KeyStone SoC Training
SRIO Demo: Board-to-Board

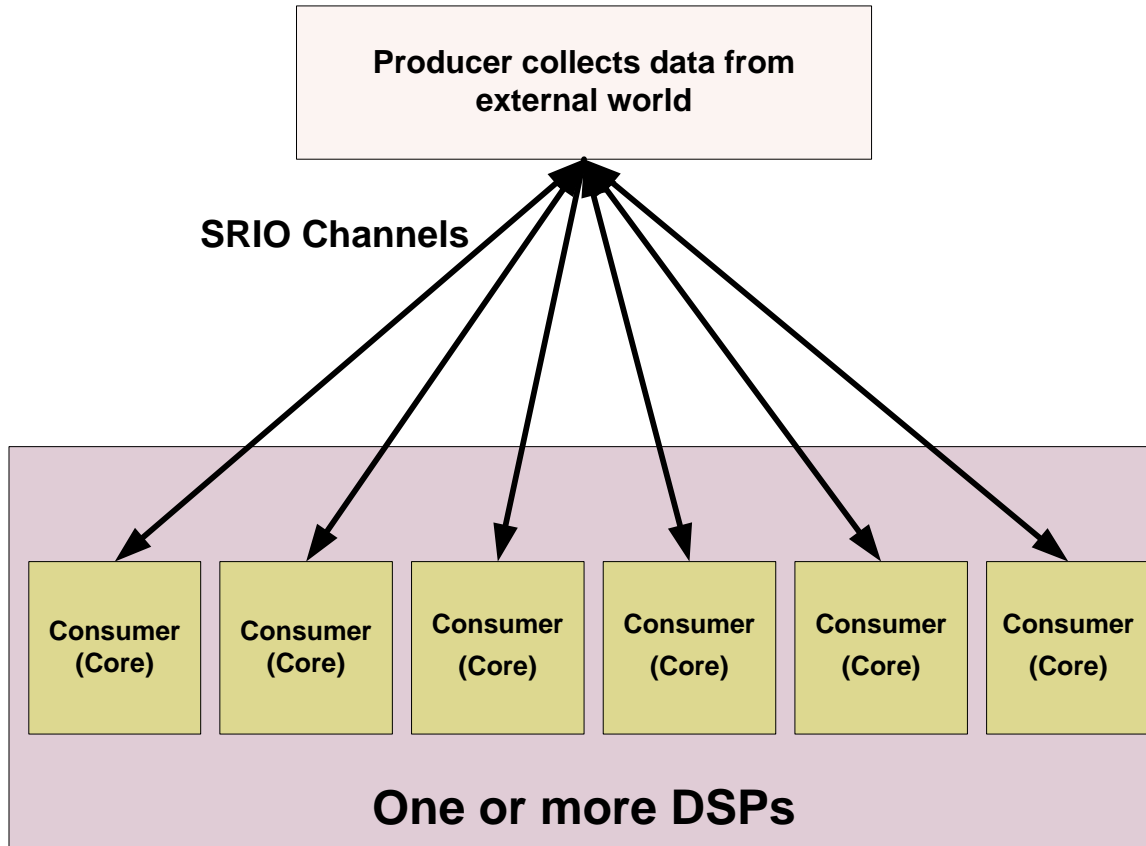MMI Application Team

December 2011

Agenda

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

The Model

Producer collects data from external world

SRIO Channels

| Consumer (Core) | Consumer (Core) | Consumer (Core) | Consumer (Core) | Consumer (Core) | Consumer (Core) |

One or more DSPs

Producer = Master
Consumer = Slave
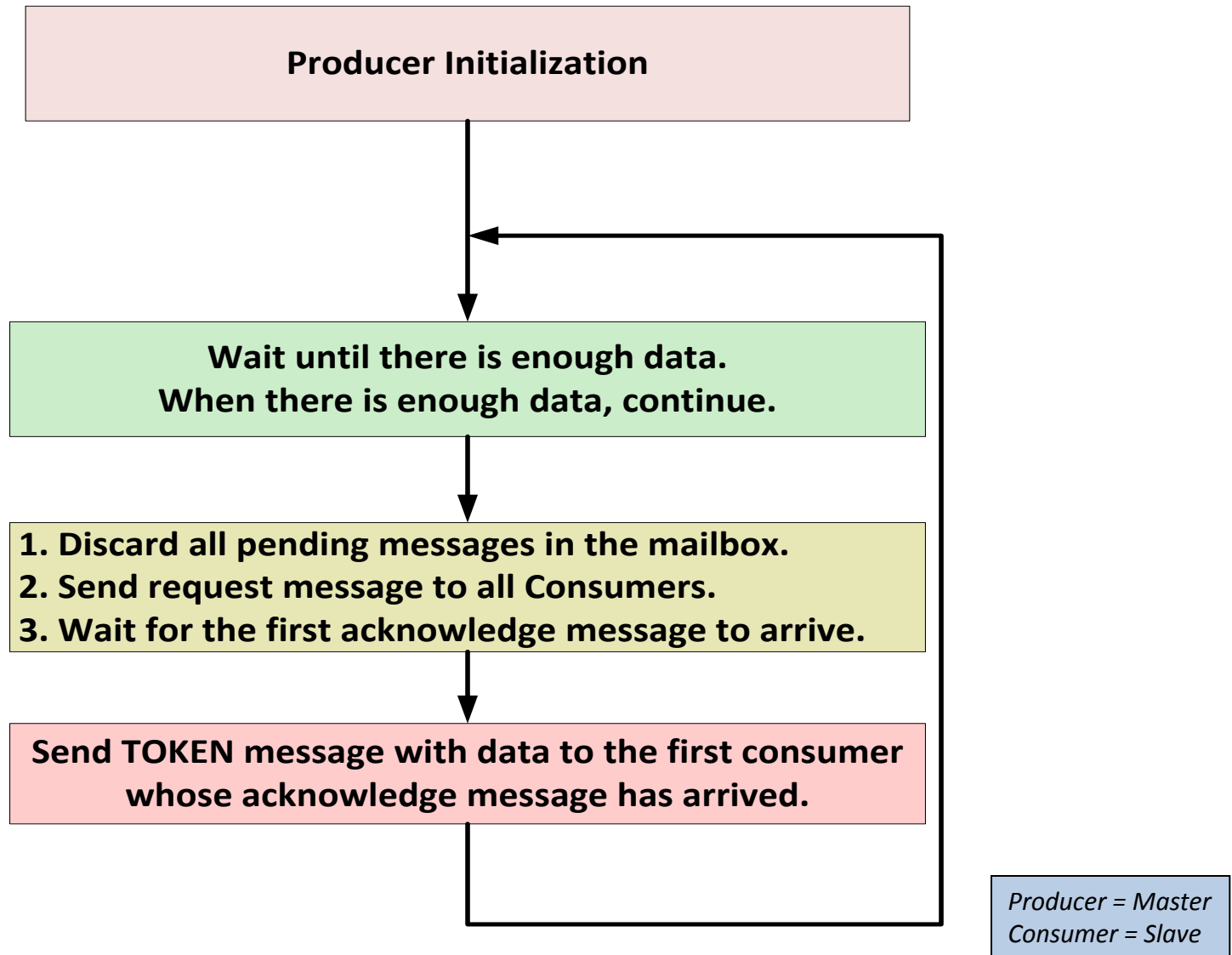
Requirements:

