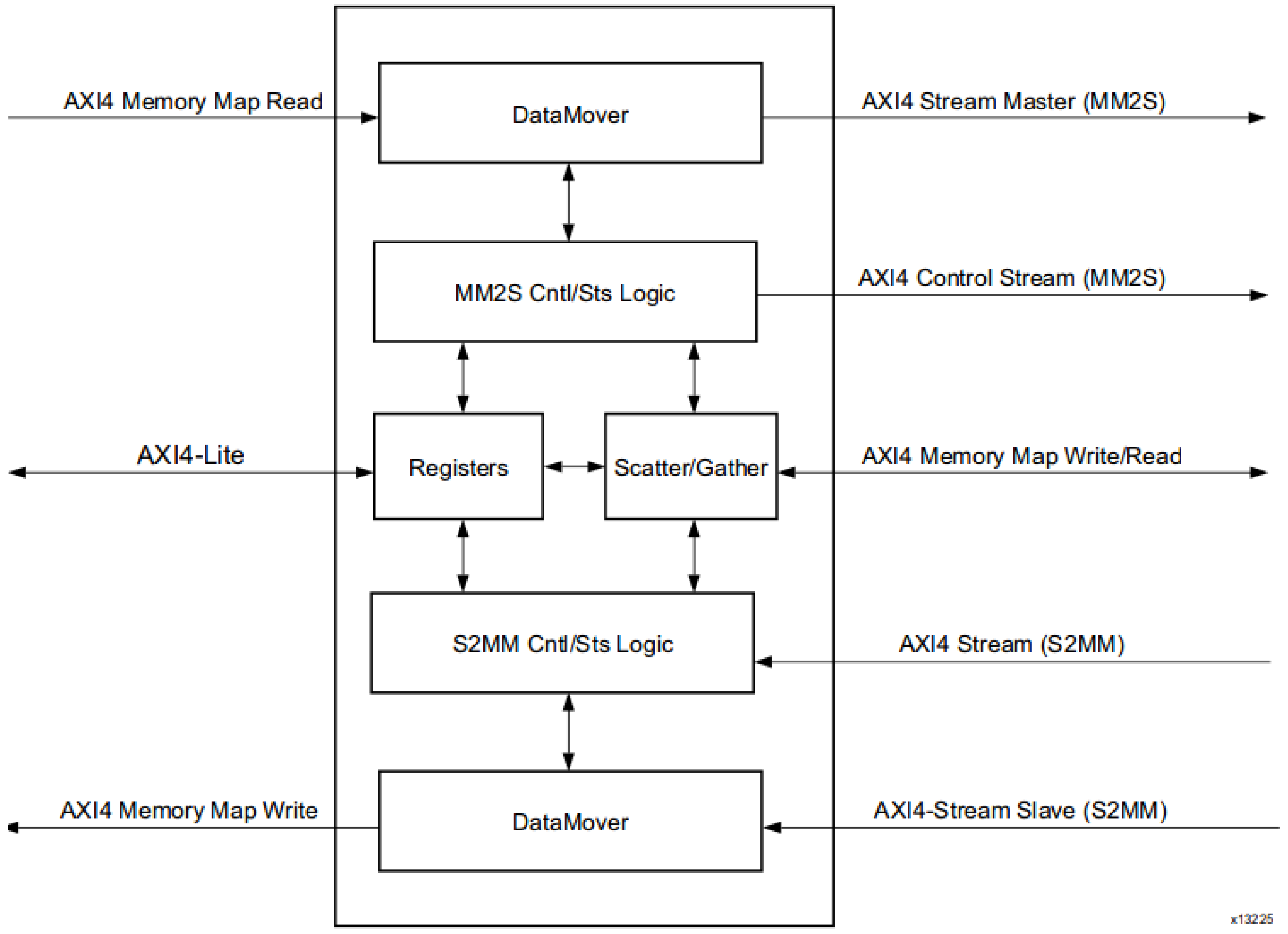


Figure 5-4: **High Performance (AXI_HP) Connectivity**

Max theoretical bandwidth of 600 MB/s - 1200 MB/s

DMA on the Zynq Architecture

- AXI DMA distinguishes two channels
  - MM2S (memory-mapped to stream) transports data from DDR memory to FPGA
  - S2MM (stream to memory-mapped) transports arbitrary data stream to DDR memory
- Uses AXI-4-Lite interface for configuring DMA control registers.



