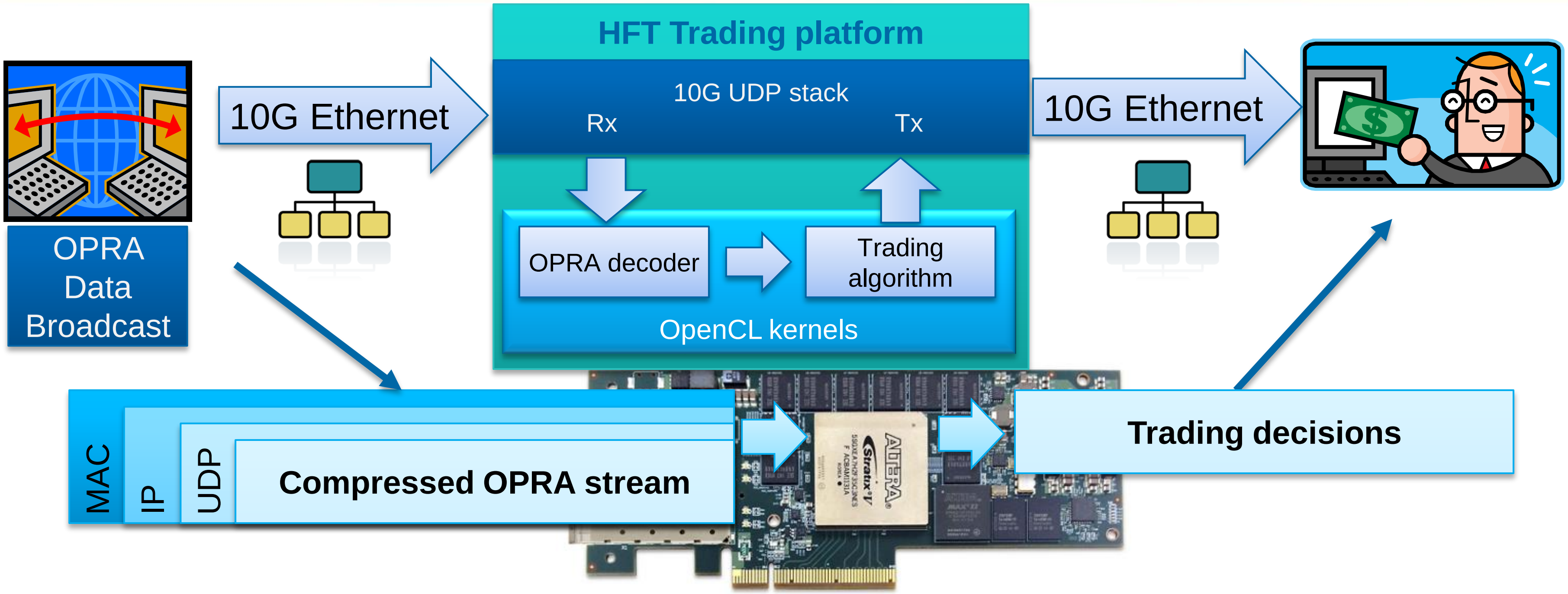
OPRA FAST decoder

Overview

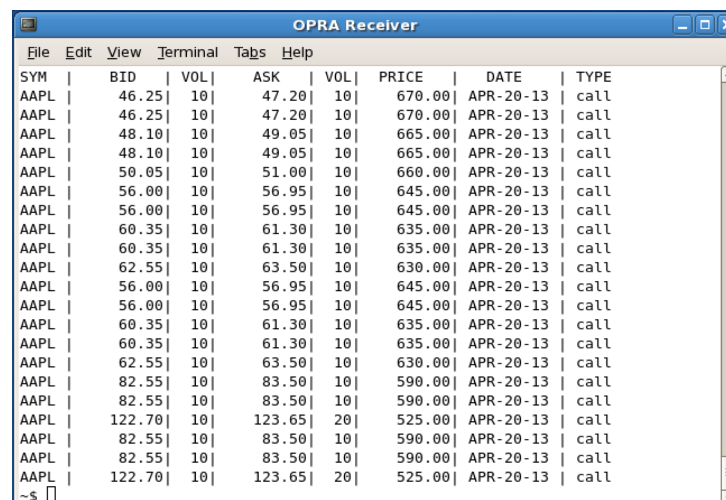
- **Application Overview**
- **Compiler Features**
  - IO Channels
  - Loop Pipelining
- **Kernel Implementation**
- **Host Implementation**

