

KeyStone SoC Training

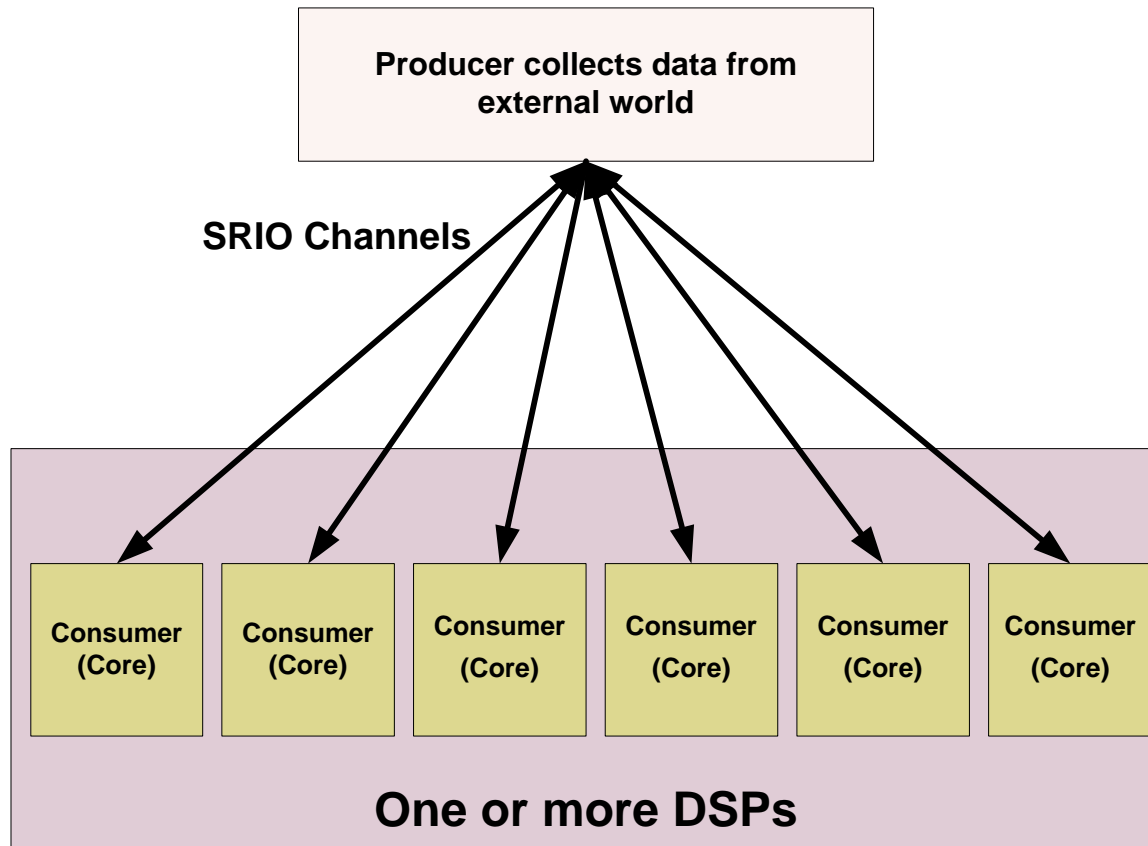
SRIO Demo: Board-to-Board

Multicore Application Team

Agenda

- Model
- Protocol
- Configuration
- Application Algorithm
- Build and Run

The Model



*Producer = Master
Consumer = Slave*

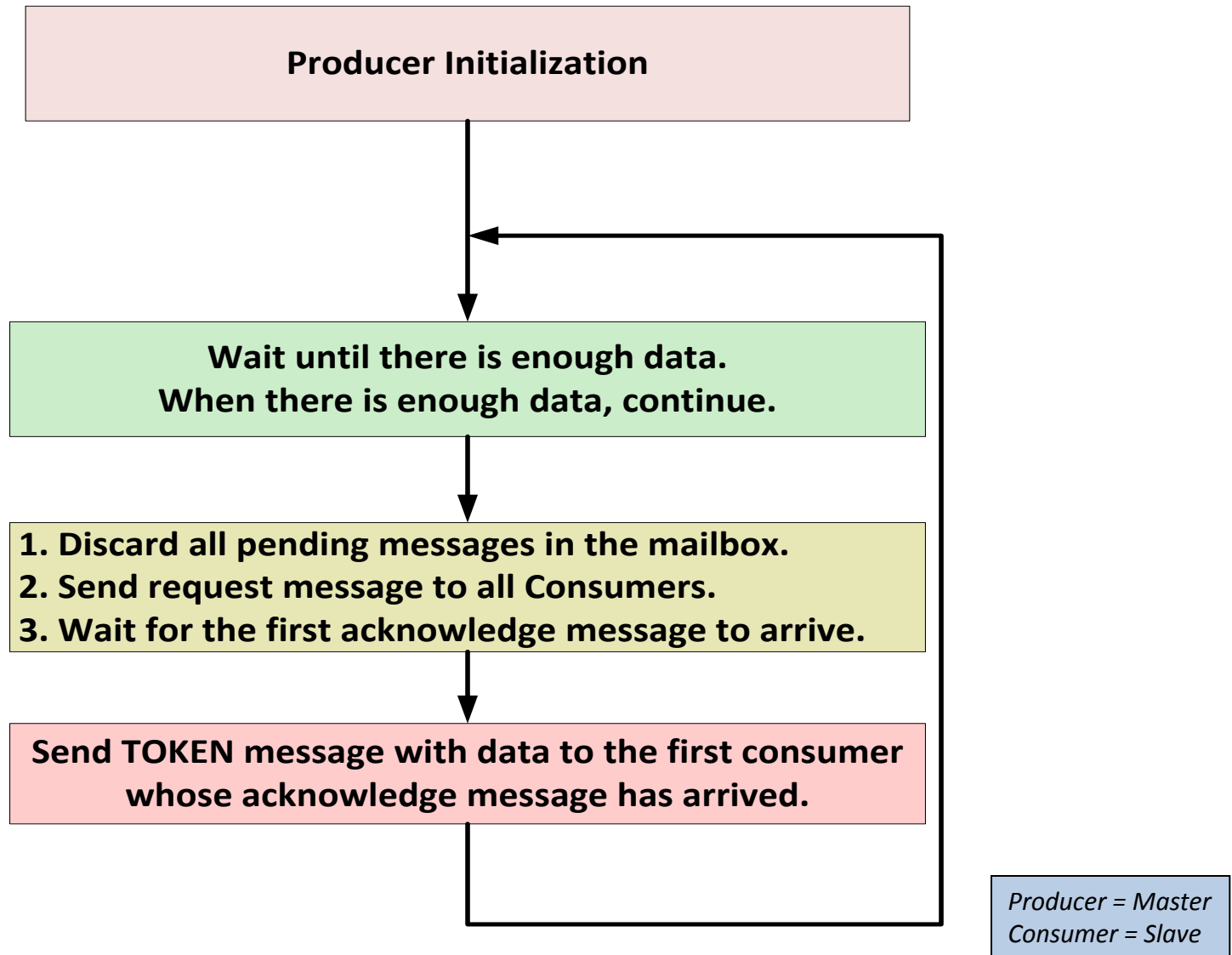
Requirements:

- Efficiency – Not fairness
- Minimize master logic
- Master is not aware of structure of internal cores

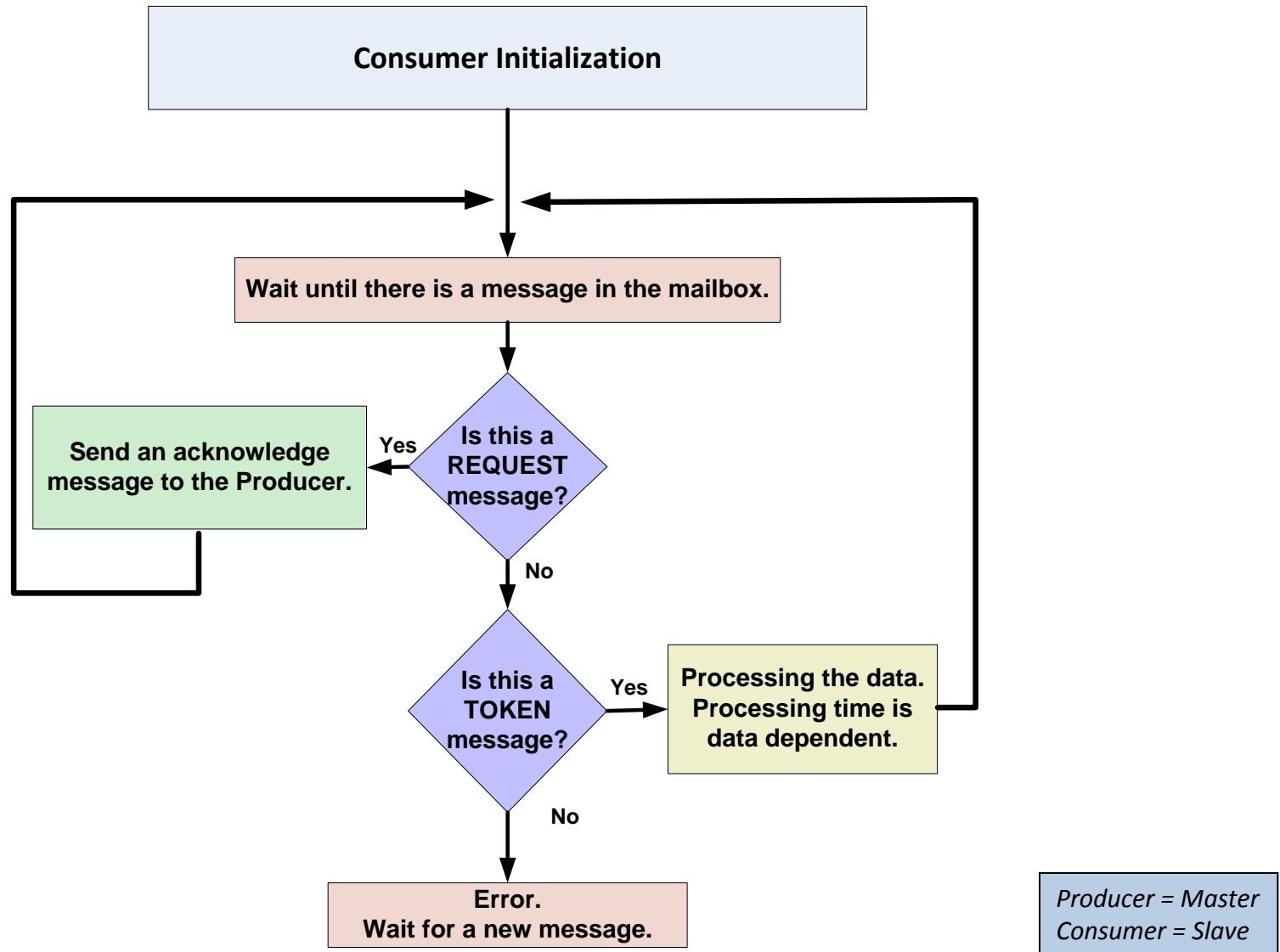
Agenda

- Model
- Protocol
- Configuration
- Application Algorithm
- Build and Run

Producer (Master) Protocol



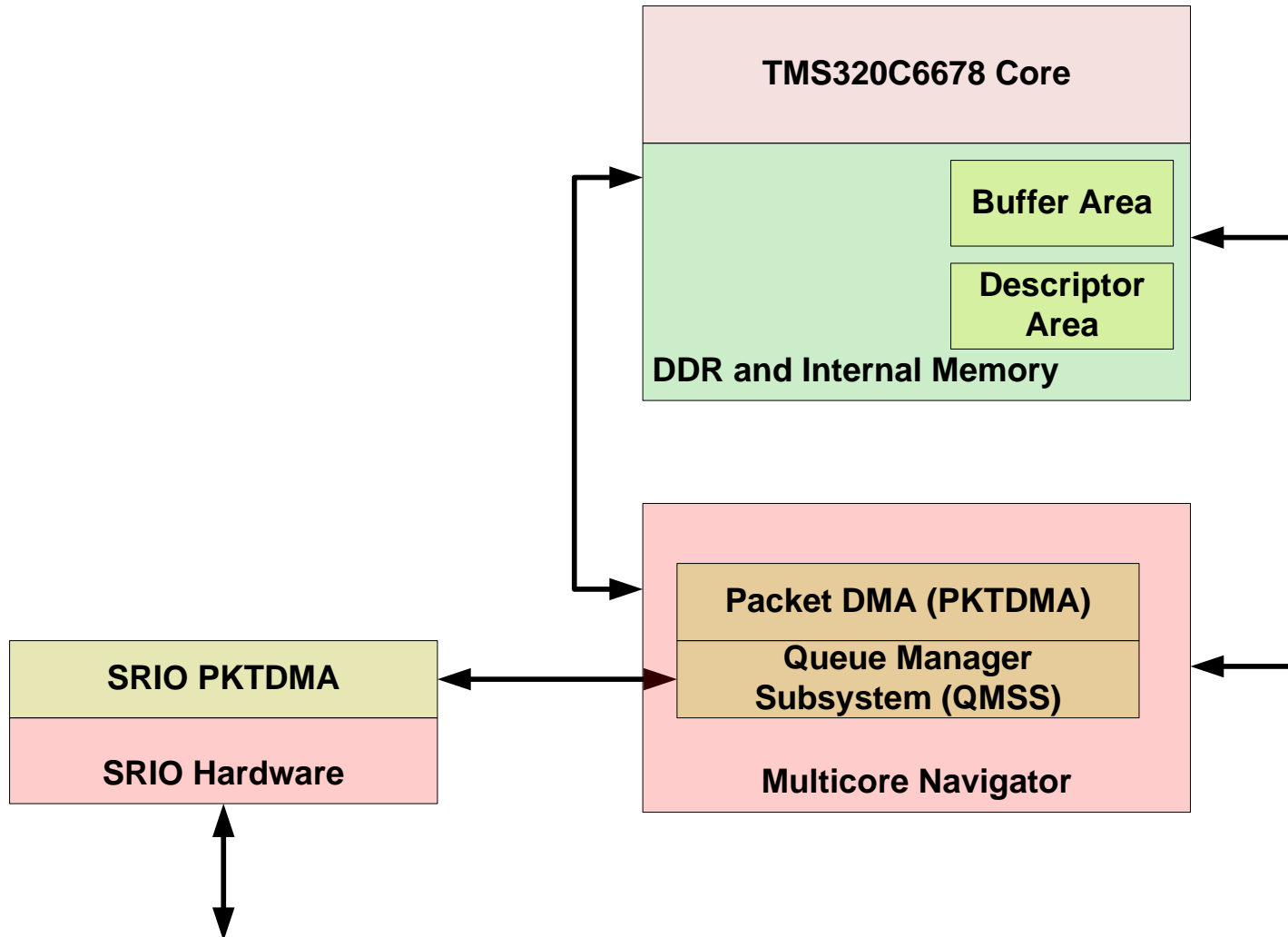
Consumer (Slave) Protocol



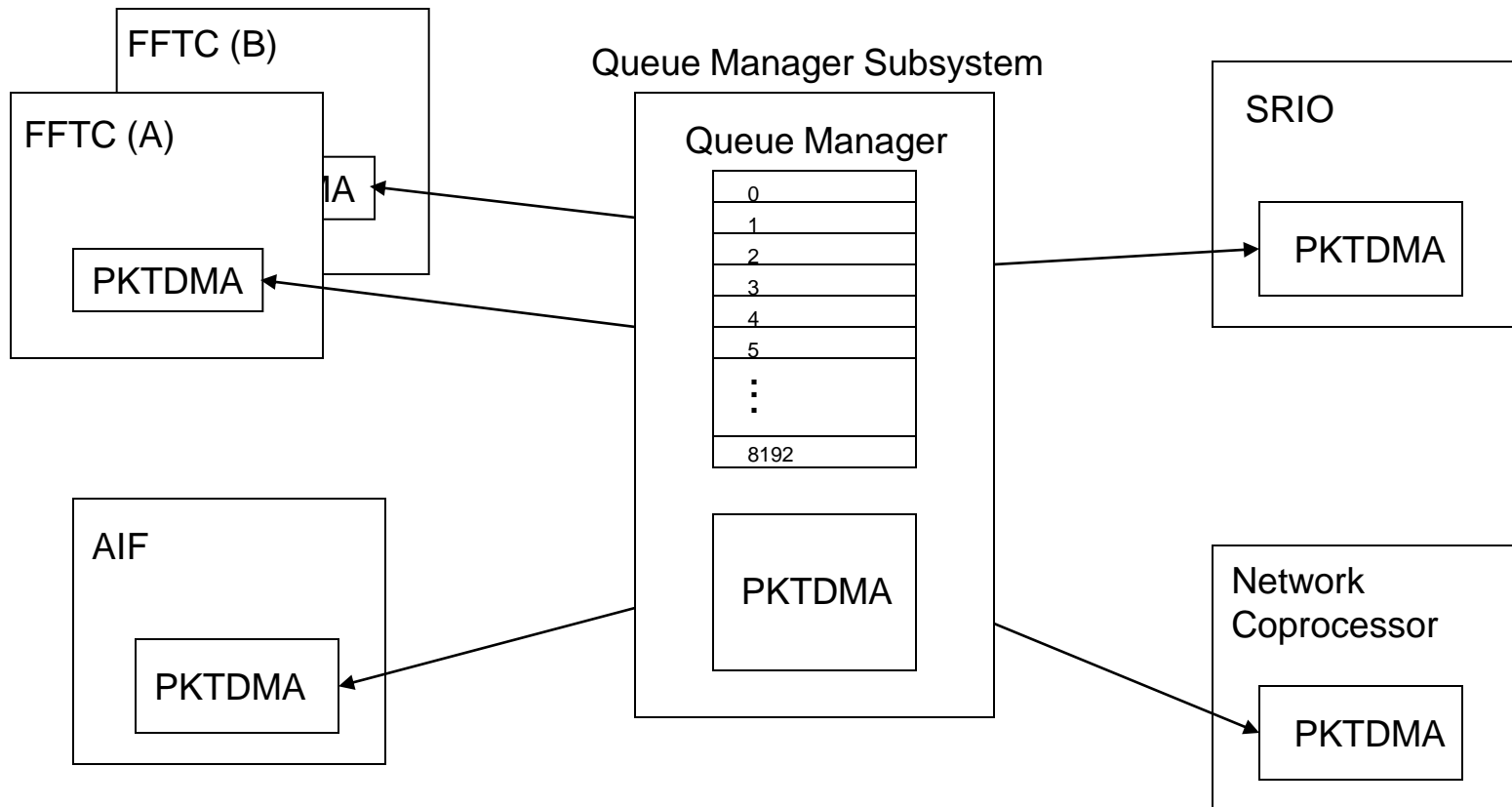
Agenda

- Model