Polling vs Interrupt Transfers

Polling driver periodically interacts with DMA registers

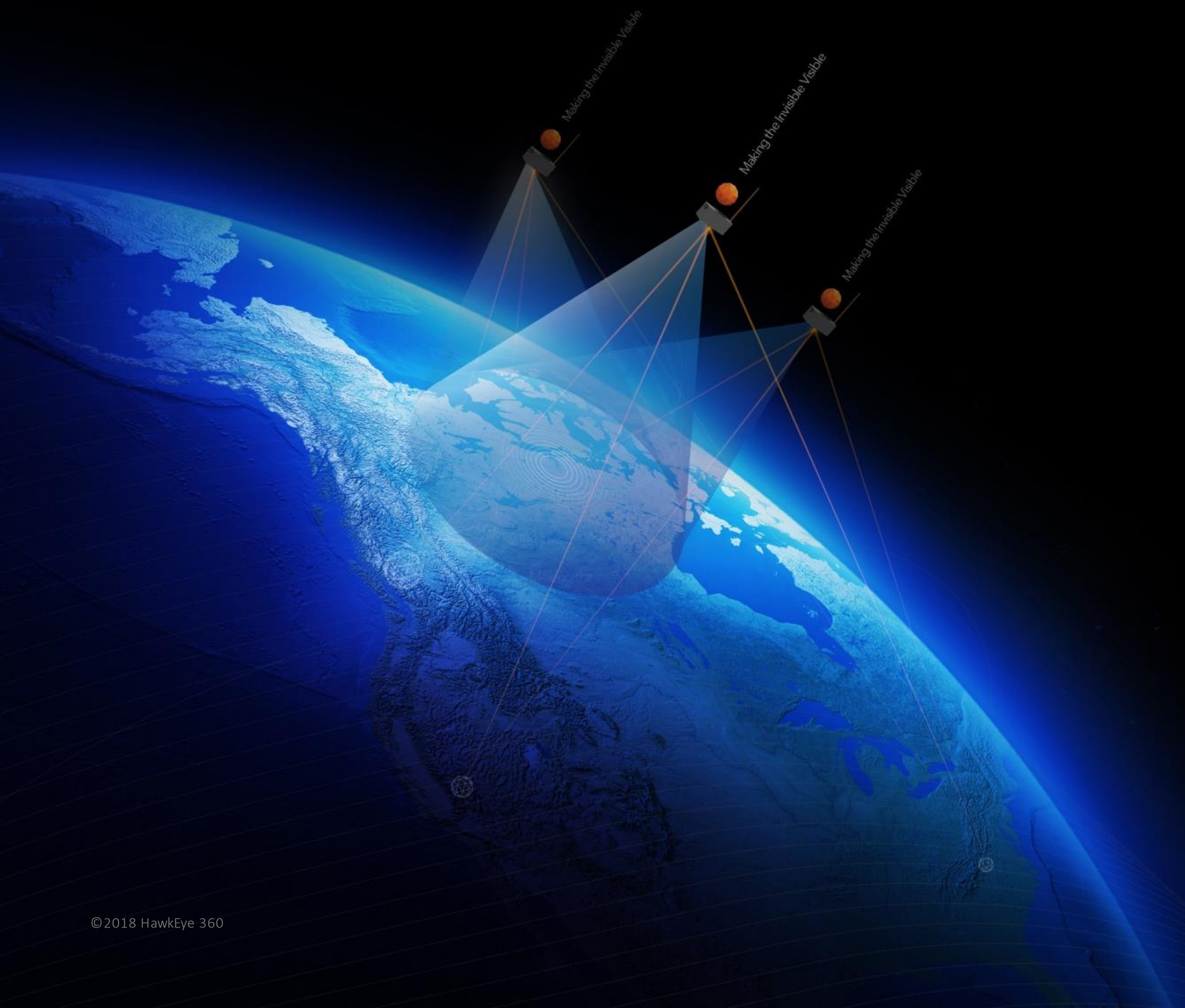
- Benefits:
  - Simple to implement and test!
- Drawbacks:
  - Must run PS in a tight loop to respond quickly to DMA

Interrupt driver interacts with DMA on demand based on IRQ signals

- Benefits:
  - Allows the PS to react immediately to interrupt request signals
  - Software does not load the PS if DMA is not used
- Drawbacks:
  - Requires kernel module development. Userspace programs do not directly handle interrupts in a Linux OS

Polling may be helpful to prototype, but mostly all DMA software ultimately requires a kernel driver with interrupt handling

HawkEye 360


Making the Invisible Visible
Making the Invisible Visible
Making the Invisible Visible

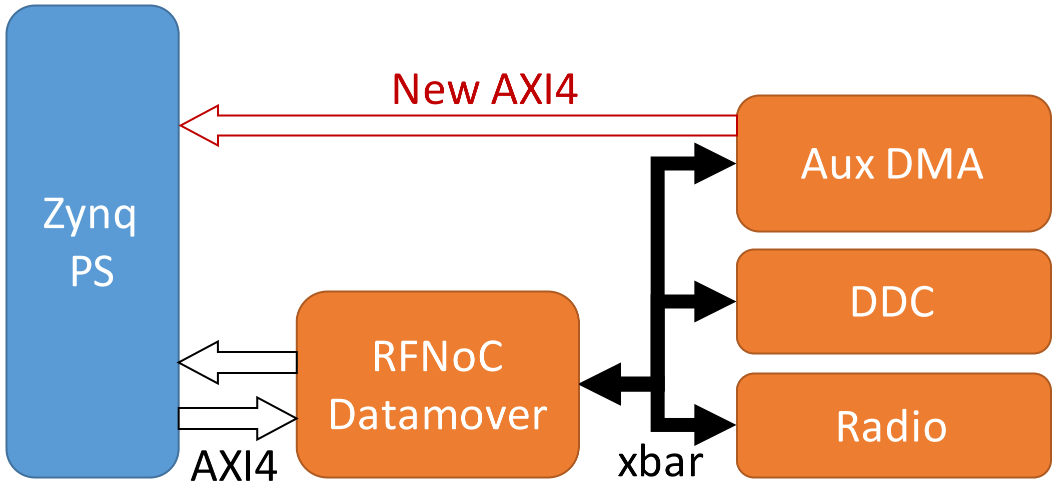
Technical Solution:
FPGA, Kernel Driver, User-Space Software