Application Overview

10G OPRA FAST



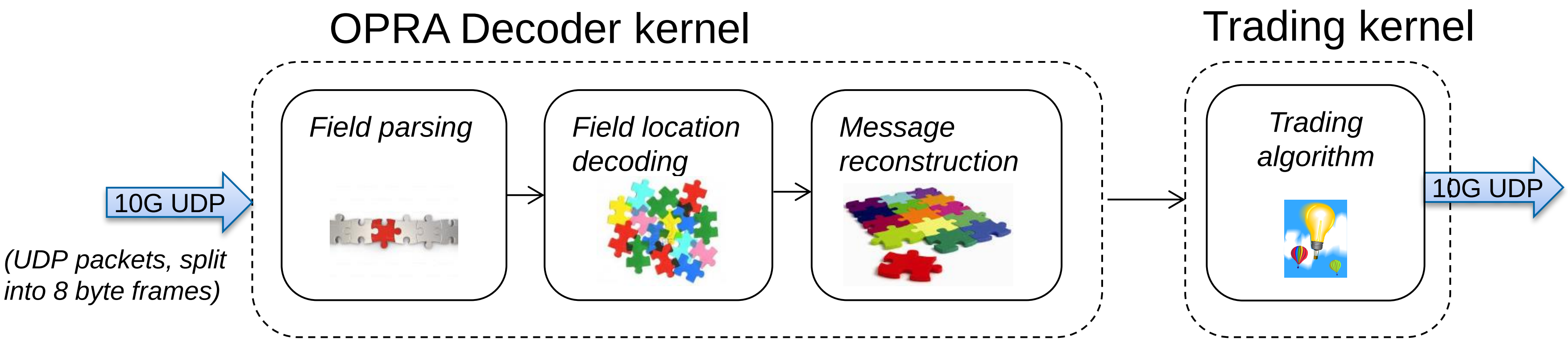
- 10G Ethernet UDP stream
- Headers removed by OpenCL kernel
- Decode compressed OPRA FAST data
- Reconstruct messages
- Platform with 10 GbE I/O channels and kernels using Altera’s OpenCL
- < .5 uS Latency



SYM	BID	VOL	ASK	VOL	PRICE	DATE	TYPE
AAPL	46.25	10	47.20	10	670.00	APR-20-13	call
AAPL	46.25	10	47.20	10	670.00	APR-20-13	call
AAPL	48.10	10	49.05	10	665.00	APR-20-13	call
AAPL	48.10	10	49.05	10	665.00	APR-20-13	call
AAPL	50.05	10	51.00	10	660.00	APR-20-13	call
AAPL	56.00	10	56.95	10	645.00	APR-20-13	call
AAPL	56.00	10	56.95	10	645.00	APR-20-13	call
AAPL	60.35	10	61.30	10	635.00	APR-20-13	call
AAPL	60.35	10	61.30	10	635.00	APR-20-13	call
AAPL	62.55	10	63.50	10	630.00	APR-20-13	call
AAPL	56.00	10	56.95	10	645.00	APR-20-13	call
AAPL	56.00	10	56.95	10	645.00	APR-20-13	call
AAPL	60.35	10	61.30	10	635.00	APR-20-13	call
AAPL	60.35	10	61.30	10	635.00	APR-20-13	call
AAPL	62.55	10	63.50	10	630.00	APR-20-13	call
AAPL	82.55	10	83.50	10	590.00	APR-20-13	call
AAPL	82.55	10	83.50	10	590.00	APR-20-13	call
AAPL	122.70	10	123.65	20	525.00	APR-20-13	call
AAPL	82.55	10	83.50	10	590.00	APR-20-13	call
AAPL	82.55	10	83.50	10	590.00	APR-20-13	call
AAPL	122.70	10	123.65	20	525.00	APR-20-13	call

~$

OPRA Application



- **Application structure**
  - OPRA FAST decoder kernel written in OpenCL
  - The decoder outputs the reconstructed OPRA messages
  - Output sent through a kernel to kernel channel for subsequent processing
- **Modularity**
  - Users can plug custom trading kernel
  - Currently using a dummy trading kernel for verification

OPRA FAST encoding

- **Each message contains several compressed fields**
- **Presence Map of each message tells us which fields of the current message are encoded**
  - Other fields are repeated from the last message or incremented
- **Compressed fields are variable number of bytes**
  - Each byte contains 7 bits of data and ‘stop bit’ for last byte
  - Uncompressed fields are fixed size