- Efficiency – Not fairness

- Minimize master logic

- Master is not aware of structure of internal cores

Agenda

- Model
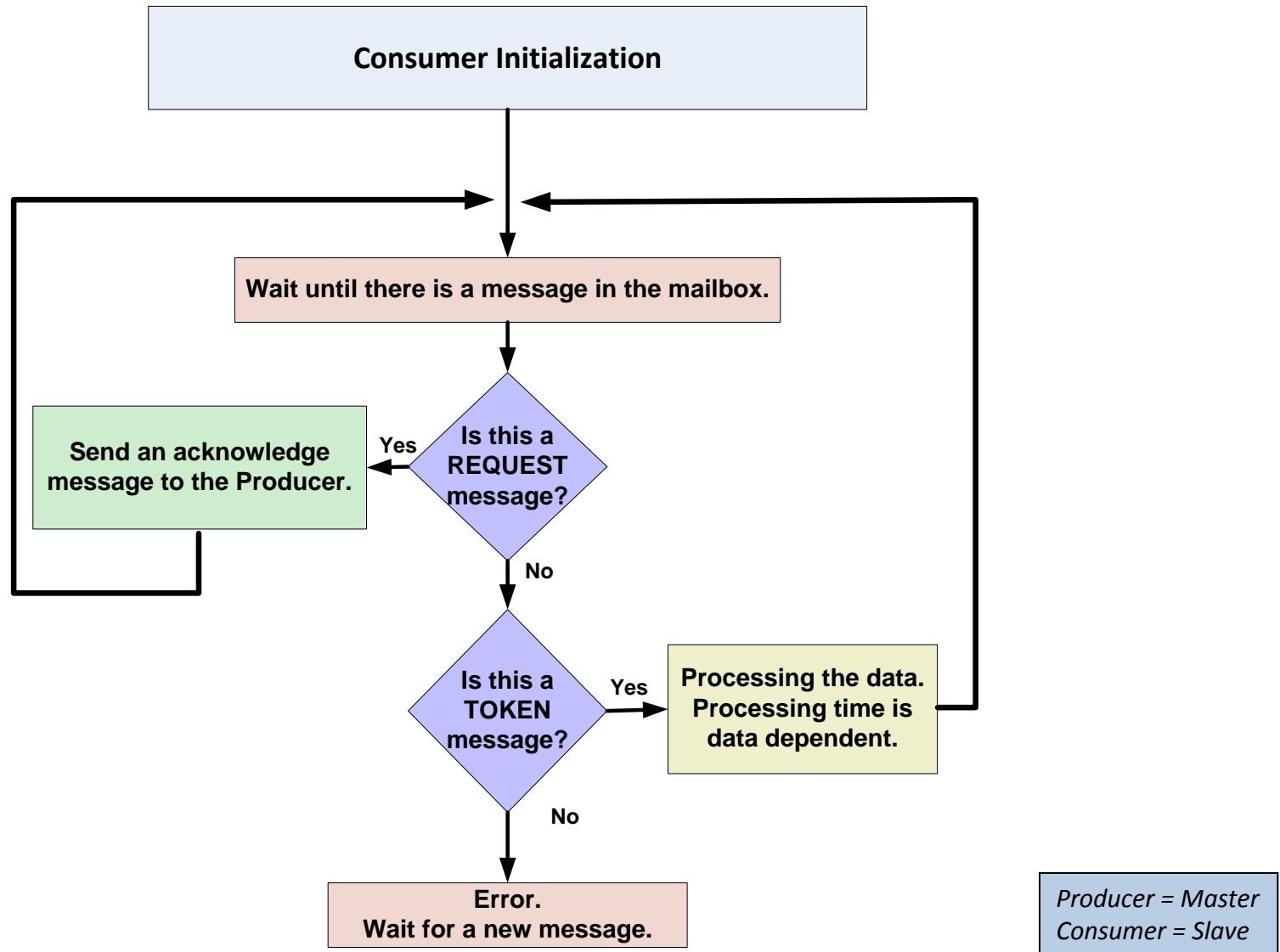
- Protocol

- Configuration

- Application Algorithm

- Build and Run
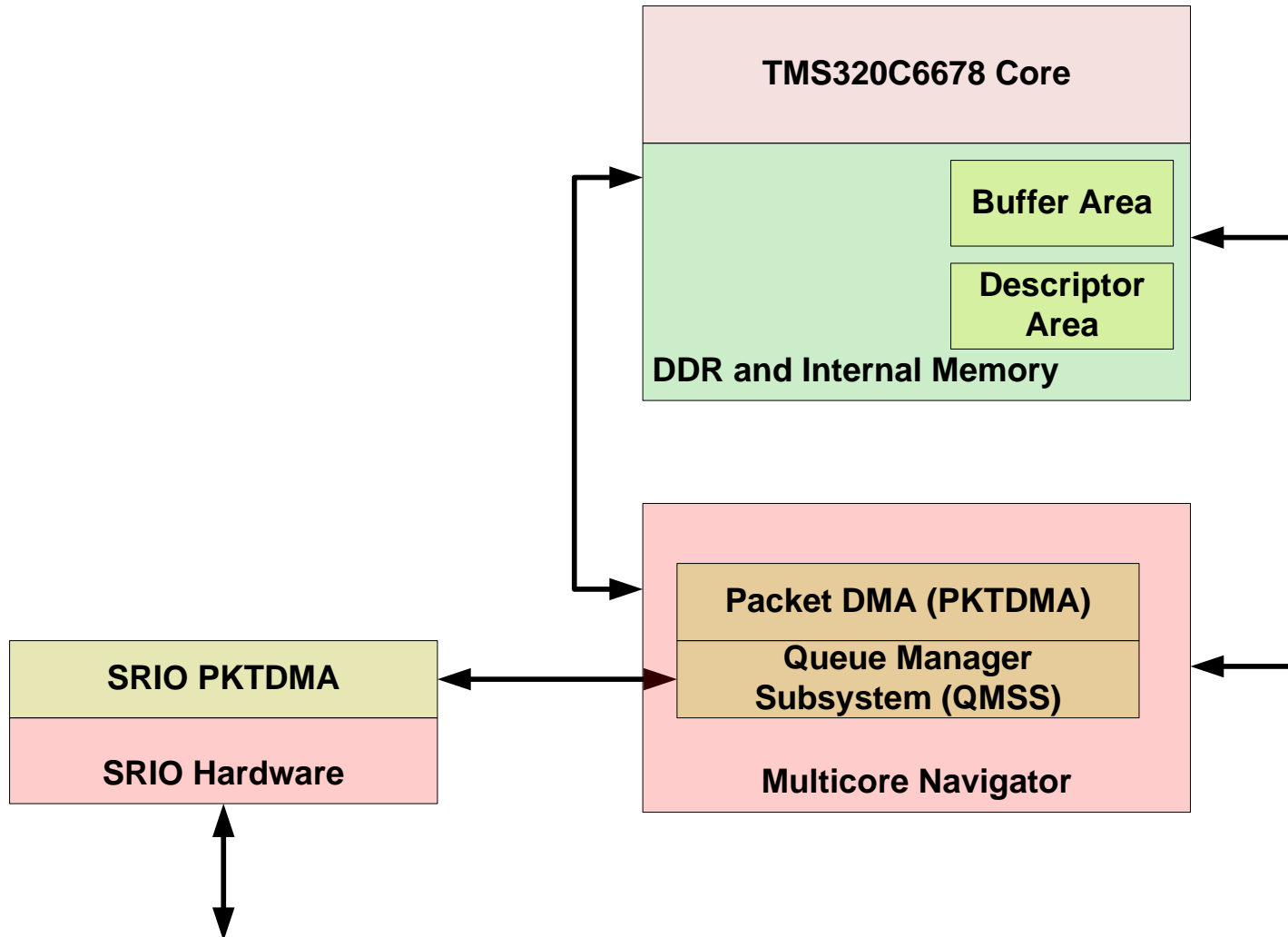
Producer (Master) Protocol

Producer Initialization

Wait until there is enough data.
When there is enough data, continue.

1. Discard all pending messages in the mailbox.
2. Send request message to all Consumers.
3. Wait for the first acknowledge message to arrive.

Send TOKEN message with data to the first consumer whose acknowledge message has arrived.

Producer = Master
Consumer = Slave

Consumer (Slave) Protocol

Consumer Initialization

Wait until there is a message in the mailbox.

Is this a REQUEST message?

Yes → Send an acknowledge message to the Producer.

No

Is this a TOKEN message?

Yes → Processing the data. Processing time is data dependent.

No

Error. Wait for a new message.

Producer = Master
Consumer = Slave

Agenda

- Model

- Protocol

- Configuration

- Application Algorithm

- Build and Run

Hardware Components

TMS320C6678 Core

Buffer Area

Descriptor Area

DDR and Internal Memory

Packet DMA (PKTDMA)

Queue Manager Subsystem (QMSS)