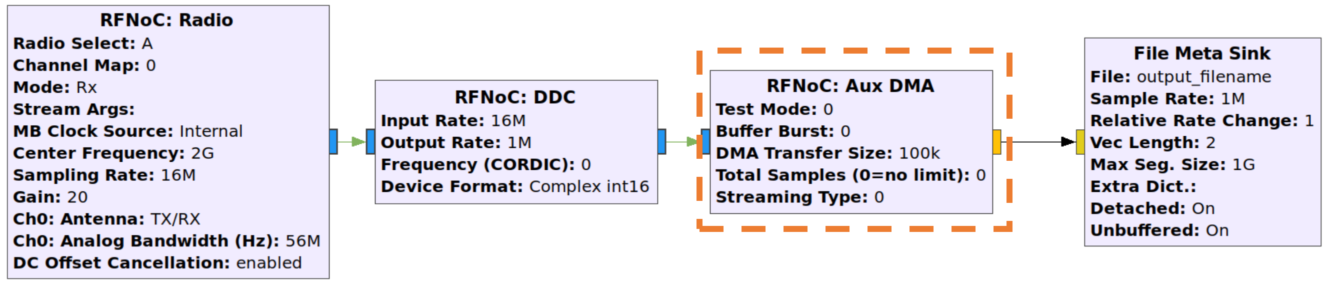
HawkEye 360

Overview

Major Surgery:

- Aux DMA: noc_block_auxdma connects directly to the E310 processor via AXI4 DMA
- New kernel driver services DMA interrupts
- Reuse typical RFNoC register reads/writes and graph infrastructure
- Aux DMA overrides "general_work" function of rfnoc_block_impl (in gr-ettus)





RFNoC: Radio
Radio Select: A
Channel Map: 0
Mode: Rx
Stream Args:
MB Clock Source: Internal
Center Frequency: 2G
Sampling Rate: 16M
Gain: 20
Ch0: Antenna: TX/RX
Ch0: Analog Bandwidth (Hz): 56M
DC Offset Cancellation: enabled

RFNoC: DDC
Input Rate: 16M
Output Rate: 1M
Frequency (CORDIC): 0
Device Format: Complex int16

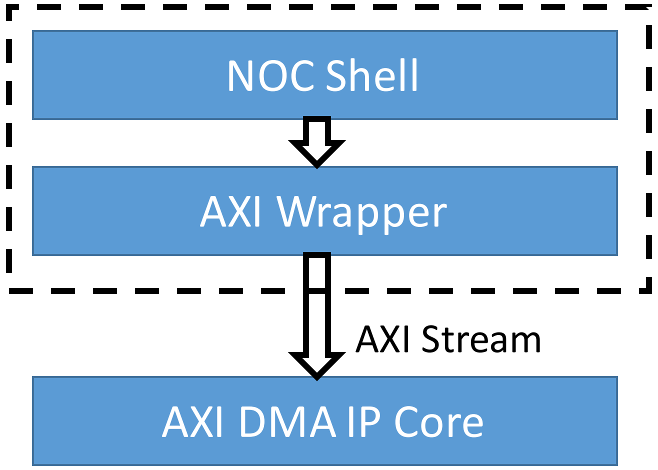
RFNoC: Aux DMA
Test Mode: 0
Buffer Burst: 0
DMA Transfer Size: 100k
Total Samples (0=no limit): 0
Streaming Type: 0

File Meta Sink
File: output_filename
Sample Rate: 1M
Relative Rate Change: 1
Vec Length: 2
Max Seg. Size: 1G
Extra Dict.:
Detached: On
Unbuffered: On

"Aux DMA" sends data directly into shared processor DRAM using large packet sizes

FPGA Architecture

- Xilinx AXI Datamover: Core data transfer functionality
  - Writes "Serial Data" to "Memory-Mapped" data (and vice versa) using the FPGA as the AXI-Master
- Xilinx AXI DMA
  - Wraps Datamover and adds control registers for Scatter Gather or Direct Register Modes
- RFNoC Integration: **Use Xilinx AXI DMA**
  - Create a new streaming "endpoint"
  - Transfer from "start time" to "end of burst"