- Protocol
- Configuration
- Application Algorithm
- Build and Run

Hardware Components



Packet DMA Topology



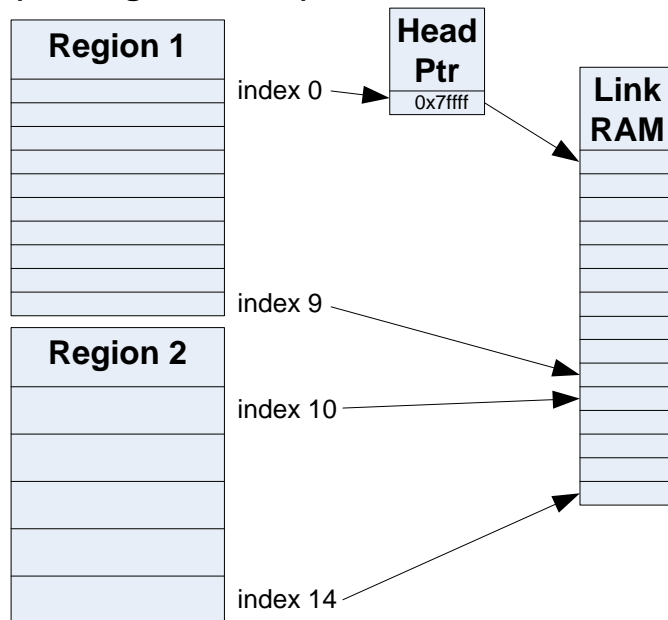
Multiple Packet DMA instances in KeyStone devices:

- PA and SRIO instances for all KeyStone devices.
- AIF2 and FFTC (A and B) instances are only in KeyStone devices for wireless applications.

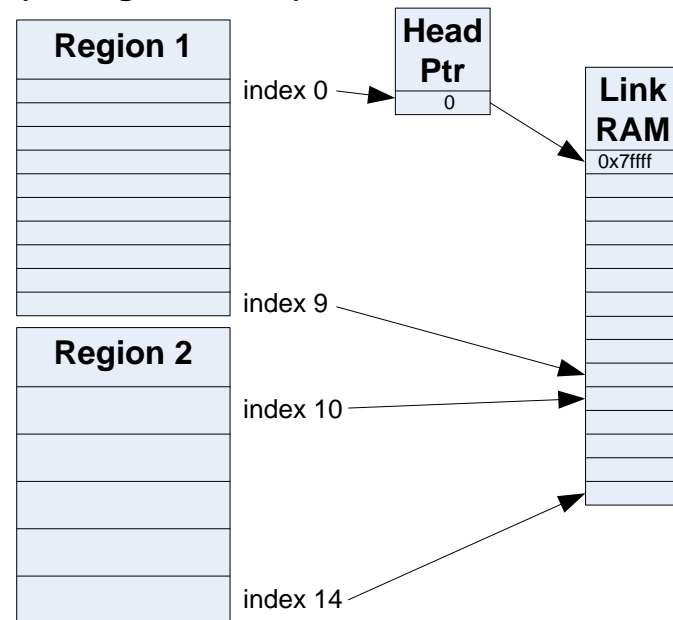