Multicore Navigator

SRIO PKTDMA

SRIO Hardware

Packet DMA Topology
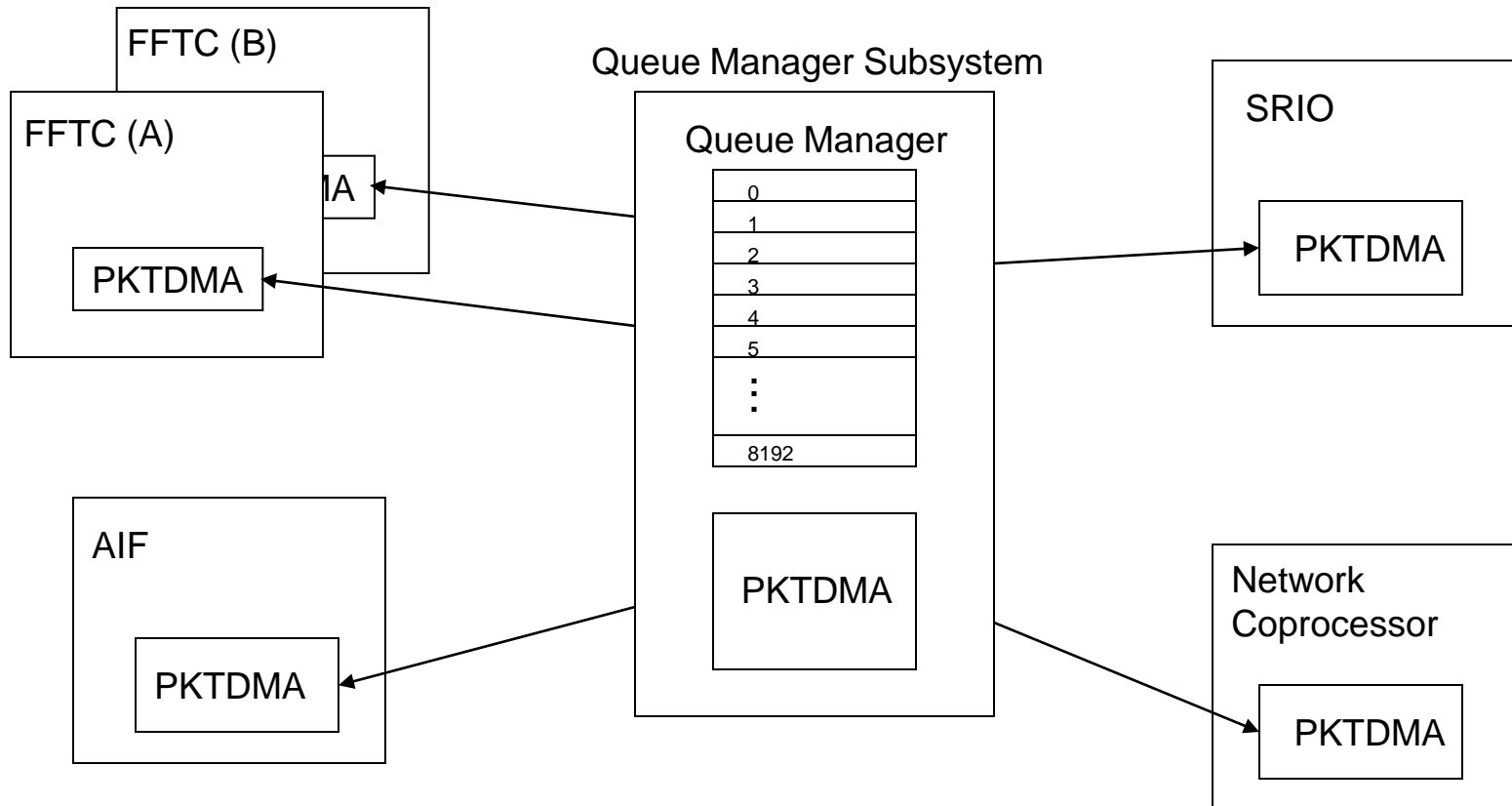
Multiple Packet DMA instances in KeyStone devices:

- PA and SRIO instances for all KeyStone devices.

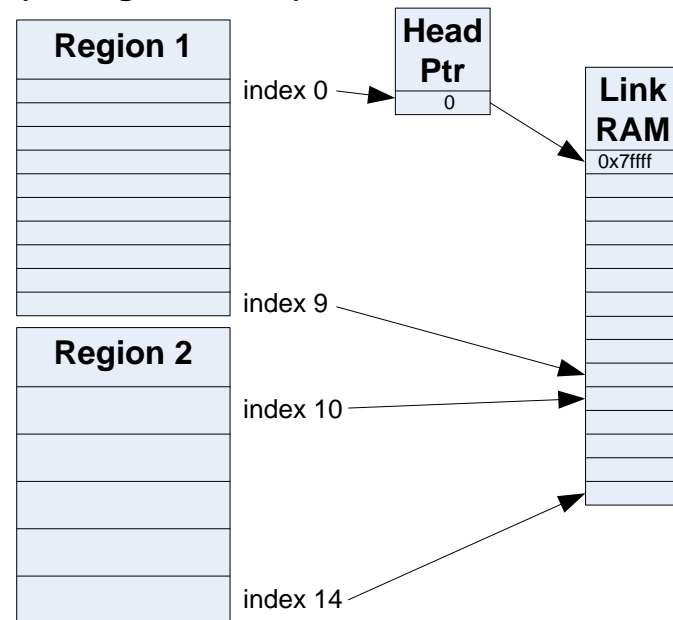- AIF2 and FFTC (A and B) instances are only in KeyStone devices for wireless applications.