SOH Ver. Sequence Number of 1st Message # Msgs in Frame

01:02:30:30:30:30:30:31:38:36:33:33:30:30:31:

0c:fe:c8:ce:cf:a0:01:11:c9:2c:29:62:8d:03

MsgSz

Presence Map

Msg Type

Fields

MEASURABLE ADVANTAGE™

Trading algorithm

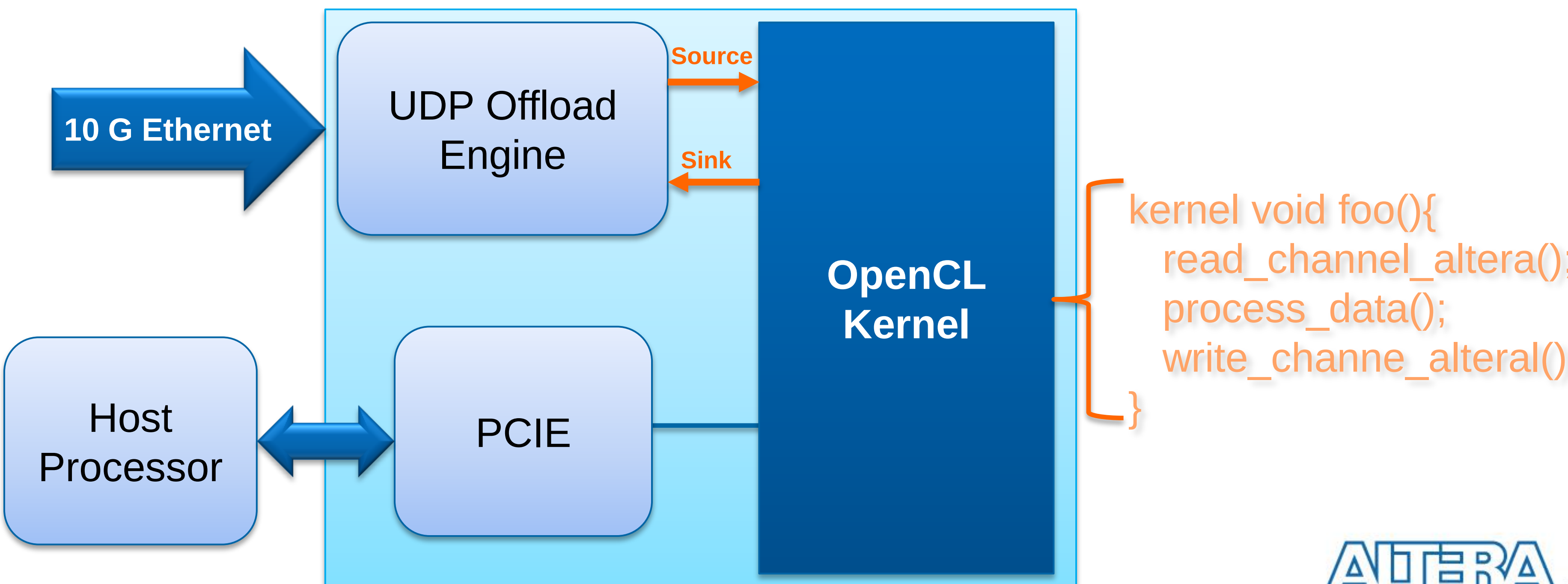
- **Dummy trading algorithm for verification purposes**
  - Demo goal to demonstrate parsing OPRA packets at line rate
  - Sending all decompressed data back through UDP would bottleneck the processing
- **Return a subset of the fields**
  - Required to throttle the amount of data sent out through UDP
  - Host configures which field range is returned

ALTERA®
MEASURABLE ADVANTAGE™

Compiler Features

Altera I/O Channels

- **Allows kernels to interface to outside world**
  - Simple API to read and write data from external sources
  - Available channel are board-specific, defined by board designer
- **This example uses IO channels connected to a UDP Offload Engine to communicate over 10 Gbps Ethernet**



ALTERA MEASURABLE ADVANTAGE™

Loop Pipelining: Loop Carried Dependencies

- **Loop-carried dependencies are dependencies where one iteration of the loop depends upon the results of another iteration of the loop**