QMSS Descriptors Queuing

- The Queue Manager maintains a head pointer for each queue, which are initialized to be empty.

Push index 0 to an empty queue
(starting condition)

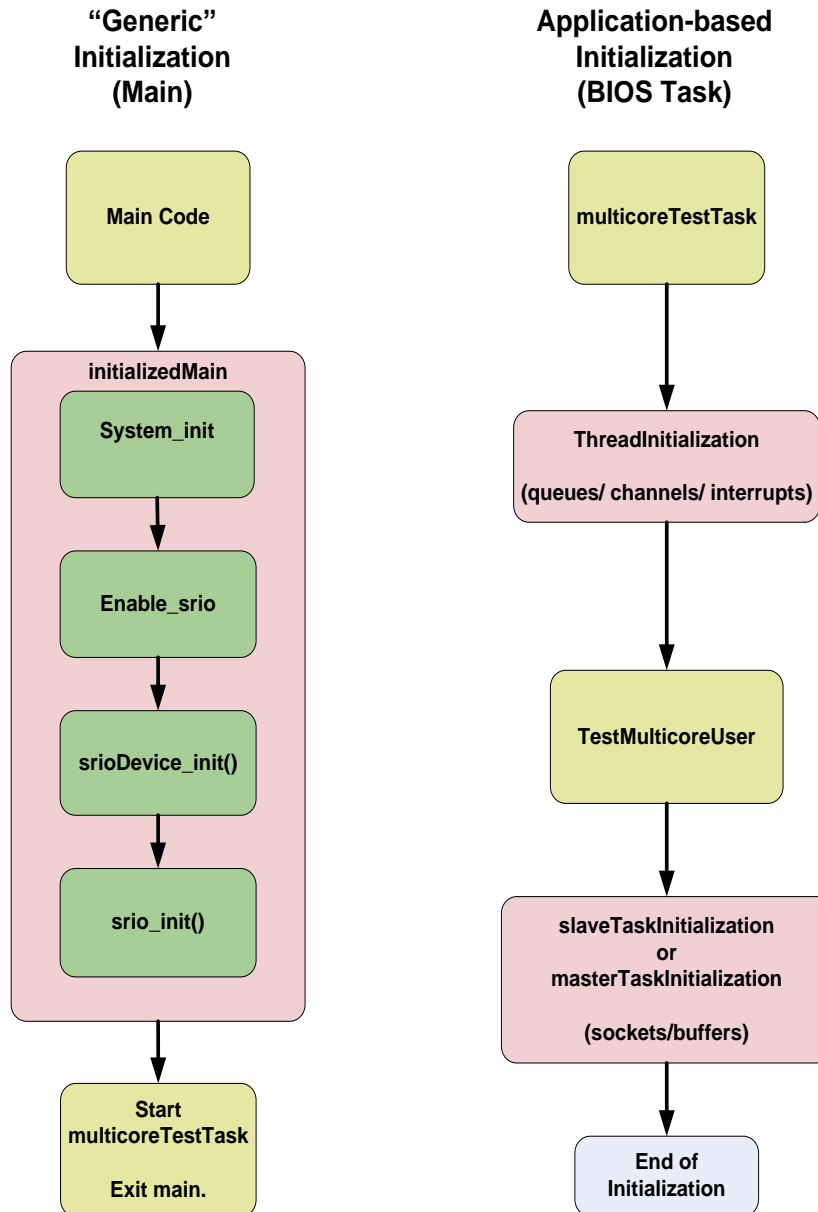


Push index 0 to an empty queue
(ending condition)



- We actually do not push indexes; We push descriptor addresses. The QM converts addresses to indexes.