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)
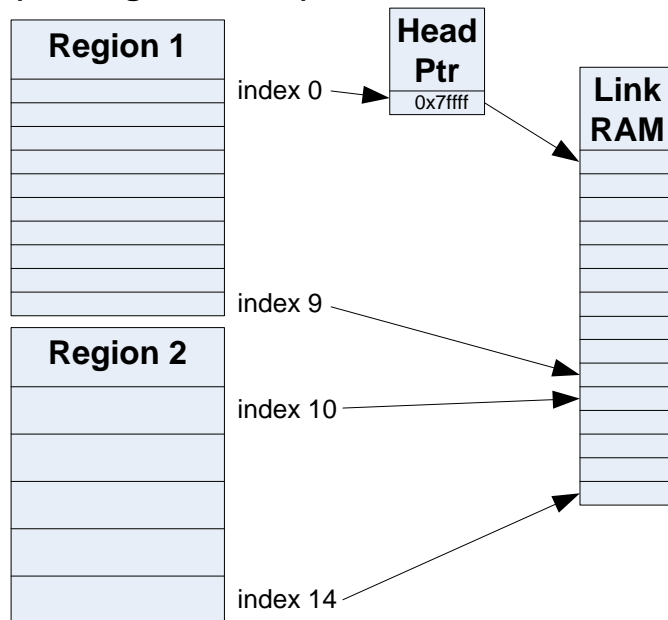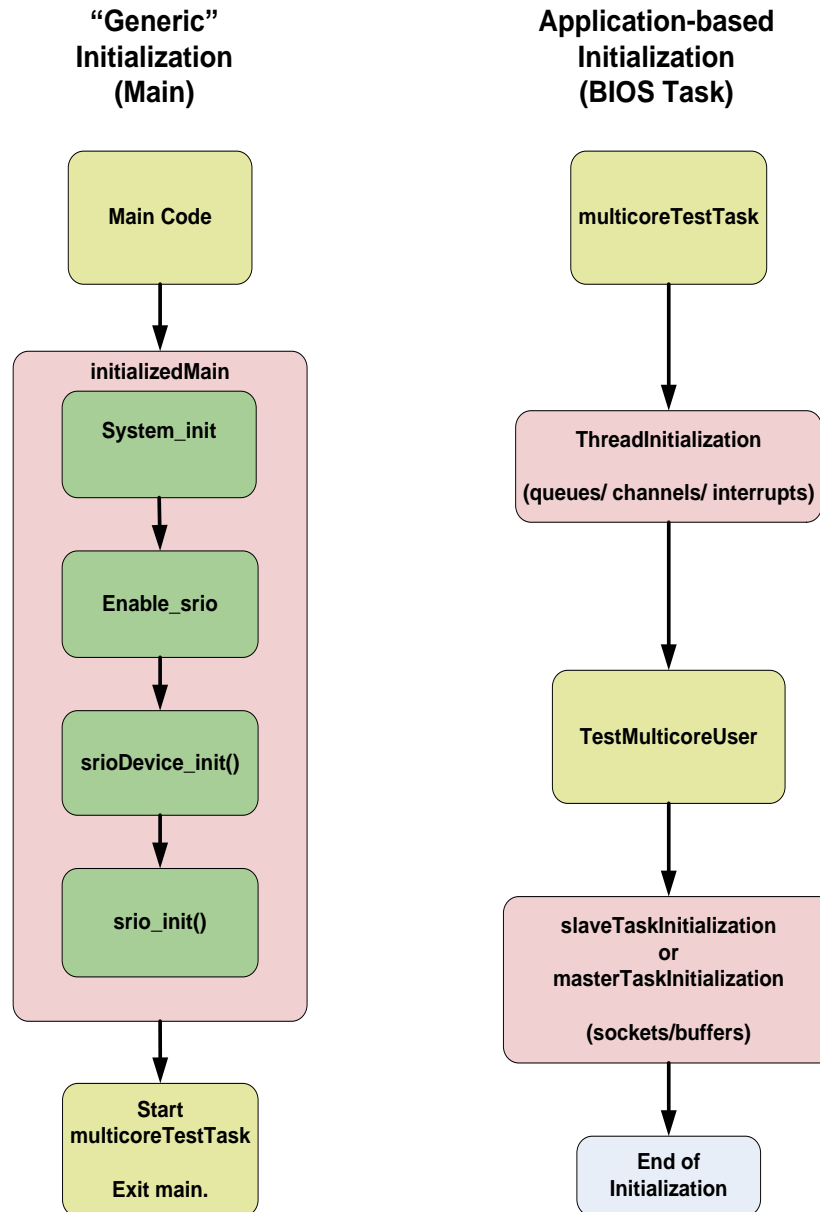
Push index 0 to an empty queue (ending condition)

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

Configuration/Initialization Flow

"Generic" Initialization (Main)

Main Code

↓

initializedMain
- **System_init**
  ↓
- **Enable_srio**
  ↓
- **srioDevice_init()**
  ↓
- **srio_init()**

↓