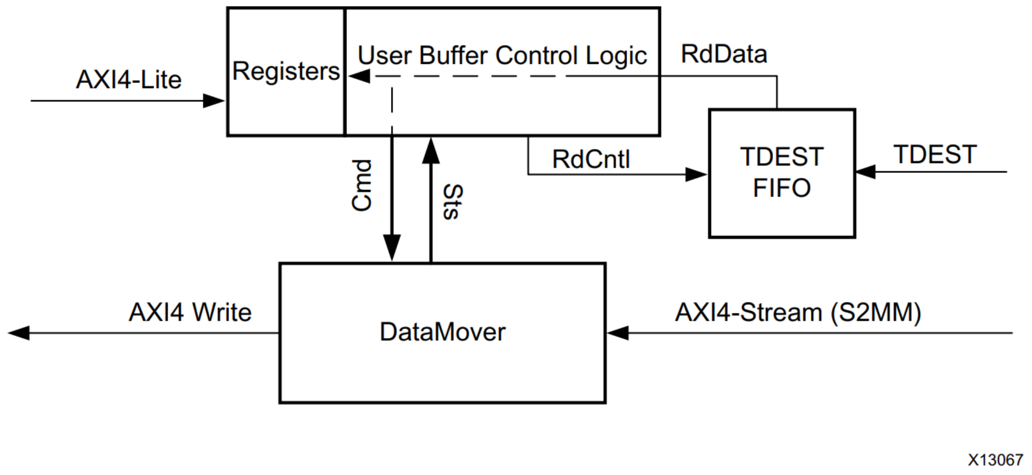


Figure 3-2: **Typical Application of S2MM DataMover**

HawkEye 360

Kernel Driver Options

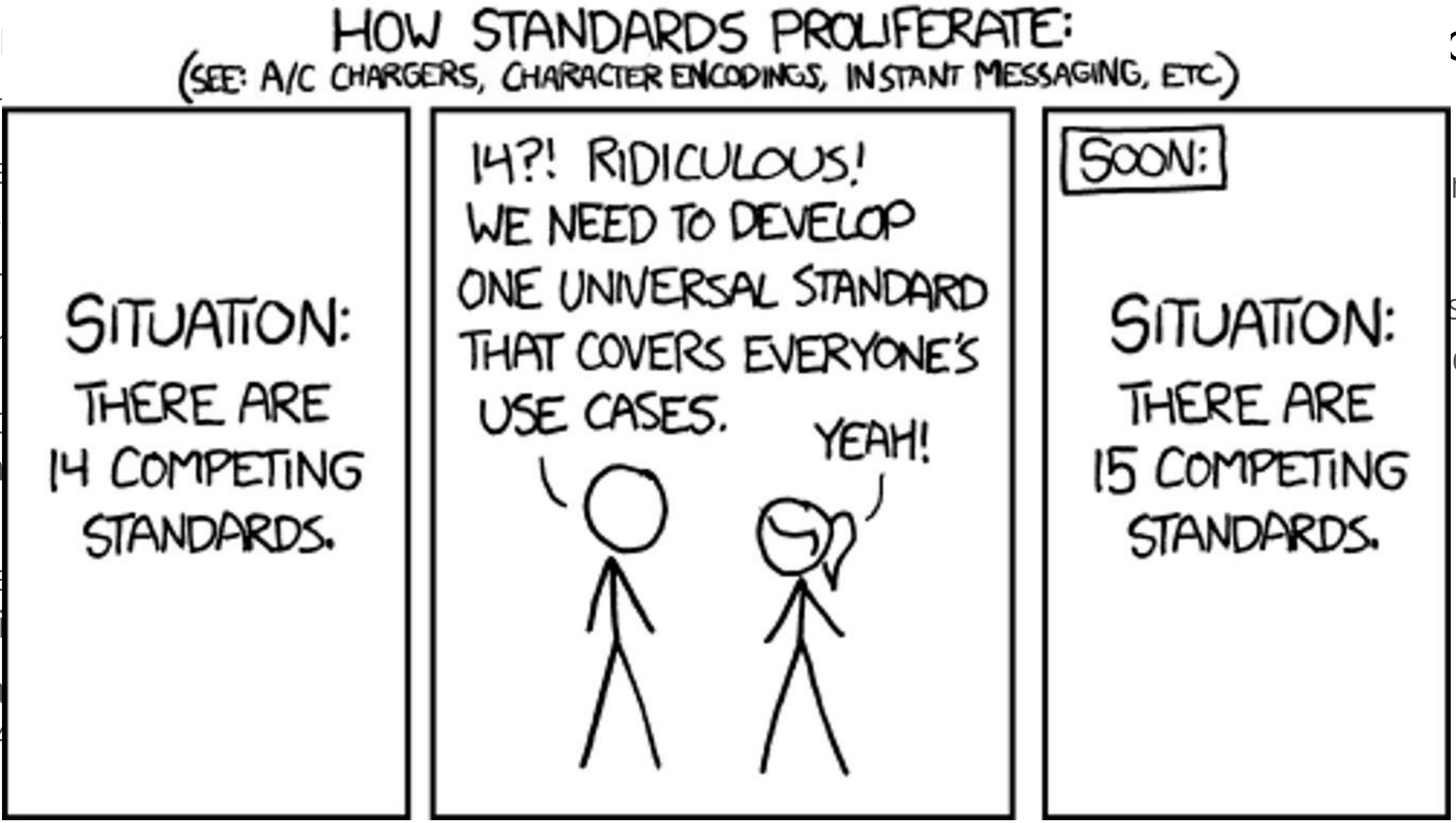
- **Xilinx AXI DMA Linux Driver**
  - Implemented as a Linux DMA engine
  - Allowed developing device/character driver against a generic DMA Linux API
  - Required a patch to supply result with 'residue' populated for transfers that were less than the buffer size. A patch fixing this was applied 5 months ago. Commit(0f7b82f11)
  - Driver was unstable for long running DMA transfers, would have spontaneous failures, requiring reset of DMA Controller
  - Several patches have been applied since 2017.4 release, which could address stability issues