Configuration/Initialization Flow



Configuration Steps:

1. QMSS
2. Generic PKTDMA
3. QMSS PKTDMA
4. SRIO
5. SRIO PKTDMA
6. Sockets

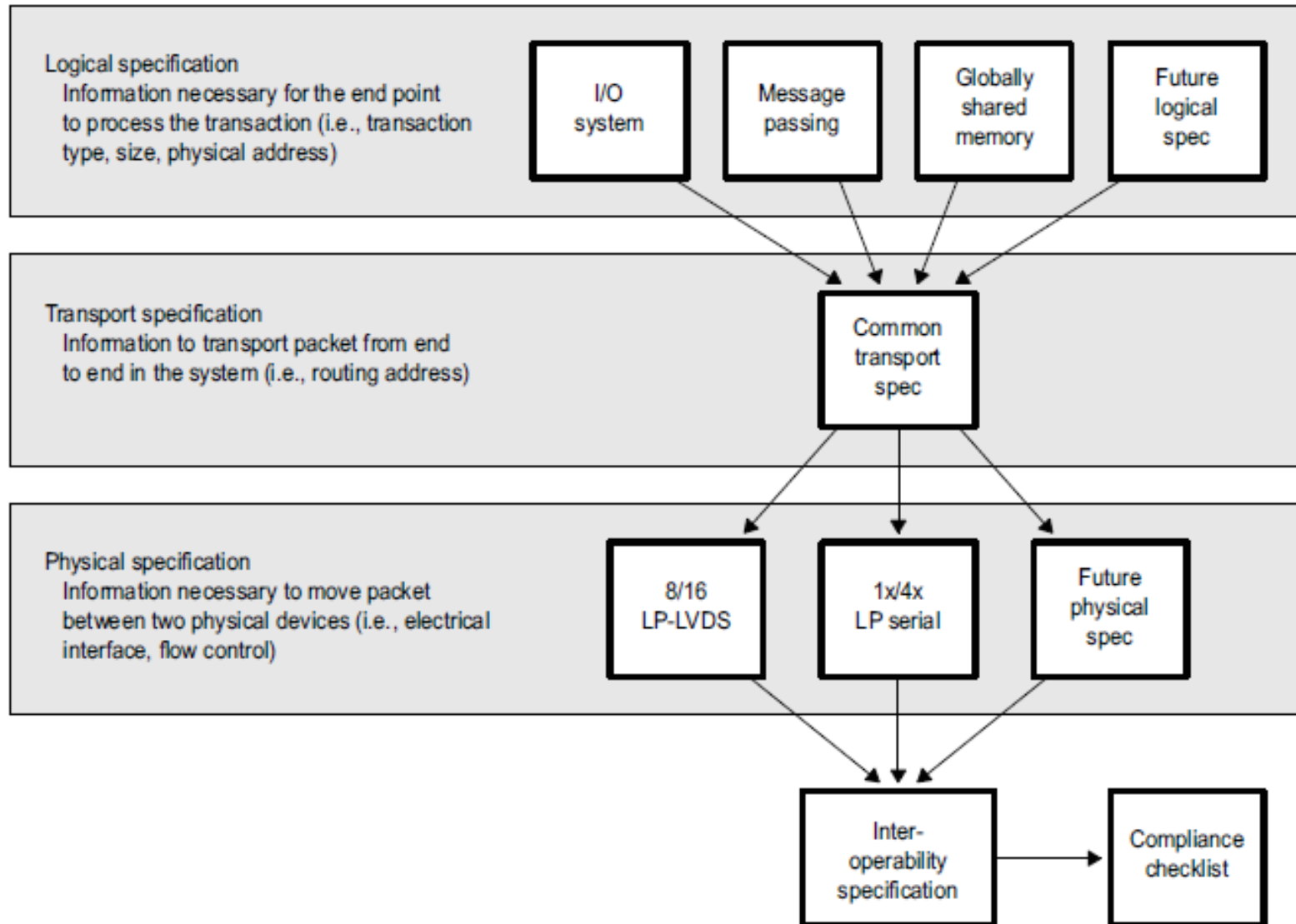
QMSS Initialization

- **Qmss_init (qmss_drv.c)**
 - Number and location of the link RAM
 - Number of descriptors
 - APDSP firmware
 - Set global structure qmssLobj to be used later
- **Qmss_start (qmss_drv.c)**
 - Load global structure into local memory of each core
- **Qmss_insertMemoryRegion (qmss_drv.c)**
 - Base address of each region
 - Number of descriptors
 - Size of descriptors
 - Region type
 - How the region is managed (either by the LLD or the application)
 - Region number (or not specified)