Start multicoreTestTask

Exit main.

Application-based Initialization (BIOS Task)

multicoreTestTask

↓

ThreadInitialization

(queues/ channels/ interrupts)

↓

TestMulticoreUser

↓

slaveTaskInitialization or masterTaskInitialization

(sockets/buffers)

↓

End of Initialization

Configuration Steps:

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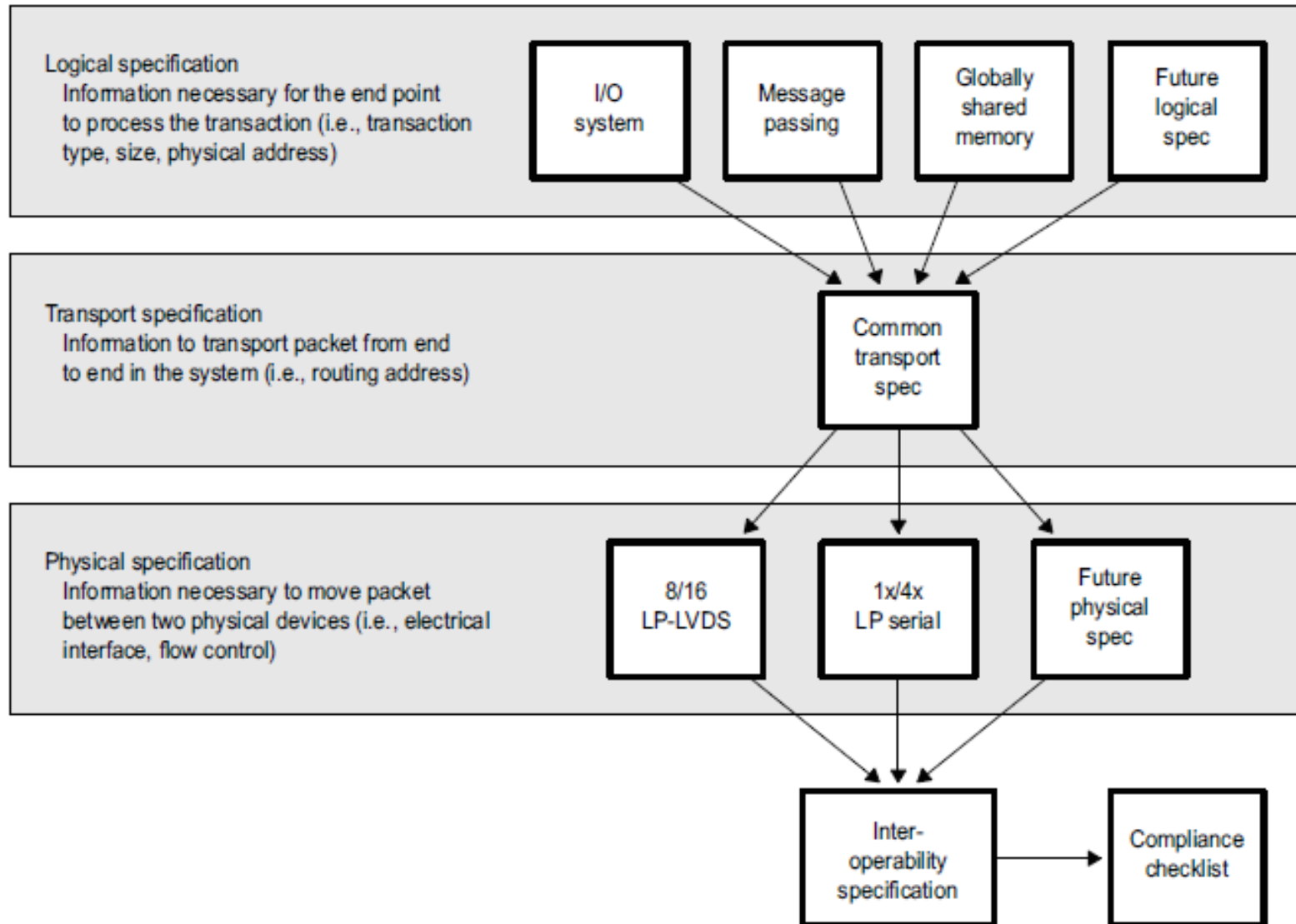3. QMSS PKTDMA
4. SRIO
5. SRIO PKTDMA
6. Sockets