- **IIO Driver for Xilinx Linux Kernel/DMA Controller**
  - Analog Devices solution!
  - Appealing option for common interface implementation
  - Ultimately decided a simple clean solution tailored to our specific needs would be better

Custom solution for our application likely the quickest and most reliable solution to the problem

Kernel Driver Options

- Xilinx AXI Controller
  - Impler
  - Allowe rface
    agains
  - Requir
    'residu solution
    less th d be better
    was ap
    Comm
  - Driver
    transfe
    requir
  - Severa
    2017.
    issues


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Custom solution for our application likely the quickest and most reliable solution to the problem

HawkEye 360

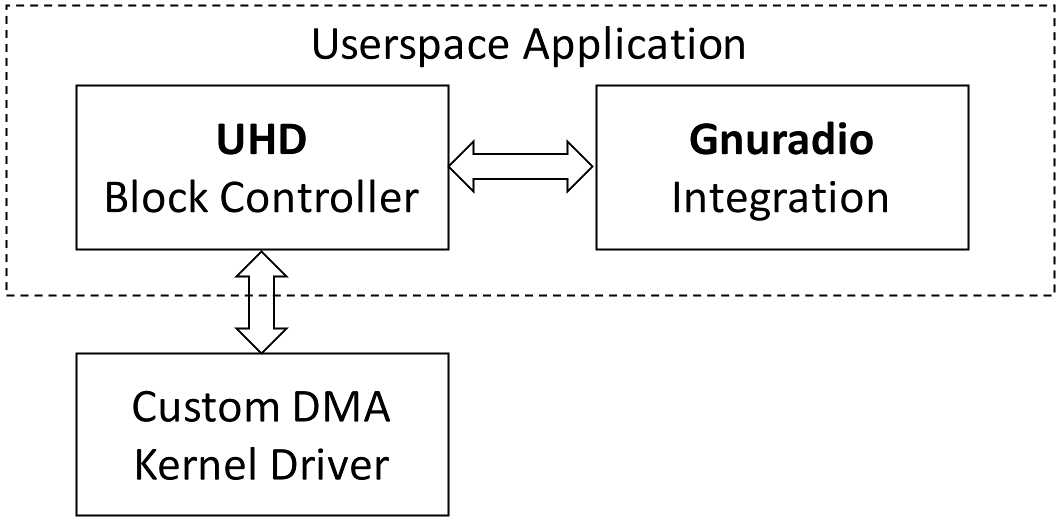
Custom Kernel Driver

- Custom AUX DMA Linux Driver!
  - Implemented as a character device driver
  - Pre-allocate memory buffers dma_alloc_coherent when user space application calls open on file handle
    - *Kernel* parameters to specify how many buffers, and what size the buffers are.
  - When multiple buffers are allocated, will automatically swap free buffers for filled buffers in DMA interrupt, until all free buffers are filled, at which point DMA controller will be stopped.
- Multiple large buffers allows us to capture several seconds of high rate RF data uninterrupted into pre-allocated kernel buffers
- Allocate 160 MiB RAM for DMA
- Userspace applications then process/store after the burst has completed

Larger buffers decrease PS interaction load; Userspace can service more data!

User-Space Software

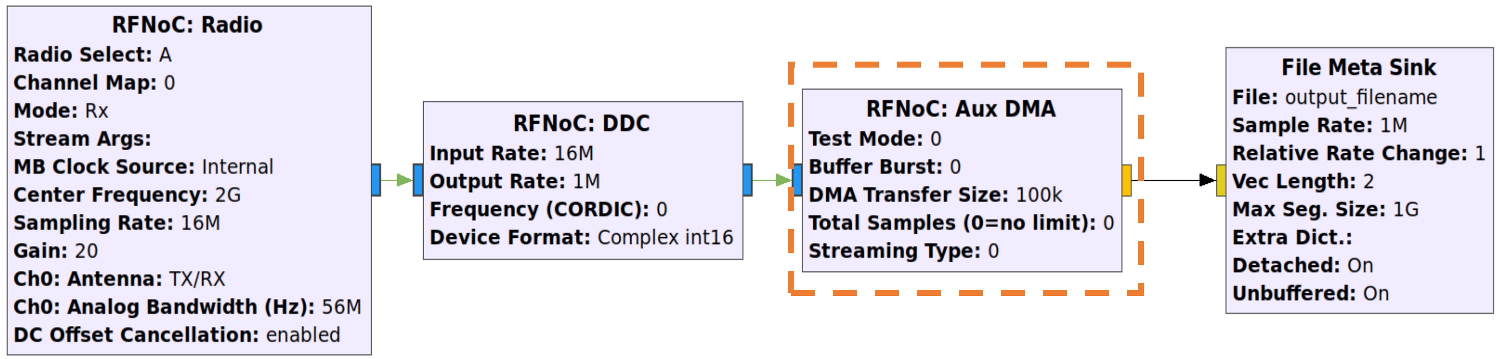
- Userspace applications can retrieve buffers 2 ways: blocking **read()** or **mmap()**
  - **read()** will copy buffer to userspace memory, and automatically free kernel buffer. Not as efficient.
  - **mmap()** maps kernel buffer to userspace, and user application gets address of next filled buffer via ioctl call. Frees read buffers with ioctl call. ("zero-copy")
- Memory-mapped buffer handling performed in UHD block controller software