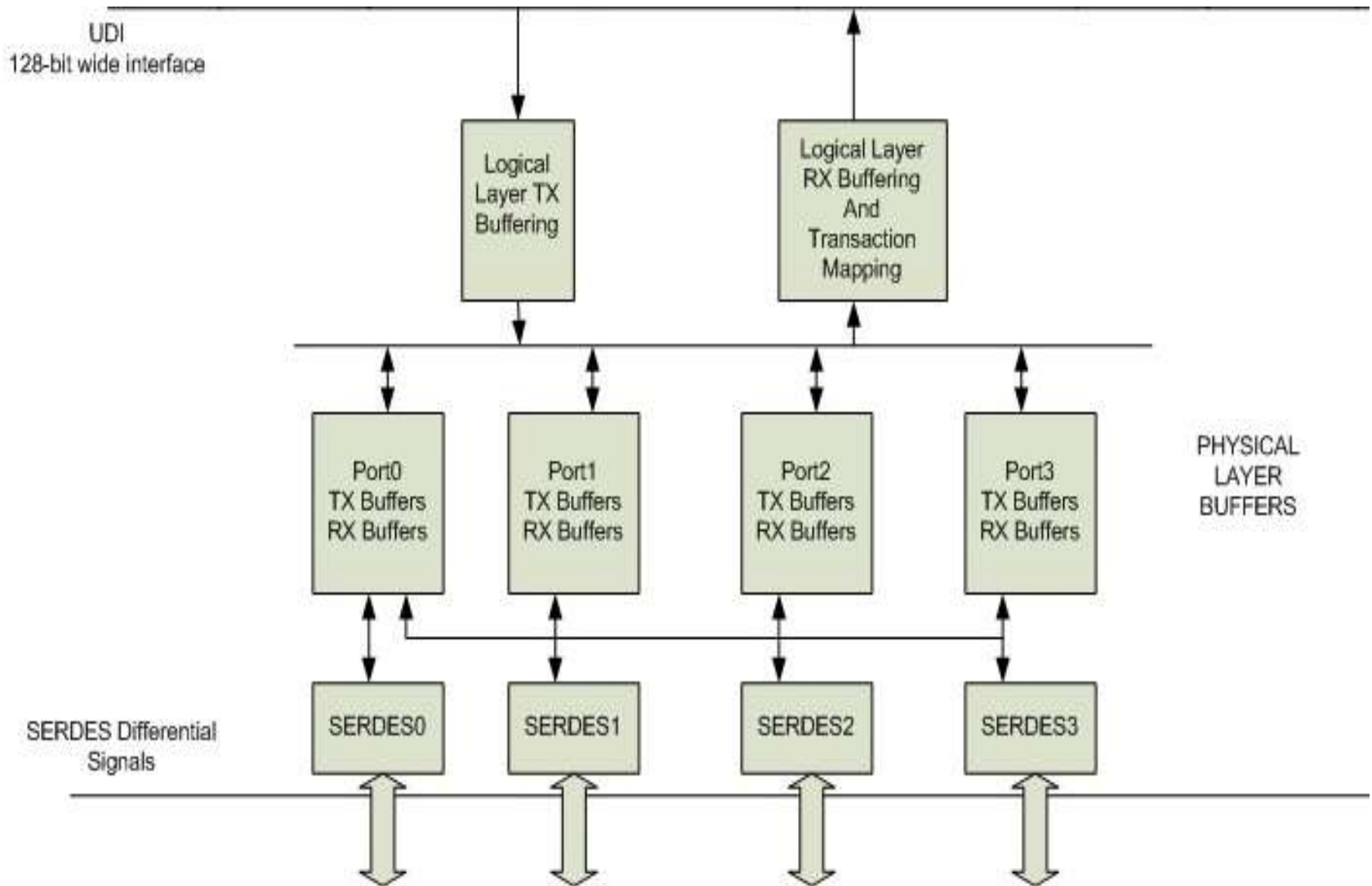
Global PKTDMA (CPPI) Initialization

- **cpapi_init (cpapi_drv.c)** loads all instances of PKTDMA from the global structure `cpapiGblCfgParas`, which is defined in the file `cpapi_device.c`
 - SRIO
 - PA
 - QMSS
 - AIF (wireless applications only)
 - FFTC (wireless applications only)
 - BCP (wireless applications only)
- SRIO PKTDMA (CPPI) configuration after SRIO configuration

SRIO Layers



SRIO Physical Layer



SRIO Initialization

- **enable_srio**
 - Power
 - PLL/Clock
- **srioDevice_init**
 - Handle for the SRIO instance
 - SERDES
 - Port
 - Routing and queues

SRIO PKTDMA (CPPI) Initialization

- Configure SRIO PKTDMA
- Set the Rx routing table to the following default locations:
 - Type 11
 - Type 9
 - Direct IO

Application-specific Configuration

“All Cores” Initialization

1. Create and initialize descriptors.
2. Allocate data buffers.
3. Associate a receive queue with each core.
4. Define receive free queue.
5. Define receive flows.
6. Define and configure transmit queues.
7. Enable transmit and receive channels.
8. Connect SRIO interrupts.

Open Sockets

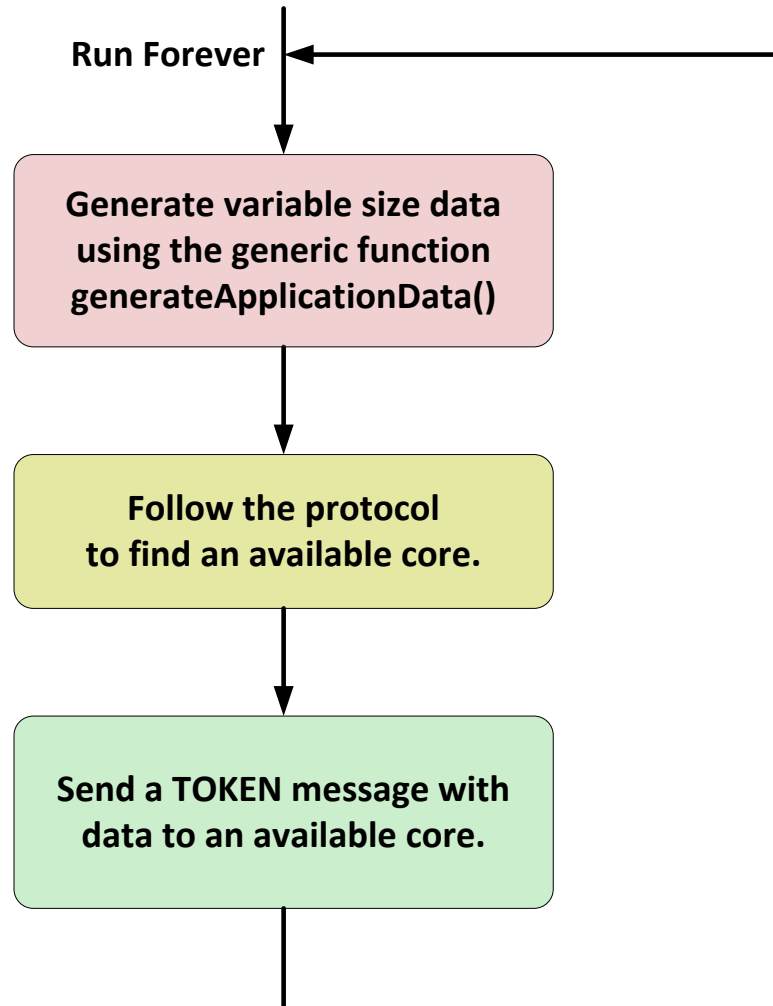
- **Srio_sockOpen()** opens a socket
- **Srio_sockBind()** binds the opened socket to routing
 - Segmentation mapping