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)
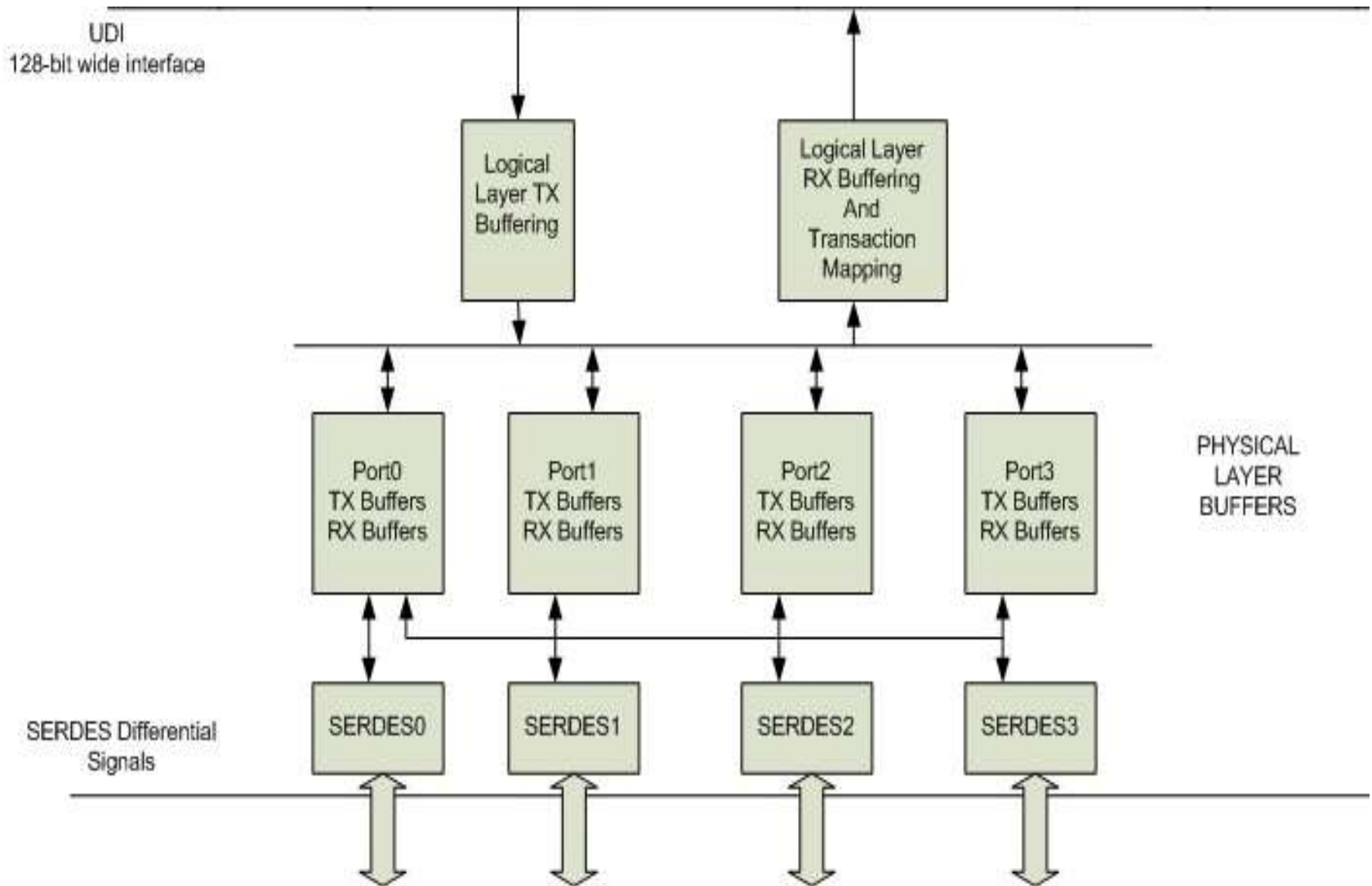 - Region number (or not specified)

Global PKTDMA (CPPI) Initialization

- **cppi_init (cppi_drv.c)** loads all instances of PKTDMA from the global structure cppiGblCfgParas, which is defined in the file cppi_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

Logical specification
  Information necessary for the end point
  to process the transaction (i.e., transaction
  type, size, physical address)

| I/O system | Message passing | Globally shared memory | Future logical spec |

Transport specification
  Information to transport packet from end
  to end in the system (i.e., routing address)

Common transport spec

Physical specification
  Information necessary to move packet
  between two physical devices (i.e., electrical
  interface, flow control)

| 8/16 LP-LVDS | 1x/4x LP serial | Future physical spec |

Inter-operability specification → Compliance checklist

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.