HawkEye 360

Gnuradio Software

- Override general_work function of Gnuradio block
  - Request a new buffer from kernel driver
  - Fill the output buffer with the desired number of samples
  - Save samples greater than the requested noutput_items for the next iteration
- Optimized to transfer large buffers (128 KiB) straight into Gnuradio as fast as possible!
- Downsides: One channel only, zero flexibility, potentially large latency



RFNoC: Radio
Radio Select: A
Channel Map: 0
Mode: Rx
Stream Args:
MB Clock Source: Internal
Center Frequency: 2G
Sampling Rate: 16M
Gain: 20
Ch0: Antenna: TX/RX
Ch0: Analog Bandwidth (Hz): 56M
DC Offset Cancellation: enabled

RFNoC: DDC
Input Rate: 16M
Output Rate: 1M
Frequency (CORDIC): 0
Device Format: Complex int16

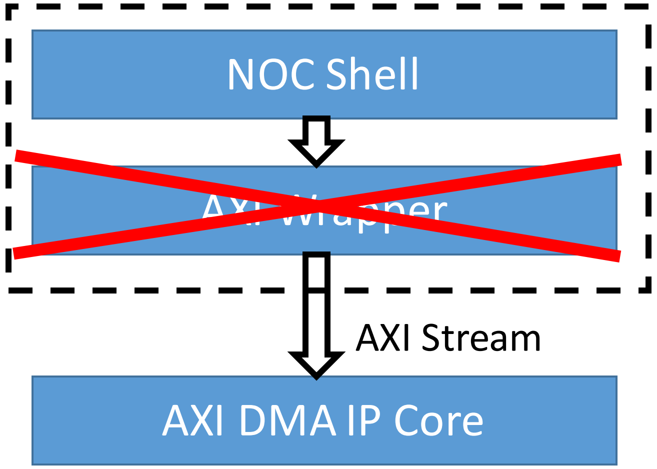
RFNoC: Aux DMA
Test Mode: 0
Buffer Burst: 0
DMA Transfer Size: 100k
Total Samples (0=no limit): 0
Streaming Type: 0

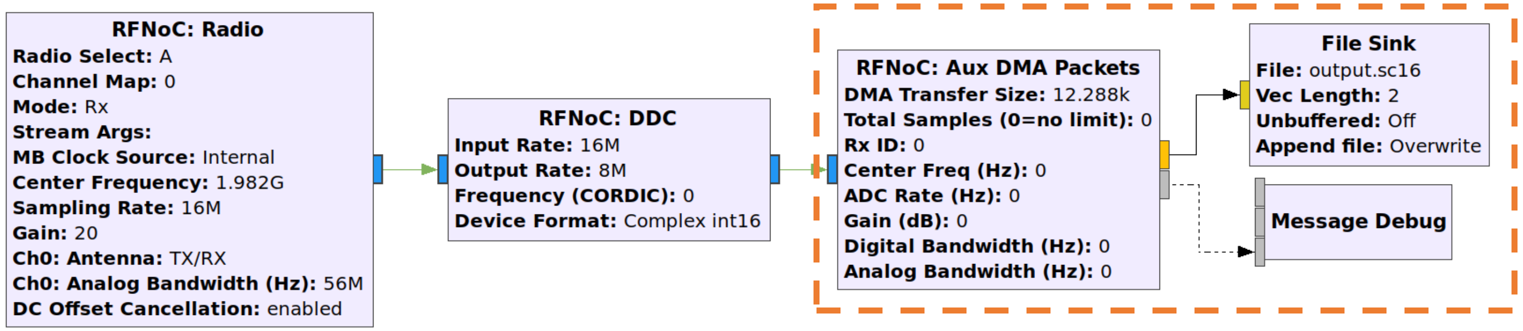
File Meta Sink
File: output_filename
Sample Rate: 1M
Relative Rate Change: 1
Vec Length: 2
Max Seg. Size: 1G
Extra Dict.:
Detached: On
Unbuffered: On

Audience Inquiry: Better ways to transfer data to/from Gnuradio besides "general work"?

Packet Coalescing

- Transfer **full CHDR packets** rather than streaming data
  - Skip "AXI Wrapper" in RFNoC Block
  - Pack many RFNoC transactions into one DMA transfer
  - Parse CHDR timestamp and metadata in Gnuradio
  - Convert output to PDUs or dump to disk
- Optimized for transferring many small packets or bursts
  - May also interleave multiple channels!





