

Next-gen Market Access: ULL tick-to-trade made easier with Enyx

STAC London | November 2018



Full Hardware Feed Handler



Enyx | A Global Offering

Full Hardware Feed Handler



Bandwidth Management



Enyx | A Global Offering

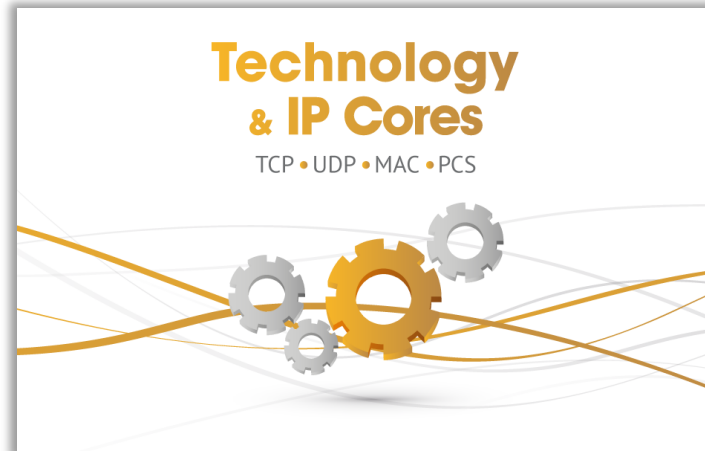
Full Hardware Feed Handler



Bandwidth Management