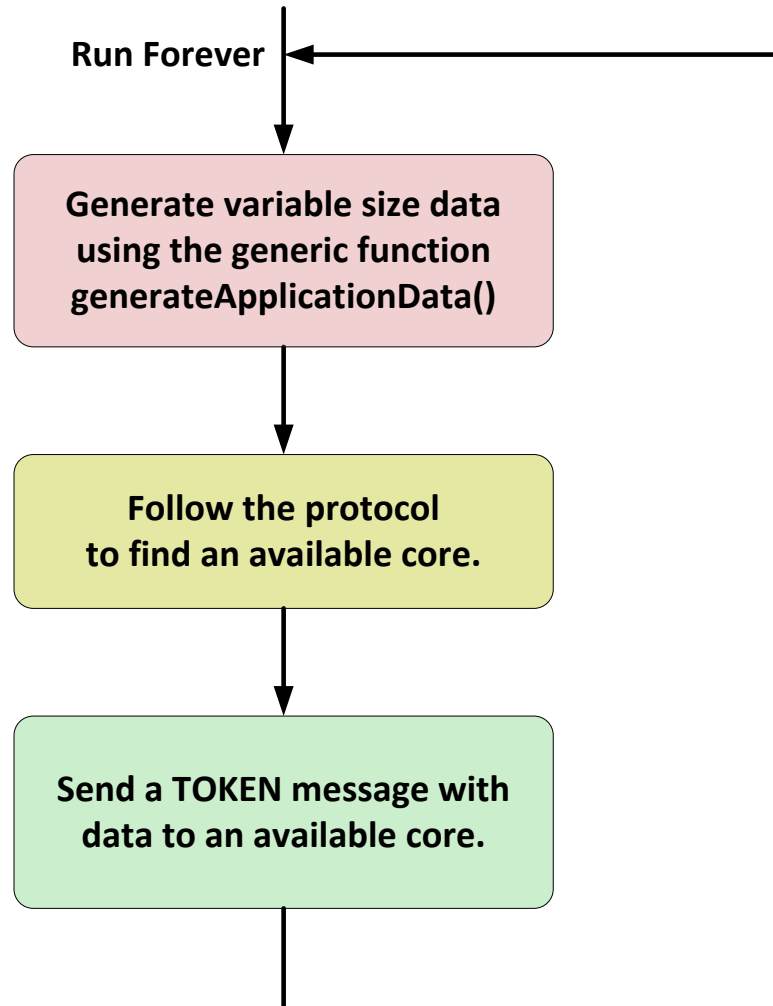8. Connect SRIO interrupts.