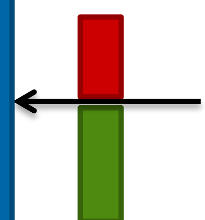
```
__kernel void generate_rngs(ulong num_rnds)
{
  t_state_vector state = initial_state();
  for (ulong i=0; i<num_rnds; i++) {
    state = generate_next_state( state );
    unit y = extract_random_number( state );
    write_channel_altera(RANDOM_STREAM, y);
  }
}
```

- **The variable state in iteration 1 depends on the value from iteration 0. Similarly, iteration 2 depends on the value from iteration 1, etc.**

Loop Pipelining (2)

- **To achieve acceleration, we can *pipeline* each iteration of a loop containing loop carried dependencies**
  - Analyze any dependencies between iterations
  - Schedule these operations
  - Launch the next iteration as soon as possible

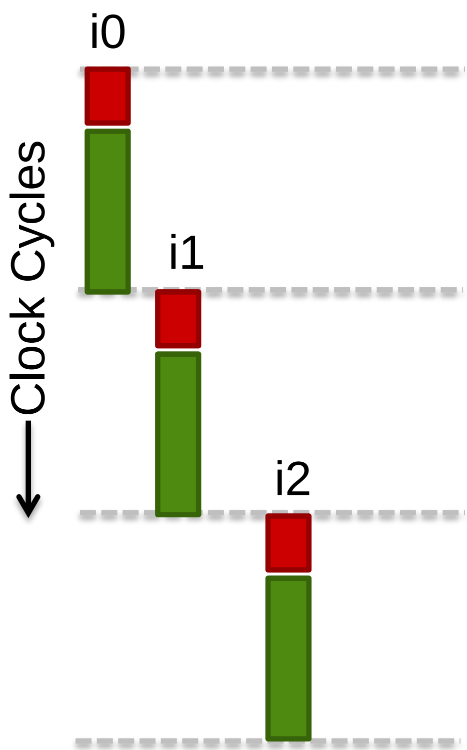
```
__kernel void generate_rngs(ulong num_rnds)
{
  t_state_vector state = initial_state();
  for (ulong i=0; i<num_rnds; i++) {
    state = generate_next_state( state );
    unit y = extract_random_number( state );
    write_channel_altera(RANDOM_STREAM, y);
  }
}
```



At this point, we can launch the next iteration

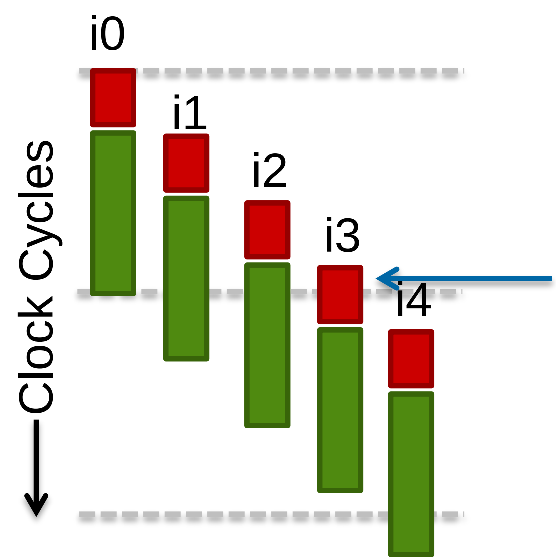
Loop Pipelining Example

- **No Loop Pipelining**



No Overlap of Iterations!