Development Framework



Enyx | A Global Offering

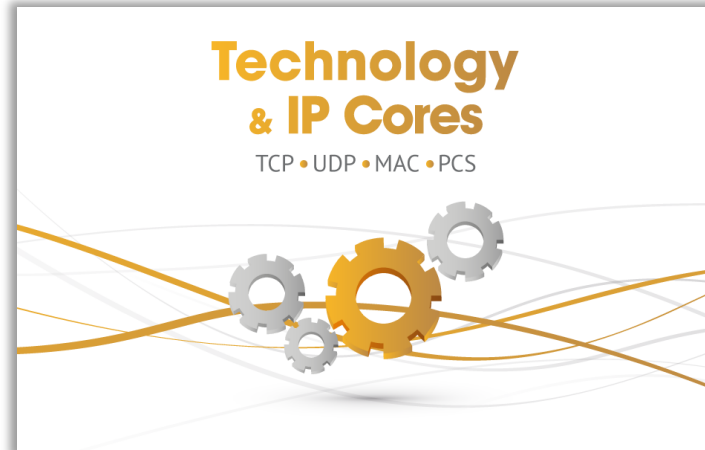
Full Hardware Feed Handler



Bandwidth Management



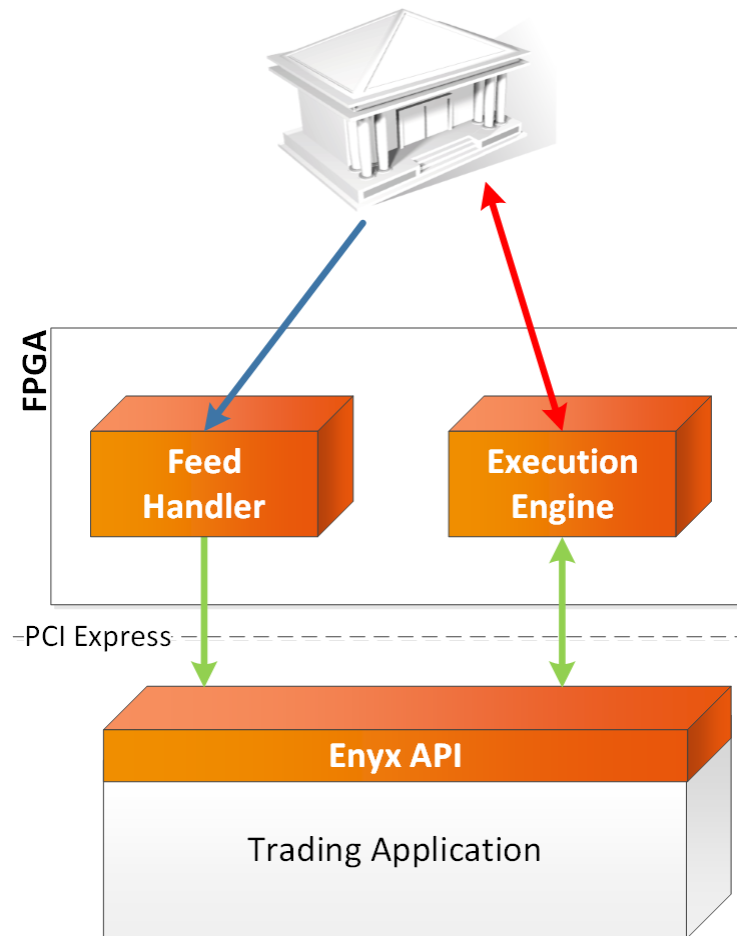
Development Framework



Low Latency Market Access



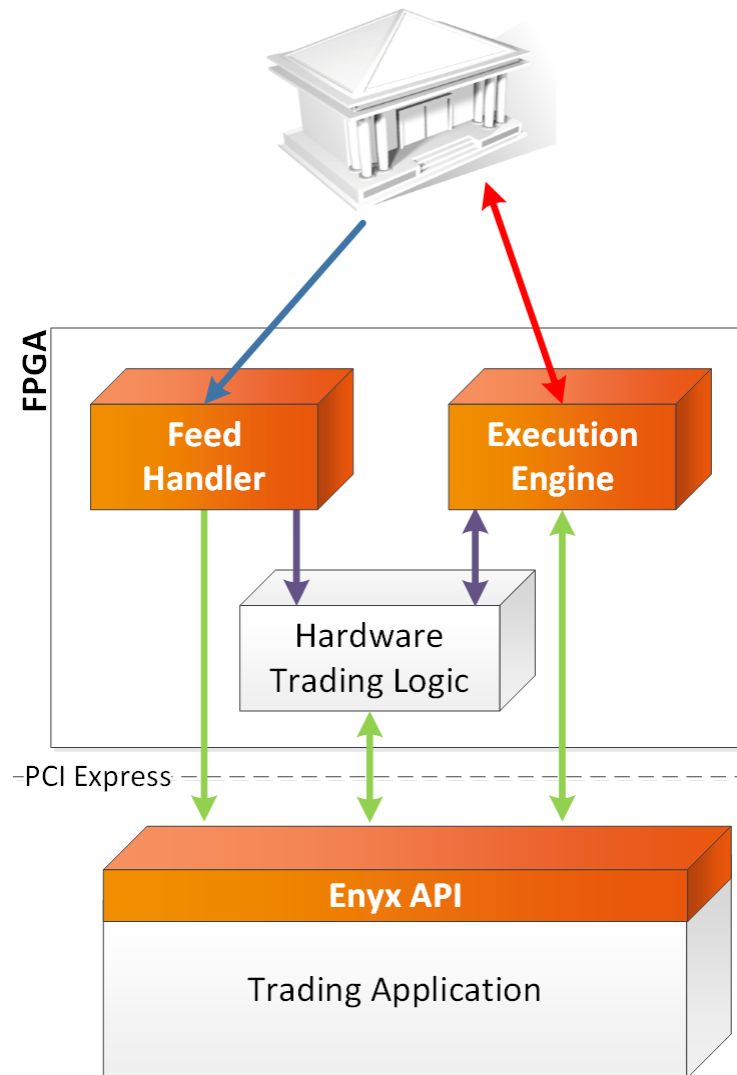
nxAccess | A Full Hardware Tick-to-Trade