RFNoC: Radio
Radio Select: A
Channel Map: 0
Mode: Rx
Stream Args:
MB Clock Source: Internal
Center Frequency: 1.982G
Sampling Rate: 16M
Gain: 20
Ch0: Antenna: TX/RX
Ch0: Analog Bandwidth (Hz): 56M
DC Offset Cancellation: enabled

RFNoC: DDC
Input Rate: 16M
Output Rate: 8M
Frequency (CORDIC): 0
Device Format: Complex int16

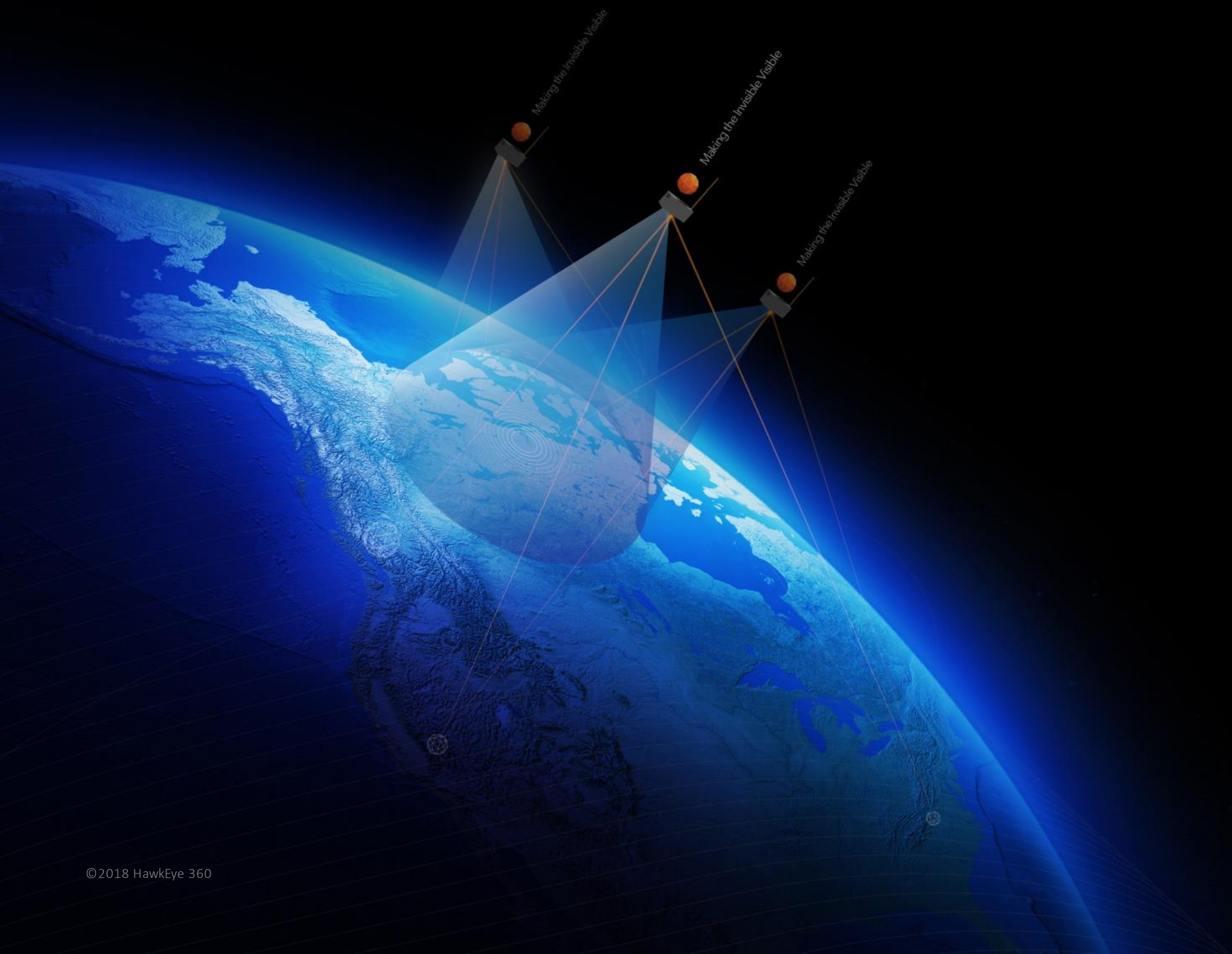
RFNoC: Aux DMA Packets
DMA Transfer Size: 12.288k
Total Samples (0=no limit): 0
Rx ID: 0
Center Freq (Hz): 0
ADC Rate (Hz): 0
Gain (dB): 0
Digital Bandwidth (Hz): 0
Analog Bandwidth (Hz): 0

File Sink
File: output.sc16
Vec Length: 2
Unbuffered: Off
Append file: Overwrite

Message Debug

Get Creative! It's software!