- **With Loop Pipelining**



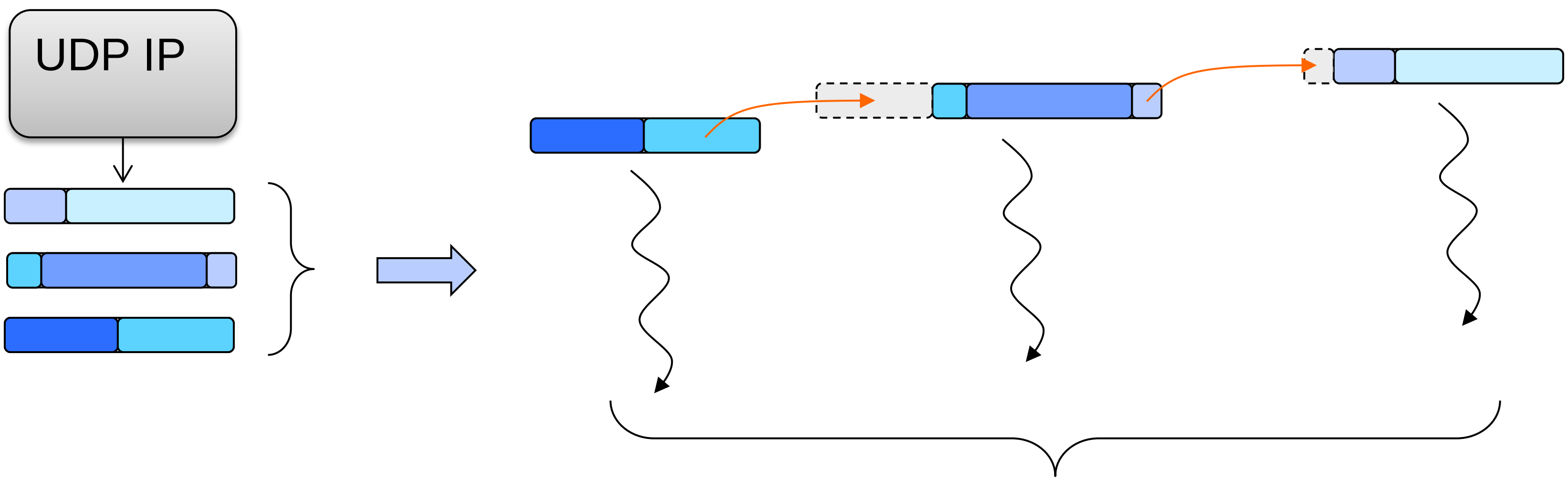
Looks almost like ND-range thread execution!

Finishes Faster because Iterations Are Overlapped

Kernel Implementation

OPRA Field Parser

- **UDP packet split into fixed length frames**
  - OPRA FAST processing expressed as a loop
  - Allow the OpenCL compiler to extract pipeline parallelism from loop iterations
  - Each iteration processes one frame
- **Fields may span across multiple frames**
  - Loop carried dependencies
  - The compiler understands and generates efficient hardware in the presence of dependencies



Pipelined parallel processing of frames

OPRA Decoding Loop

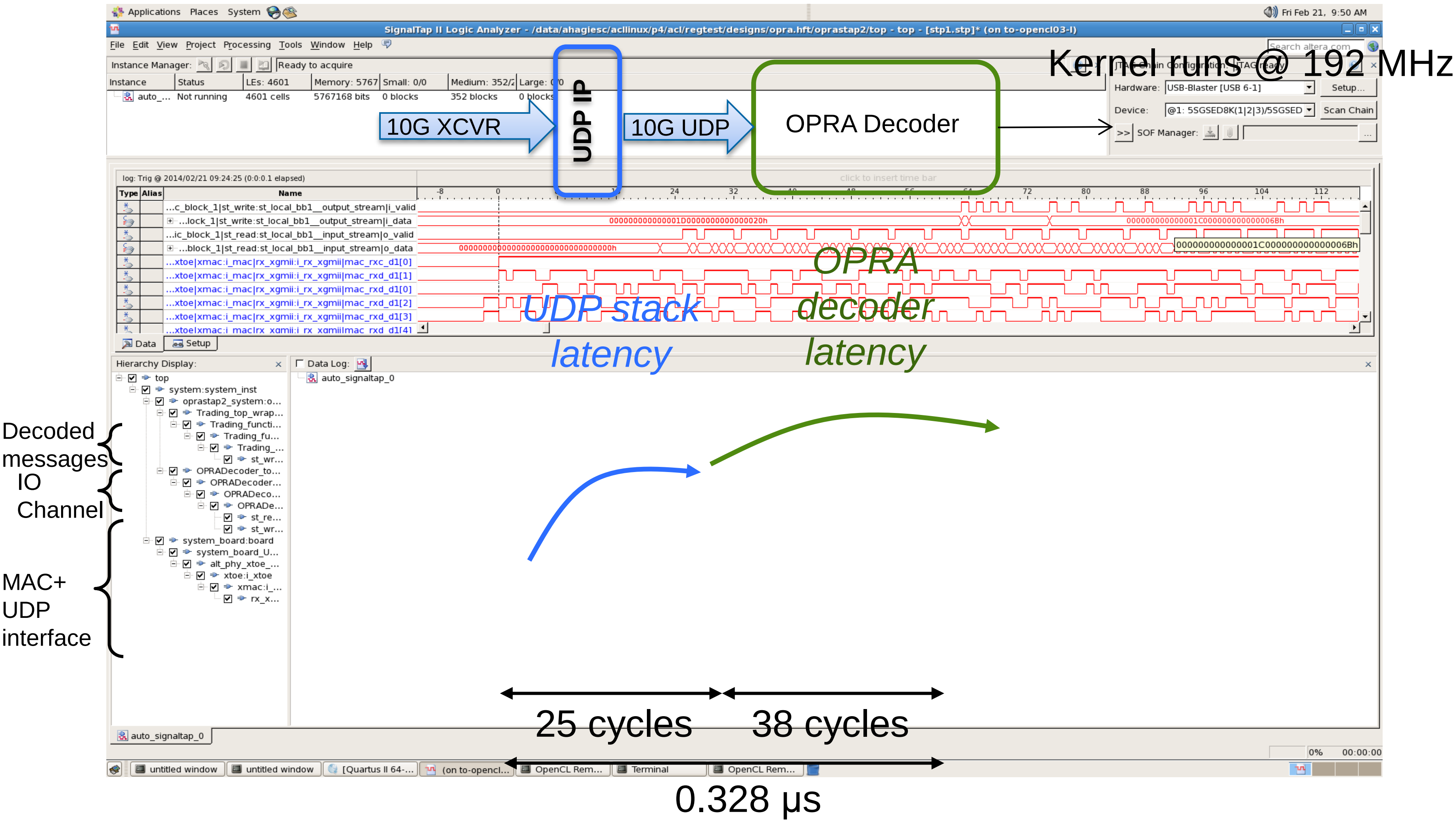
- **Every loop iteration, 8 byte frame of data is read from the UDP interface**
  - Except in rare case when there is data left from last frame
- **The multiple fields of the OPRA message are constructed from over several iterations of the loop**
  - As each field is decoded, it is written to a location in the message structure
- **When the message is fully parsed, it is sent to the trading kernel via a kernel-to-kernel channel**
  - Non-present fields (according to the Presense Map) are maintained from last message or incremented
- **Loop carried dependencies are minimized to allow one iteration to launch every cycle**
  - If hardware frequency >= 156.25 MHz , we can saturate the 10G connection