Advantages

- **Full hardware Feed Handler and Execution Engine**
- **C++ API** to interface with your software application
- Supports **every market data message** and **every type of order**

nxAccess | A Full Hardware Tick-to-Trade



Advantages

- **Full hardware Feed Handler and Execution engine**
- **C++ API** to interface with your software application
- Supports **every market data message** and **every type of order**
- **Autonomous development of your hardware logic** using our Hardware Development Framework
- Compatible with our **50+ venues** coverage

- Integration with your trading infrastructure using our C++ software API
- Development of your hardware logic with our Hardware Framework:
 - Using **VHDL** or **Verilog**
 - Using **HLS** and the **Vivado HLS** framework



Trading Decision on Trade Summary:

Advantages

- High level efficient language using C++ type syntax
- Sequential description but **parallel execution**
- Can be unit-tested in a software manner

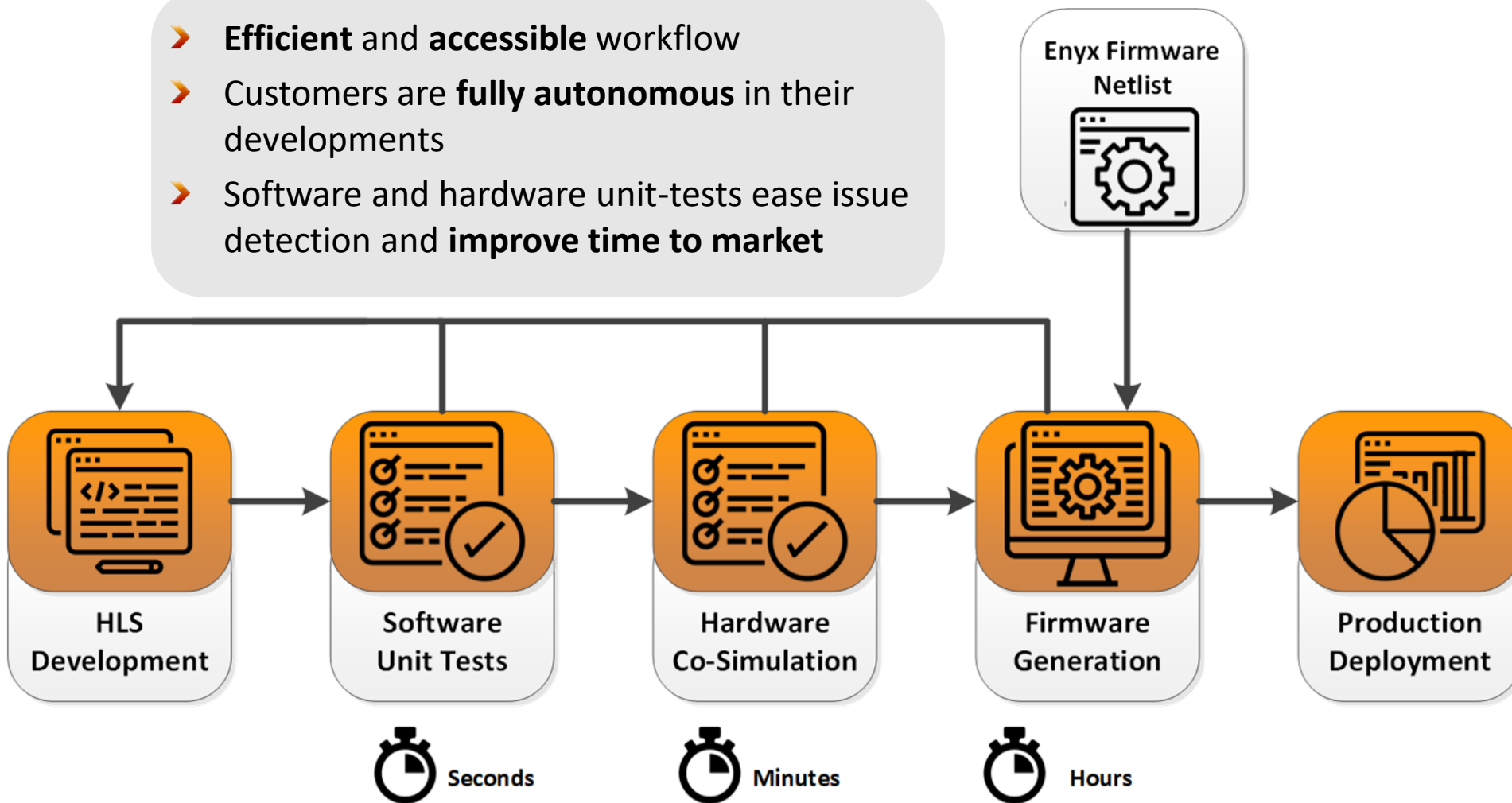
```
Void handle_trade_summary(nxbus_word_t & nxbus_word){
    if (is_instr_subscribed(nxbus_word)){
        nxbus_instr_id_t const instr_id = get_instr_id(nxbus_word);
        nxbus_price_t const trade_price = get_price(nxbus_word);

        switch (agressor_side(nxbus_word)){
            case BUY_SIDE:
                if (trade_price < order_book_bid[instr_id][0].price){
                    price_collar_scenario(instr_id, trade_price, TRUE);
                }
            case SELL_SIDE:
                if (trade_price > order_book_ask[instr_id][0].price){
                    price_collar_scenario(instr_id, trade_price, FALSE);
                }
            case default:
                assert(!"Unexpected agressor side");
        }
    }
}
```

