MDP 3.0 TICKERPLANT
What is MDP3.0

MDP3.0 is a completely new data feed implementation by the CME Group. Sends Incremental Market Updates among a variety of other information. Designed to be super quick and efficient.
The encoded FIX transmission is sent in a packet structured as follows:

- **Packet header** - contains packet sequence number, sending time.
- **Message Size** - field indicating size of message.
- **Message header** - contains block length, TemplateID, SchemaID, and Version.
- **FIX header** - indicates FIX message type (example: 35=X)
- **FIX message body** - event driven business data such as book updates and trade summary.
Sample Message - Market Data Incremental Refresh (35=X)

```xml
<!-- MarketDataIncrementalRefresh (35=X) message -->
<sbe:message name="MarketDataIncrementalRefreshTrades" id="02" fixMsgType="X" description="Trade">
    <field name="TransactTime" id="60" fixUsage="UTCTimestamp" type="UTCTimestamp" timeUnit="nanosecond" />
    <field name="EventTimeDelta" id="37704" fixUsage="String" type="uint16" />
    <field name="MatchEventIndicator" id="5799" fixUsage="char" type="MatchEventIndicator" />
    <field name="NoMDEntries" id="268" fixUsage="NumInGroup" type="NumInGroup" groupNam=e="MDIncGrp" />
    <group name="MDIncGrp">
        <field name="MDUpdateAction" id="279" fixUsage="char" type="MDUpdateAction" />
        <field name="MDEntryType" id="269" fixUsage="char" type="MDEntryType" constant="2" />
        <field name="SecurityID" id="48" fixUsage="String" type="UniqueID" />
        <field name="RptSeq" id="83" fixUsage="int" type="SeqNum" />
        <field name="MDEntryPx" id="270" fixUsage="Price" type="Price" />
        <field name="MDEntrySize" id="271" fixUsage="Qty" type="Qty" />
        <field name="NumberOfOrders" id="346" fixUsage="int" type="uint16" />
        <field name="AgressorSide" id="5797" fixUsage="int" type="AgressorSide" />
    </group>
</sbe:message>
```
Our Project

We decode market data incremental refresh messages sent from the CME Group.

Using this data we generate our own version of order-books for specific securities.

We then send out snapshots of these order books at regular intervals.
Software Implementation

- Python Code
- Book Builder

```python
def shift(l, n, r1):
    if r1=="r":
        return l[n:] + l[:n]
    return l[:n] + l[n:]

class Order():
    def __init__(self, price, quantity):
        self.price = price
        self.quantity = quantity

    def __str__(self):
        return `self.quantity` + ": $` + `self.price``

class FullOrderBook:
    def __init__(self):
        self.askBook = Book()
        self.bidBook = Book()

    def __str__(self):
        askString = 'ASK
----------
'
        bidString = 'BID
----------
'

            askString += `bookOrder.quantity` + ": $` + `bookOrder.price` + `\n`

            bidString += `bookOrder.quantity` + ": $` + `bookOrder.price` + `\n`
```
```
General Architecture

CME sends Data → Packetizer reads Input → FIFO stores data → Parser decodes

Order Book Updates → FIFO stores snapshot → Snapshot Stream → Sent to Network
Sample data fed as input → Avalon MM2ST → Packetizer reads Input → FIFO stores data → Parser decodes

Order Book Updates → Avalon ST2MM
Hardware Implementation
Packetizer
Our parser reads in data from the FIFO

Message headers are always multiples of 64 bits

But each message can contain multiple entries.

Each entry is typically 214 bits (which is not a multiple of 64)

This requires us to keep track of the entry offset

Simple Equation:

\[ \text{Offset} = (\text{Offset} + 40) \mod 64 \]
FIFO

- Buffer between components
- 64-bits wide
- 256 blocks deep
Figure 1

Sample Dequeue
Order Book

- 10 levels of Bid and Ask prices

Bid Book

<table>
<thead>
<tr>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>96.30</td>
</tr>
<tr>
<td>96.40</td>
</tr>
<tr>
<td>96.50</td>
</tr>
<tr>
<td>96.55</td>
</tr>
<tr>
<td>96.65</td>
</tr>
<tr>
<td>96.70</td>
</tr>
<tr>
<td>96.80</td>
</tr>
<tr>
<td>96.85</td>
</tr>
<tr>
<td>97.00</td>
</tr>
</tbody>
</table>
Challenges

- Oversimplified Initial sample data
- Needed a robust testing suite
- Too much trust in Modelsim
- New data format
Lessons Learned

- More robust Modelsim tests
- The initial design should have been more macro-focused
- Clarify confusing financial concepts earlier
Future Work

- Implied Orders
  - Implied “IN”
    - Order In spread from outright
  - Implied “OUT”
    - Order In the outright from spread

- Our future work on the project aims to be able to read the saved Order Books across different months to create Implied books
Conclusion

- Thanks for all the help!
  - Prof Edwards & Lariviere
  - Qiushi Ding