Agenda

- Model
- Protocol
- Configuration
- **Application Algorithm**
- Build and Run

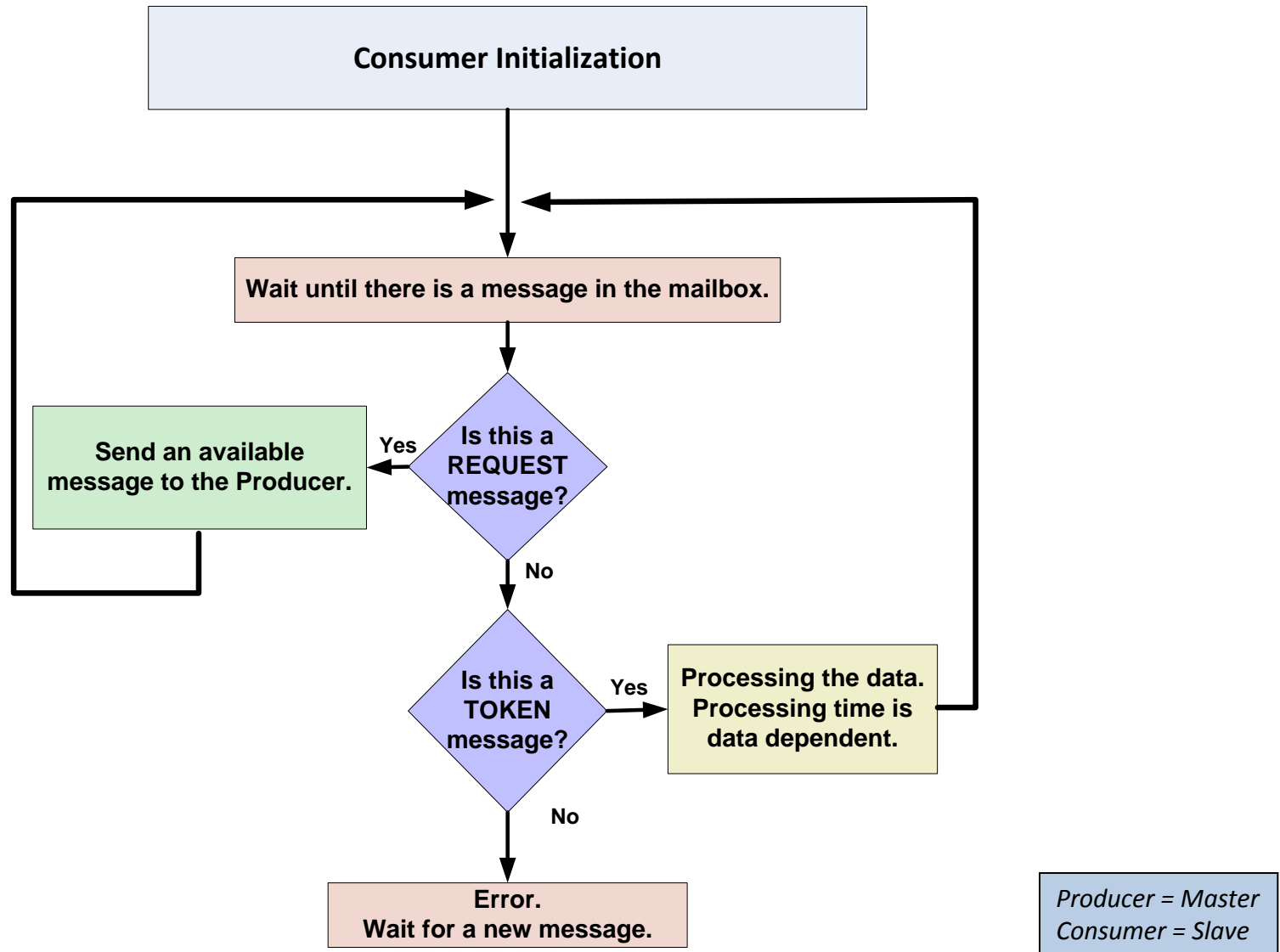
Producer (Master) Application Algorithm

Master Algorithm Flow



*Producer = Master
Consumer = Slave*

Consumer (Slave) Application Algorithm



Code Change: Producer

```
generateApplicationData(  
    fftInputBuffer[0],  
    &parameter1) ;  
size = 1 << parameter1 ;
```

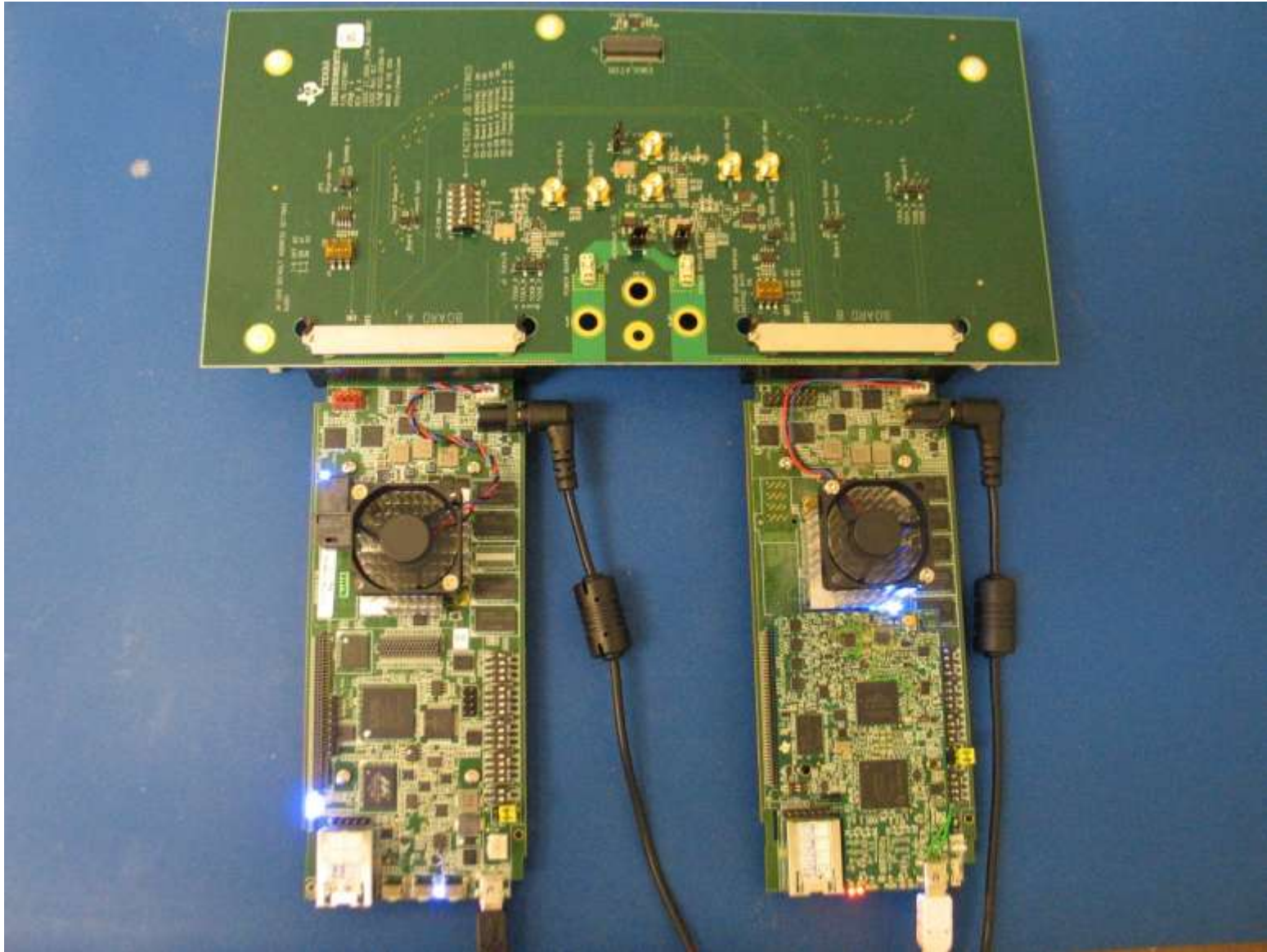
Code Change: Consumer

```
else if (messageValue == TOKEN)
{
    applicationCode (
        ptr_rxDataPayload, parameter1,
        coreNum) ;
}
```