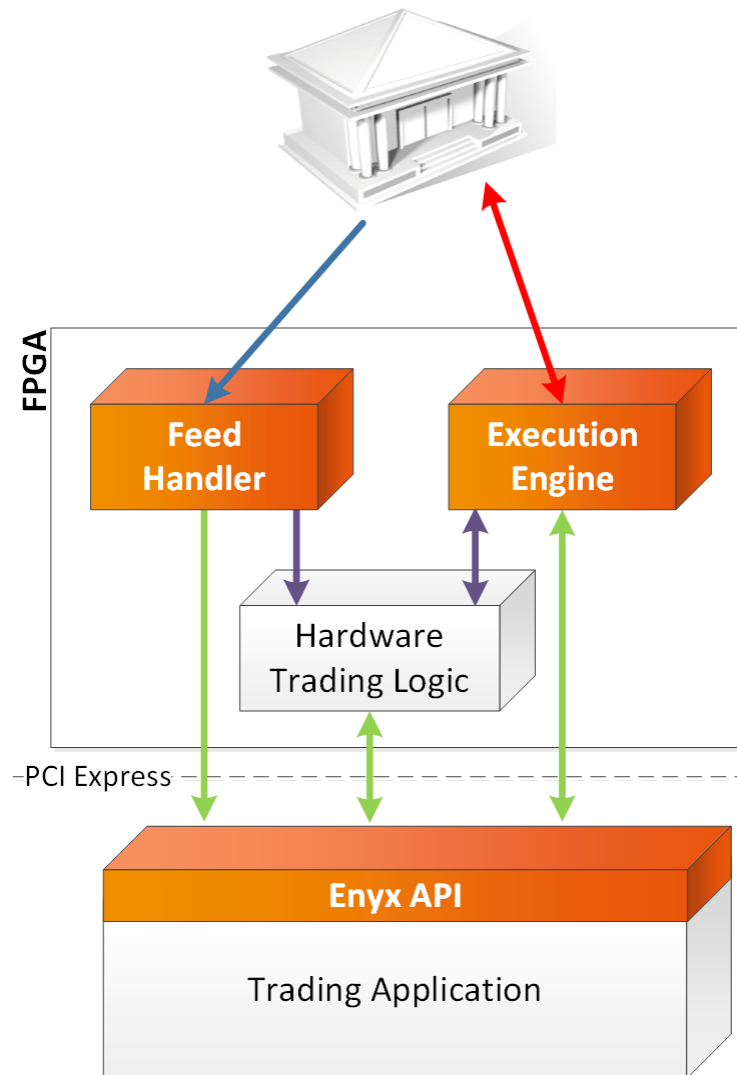
nxAccess | HLS Development Workflow

Advantages

- **Efficient** and **accessible** workflow
- Customers are **fully autonomous** in their developments
- Software and hardware unit-tests ease issue detection and **improve time to market**



nxAccess | A Full Hardware Tick-to-Trade



Summary

- Solution compatible with our **coverage of 50+ venues**
- **Full featured Feed Handler and Execution Engine** simplifying the development of your algorithm
- Open Hardware and Software API allows **autonomous development** and **preserves the customer's IP**

Thank you

**Tick the Enyx box to request
more information and an evaluation**



Market Data Update Management:

```
Void handle_nxbus_update(hls::stream<nxbus_word> & axi_ingress_nxbus) {
    #pragma HLS INLINE recursive
    #pragma HLS PIPELINE enable_flush

    if (! nxbus_update_in.empty()) {
        // valid data word on the ingress AXI stream bus
        nxbus_command_t const nxbus_command_in = axi_ingress_nxbus.read();

        switch (nxbus_command_in.opcode) {
            case NXBUS_OPCODE_BOOK_UPDATE:
            case NXBUS_OPCODE_BOOK_SNAPSHOT:
                handle_book_update(nxbus_command_in.data);
                break;

            case NXBUS_OPCODE_TRADE_SUMMARY:
            case NXBUS_OPCODE_TRADE_SUMMARY_BREAK:
            case NXBUS_OPCODE_TRADE_SUMMARY_CORRECTION:
                handle_trade_summary(nxbus_command_in.data);
                break;

            default:
                break;
        }
    }
}
```

Book Update:

```
Void handle_book_update(nxbus_word_t & nxbus_word) {
    if (is_instr_subscribed(nxbus_word)) {
        Nxbus_instr_id_t const instr_id = get_instr_id(nxbus_word);