HawkEye 360


Making the Invisible Visible
Making the Invisible Visible
Making the Invisible Visible

Results & Tradeoffs

HawkEye 360

Results

- (No PS Load) Max continuous receive rate without overflows: **40 MHz**
- (Writing to File) Max **continuous** receive rate without overflow: **~4 MHz**
  - 1-channel receive only
  - Collect continuously for 30 seconds, using Gnuradio's file meta sink
- (Writing to File) Max **bursted** receive rate without overflow: **40 MHz**
  - First, write data into 160 MB allocated RAM
  - Second, write data from RAM into hard drive

"Continuous" rate limited by SD-card write speed and conflicting IO interrupts

"Bursted" rate saves the full data buffer into RAM before dumping to disk

HawkEye 360

Limitations and Tradeoffs

RFNoC Default DMA

- Small DMA buffers:
  - Lower **latency**
- Scatter Gather:
  - More **FPGA** control

HE360 DMA

- Large DMA buffers:
  - Higher **bandwidth**
- Direct Register Mode:
  - More **Processor** control
  - Simpler to implement

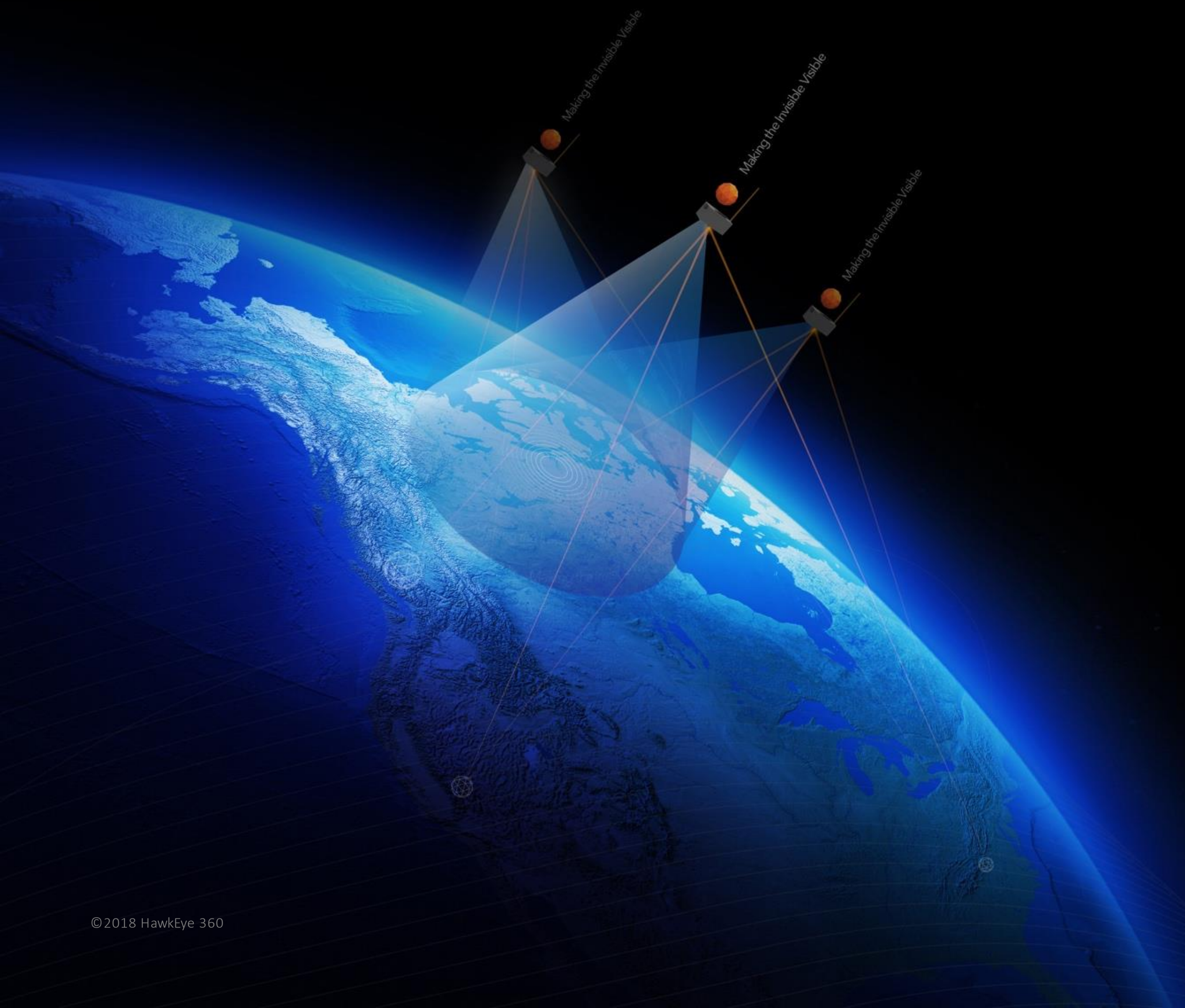
Consider **DMA implementation** vs **application goals**
when designing or selecting your transfer layer

HawkEye 360

Summary

- Heterogenous data transfer between processor and coprocessor (FPGA, GPU, etc) is a standard "problem" area
- Not all platforms/implementations are optimal for YOUR application
- Successful implementation requires a combination of custom FPGA, Linux kernel drivers, and userspace software
- We hope you feel equipped to address these issues in the future!

Full-rate heterogenous transfers into and out of Gnuradio can run on embedded Zynq-based SoC platforms


Making the Invisible Visible
Making the Invisible Visible
Making the Invisible Visible

Thank You!

EJ Kreinar: ej@he360.com

Lorin Metzger: lorin@he360.com

Dan CaJacob: dan@he360.com

HawkEye 360