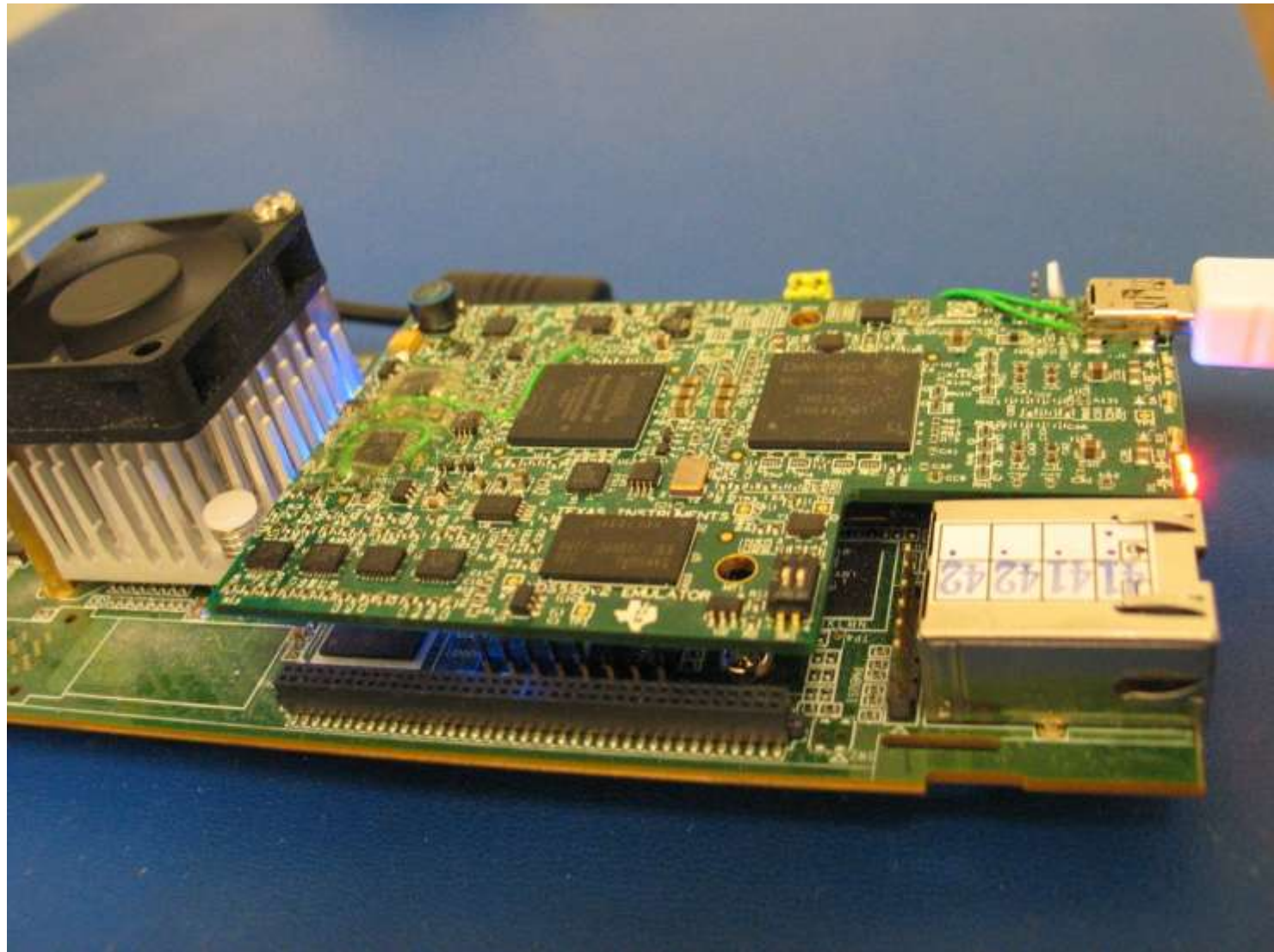

Agenda

- Model
- Protocol
- Configuration
- Application Algorithm
- Build and Run

Breakout Connector Board



C6678L w/ Mezzanine Emulator



Build and Run Process

1. Unzip the two projects (producer and consumer).
2. Update the include path (compiler) and the files search path (linker).
3. Build both projects.
4. Connect DSP 0 and load producer to all cores.
5. Connect DSP 1 and load consumer to all cores.
6. Run DSP 0 and DSP 1.

Expected Results

[C66xx_3] fft size 512 output 800058b0 real 8000bd00 imag 80009d00

[C66xx_2] fft size 128 output 800050a0 real 8000b900 imag 80009900

[C66xx_7] fft size 64 output 800078f0 real 8000cd00 imag 8000ad00

[C66xx_4] fft size 32 output 800060c0 real 8000c100 imag 8000a100

[C66xx_0] fft size 512 output 80004080 real 8000b100 imag 80009100

[C66xx_1] fft size 512 output 80004890 real 8000b500 imag 80009500

[C66xx_2] fft size 128 output 800050a0 real 8000b900 imag 80009900

[C66xx_7] fft size 512 output 800078f0 real 8000cd00 imag 8000ad00

[C66xx_4] fft size 512 output 800060c0 real 8000c100 imag 8000a100