        order_book_index_t const index = get_book_index(nxbus_word);
        order_book_limit_t const limit = get_book_limit(nxbus_word);

        switch (nxbus_word.side){
            case BUY_SIDE:
                order_book_bid[instr_id][index] = limit;

            case SELL_SIDE:
                order_book_ask[instr_id][index] = limit;

            case default:
                assert(!"Unexpected side");
        }
    }
}
```

Trading Decision on Trade Summary:

```
Void handle_trade_summary(nxbus_word_t & nxbus_word) {
    if (is_instr_subscribed(nxbus_word)) {
        nxbus_instr_id_t const instr_id = get_instr_id(nxbus_word);
        nxbus_price_t const trade_price = get_price(nxbus_word);

        switch (agressor_side(nxbus_word)) {
            case BUY_SIDE:
                if (trade_price < order_book_bid[instr_id][0].price) {
                    price_collar_scenario(instr_id, trade_price, TRUE);
                }
            case SELL_SIDE:
                if (trade_price > order_book_ask[instr_id][0].price) {
                    price_collar_scenario(instr_id, trade_price, FALSE);
                }
            case default:
                assert(!"Unexpected aggressor side");
        }
    }
}
```


Trigger “Modify Order” with New Price:

```
Void price_collar_scenario(nxbus_instr_id_t & instr_id, nxbus_price_t
trade_price, bool is_bid){
    buffer_id_t const buffer_id = get_buffer_id(instr_id, is_bid);
    order_price_t const order_price;

    // Follow the price of the last trade
    order_price = get_buffer_id(trade_price)

    // Send a modify order with the new price
    trigger_order(buffer_id, order_price);
}
```