Open Sockets

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

Agenda

- Model

- Protocol

- Configuration

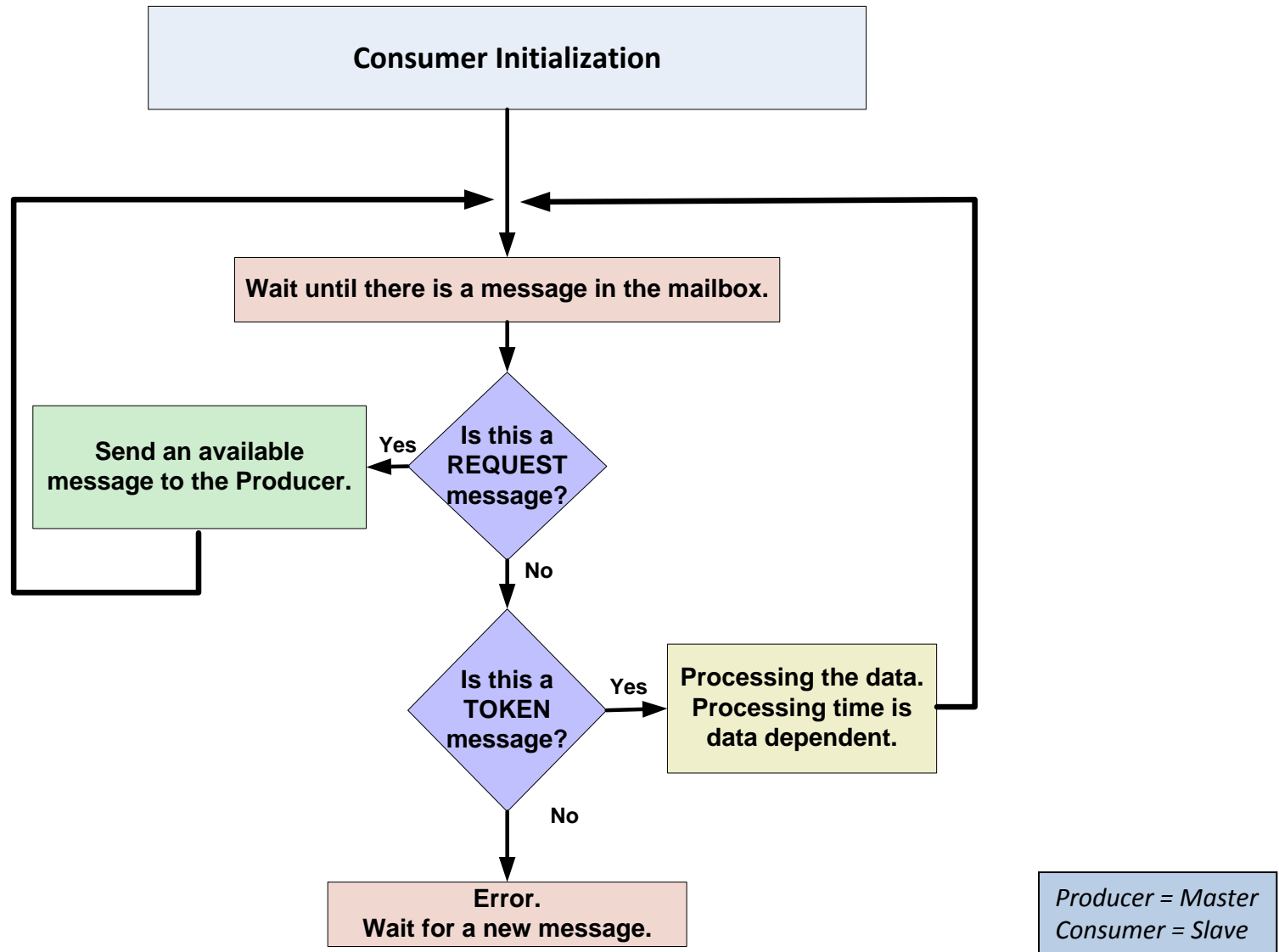- Application Algorithm

- Build and Run

Producer (Master) Application Algorithm

Master Algorithm Flow

Run Forever

Generate variable size data
using the generic function
generateApplicationData()

Follow the protocol
to find an available core.

Send a TOKEN message with
data to an available core.

Producer = Master
Consumer = Slave

Consumer (Slave) Application Algorithm

Consumer Initialization

Wait until there is a message in the mailbox.

Is this a REQUEST message?

Yes → Send an available message to the Producer.

No

Is this a TOKEN message?

Yes → Processing the data. Processing time is data dependent.

No

Error. Wait for a new message.

Producer = Master
Consumer = Slave

Code Change: Producer

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

Code Change: Consumer

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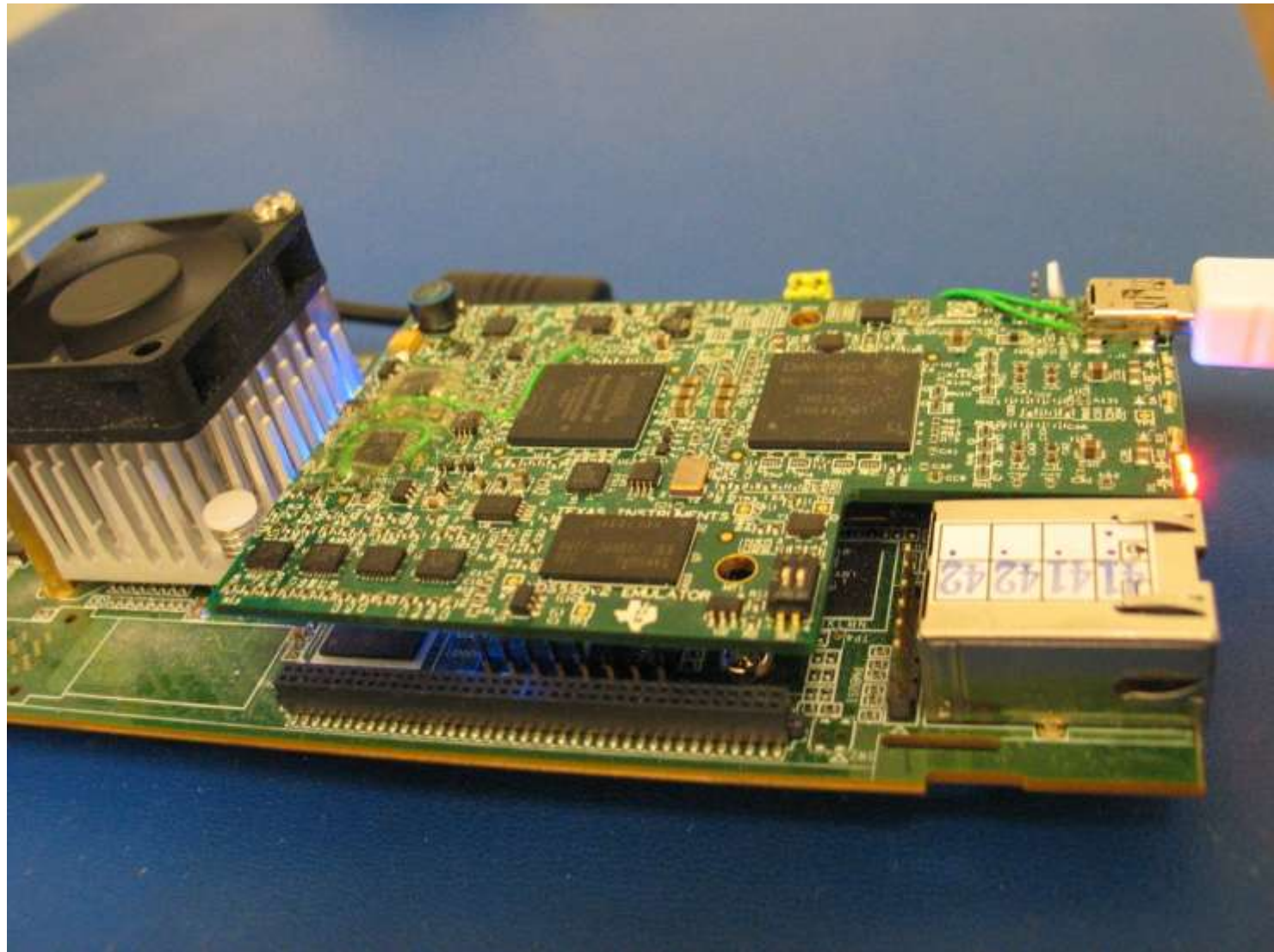
Agenda

- 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