ALTERA® MEASURABLE ADVANTAGE™

Other Kernels

- UDP IO data interface is 16 bytes wide, while the Decoder kernel takes 8 bytes, two kernels are used as 16-to-8 byte adaptors
- Two kernels are used to choose between sending data from the IO channels or from global memory

Latency measurement



Host Implementation

Host Program

- **OPRA Kernels can either communicate over UDP IO Channels, or by reading and writing to memory**
- **If UDP IO channels are being used, the host will send and receive data over UDP sockets to the IP address of the FPGA card**
  - 10G Ethernet card should be installed in host PC, connected to FPGA card
- **The host program forks into two processes, which allows the host to send and receive data over UDP independently**
- **Tested using Solarflare network interface card**
  - OpenOnload drivers are used to accelerate UDP transfers, and are needed to consistently saturate the 10G interface

MEASURABLE ADVANTAGE™

Tips for tuning host to achieve line rate

- **Host OS is not a real time OS**
  - System jitter can cause packet loss on the host
  - Do not run any unnecessary services or applications
- **Run app through demo.sh script**
  - Does some driver tuning to minimize overhead
  - This proves sufficient on the test machine we have on our side

    (Intel(R) Core(TM)2 Quad CPU   Q9550 @ 2.83GHz)

MEASURABLE ADVANTAGE™

Running the example design

host/opra [mode] [UDP frames]

- **[mode] configures where the data source and destination are located**
  - memory input → memory output (default, 0)
  - UDP input → memory output (1)
  - Memory input → UDP output (2)
  - UDP input →UDP output (3)
- **[UDP frames] specifies how many frames to transmit**
  - The frames are read from a pcap file that comes with the example

MEASURABLE ADVANTAGE™

Demo output

> host/opra 3 300000

Using AOCX: opra.aocx
Short memory run --> results.txt
Long run, use as reference for subsequent runs
Use compact UDP packets for better throughput
All integrity checks are DONE.

Memory based runs for integrity checks and generating reference data

Performance testing: UDP Rx --> OPRA decoder --> UDP Tx

Verified field 0
Verified field 1
Verified field 2
Verified field 3
Verified field 4

Multiple UDP runs
In each run, configure the dummy trading algo to return one field of the message

OPRA stream throughput: 9.417523 Gbit / s
(99.7 % of theoretical maximum of 9.446Gb/s)

Max performance measured across all runs

ALTERA MEASURABLE ADVANTAGE™

Thank You

MEASURABLE ADVANTAGE™