KeyStone Training

Serial RapidIO (SRIO) Subsystem

Agenda

- SRIO Overview
- DirectIO Operation
- Message Passing Operation
- RapidIO Interrupts
- Understanding SRIO Implementation
- Configuring the Demo Setup
- Summary

SRIO Overview

- SRIO Overview
- DirectIO Operation
- Message Passing Operation
- RapidIO Interrupts
- Understanding SRIO Implementation
- Configuring the Demo Setup
- Summary

Introduction To RapidIO

- RapidIO architecture is divided into three layers:
  1. Physical Layer
     - SERDES
     - RapidIO Physical layer IP
  2. Transport Layer
     - Transports packet from physical layer to logical layer protocol units
  3. Logical Layer
     - Protocol Units (e.g. LSUs, TXU, etc.)

Introduction to RapidIO

- RapidIO 2.1.1 Compliant
- For ports with options, refer to the table below:

Mode 0	1x	1x	1x	1x
Mode 1	1x	1x	2x	
Mode 2	2x		1x	1x
Mode 3	2x		2x	
Mode 4	4x			

- Data rates up to 5 Gbaud
- Different ports at different baud rates (Only integer multiple different rates are allowed)

Physical Layer

Introduction to RapidIO

- Direct IO Mode of Operation
  - Read/write operations directed to specific memory address.
  - Transmit device has knowledge of memory map of receiving device.
  - LSU (Load/Store unit) for transmission
  - MSU (Memory Access Unit) for reception
- Message passing mode of operation
  - Two types of packets that are supported (Type9 and Type 11).
  - Uses message & letter designators; No knowledge of memory map required.
  - TXU for message transmission.
  - RXU for message reception.
- Strict priority scheduler
  - Round robin interleaved on a packet basis at a given priority
  - Outbound credit-aware functional blocks
- Auto-promotion of response priorities by RXU and MAU can now be disabled
- Ability to set the CRF (Critical Request Flow) bit on outgoing requests and responses

Logical Layer

DirectIO Operation

- SRIO Overview
- DirectIO Operation
- Message Passing Operation
- RapidIO Interrupts
- Understanding SRIO Implementation
- Configuring the Demo Setup
- Summary

DirectIO Operations

- Eight (8) LSUs
- Maximum transaction size (byte_count field) of 1MB
  - Up to 4K packets of 256 bytes per LSU programming
- Shadow Registers concept
- 128 outstanding non-posted packets in total, 16 per LSU (not configurable)
- Auto-generation of doorbell at the end of transfer completion.
  - Send doorbell after sending last packet.
    OR
  - Send doorbell after receiving last response.
  - No doorbell is sent if there is an error.
- Restart and flush LSU transactions.

DirectIO Operation - LSU Registers

	31:0
LSU_Reg0	**RapidIO Destination Address MSB**
LSU_Reg1	**RapidIO Destination Address LSB/Config_Offset**
LSU_Reg2	**DSP Source Address**

LSU_Reg3	31	30:20	19:0
	Drbll_val	RSVD	**Byte_Count**

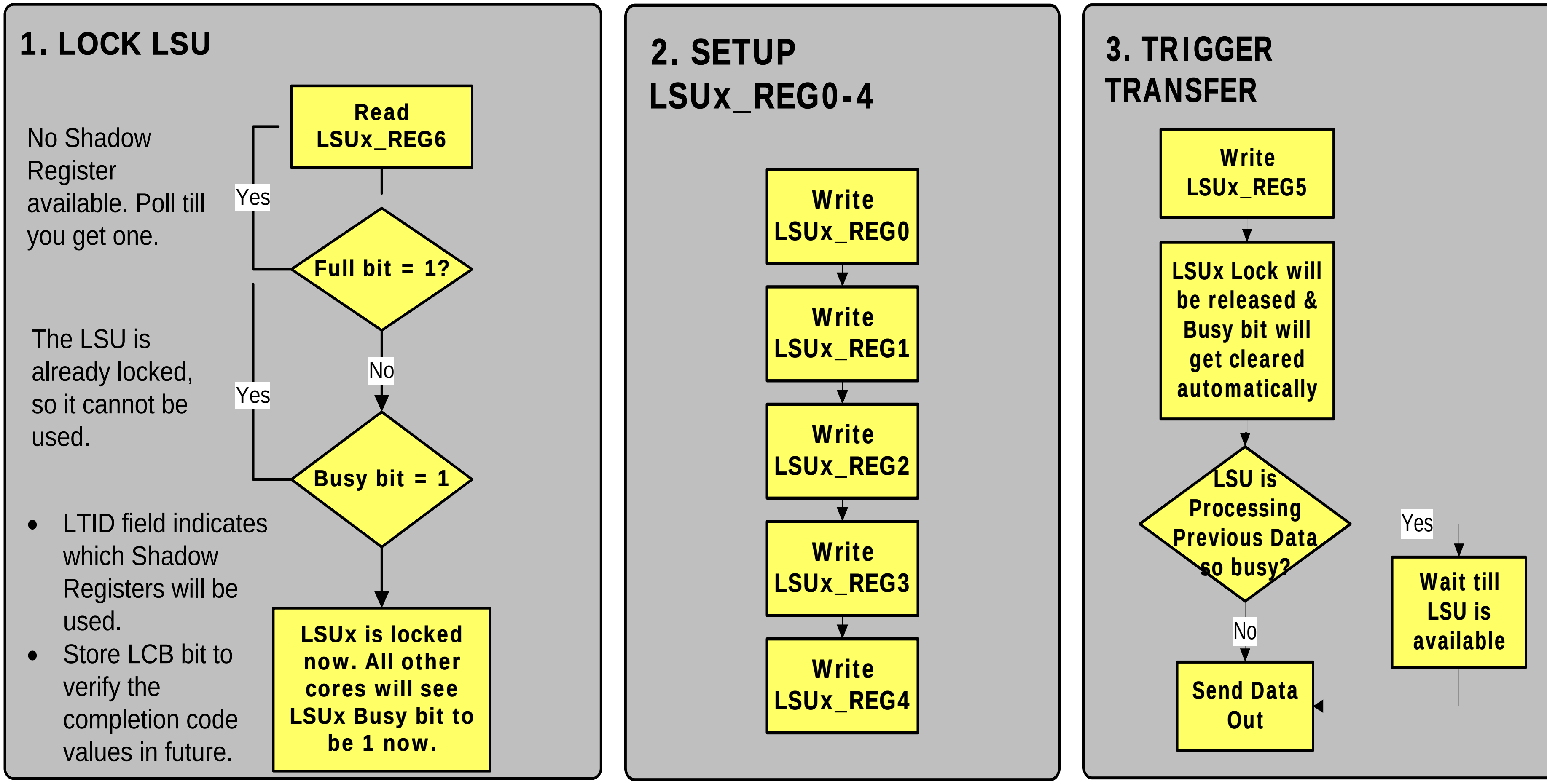
LSU_Reg4	31:16	15:12	11:10	9:8	7:4	3:2	1	0
	DestID	SrcID_MAP	ID_Size	OutPortID	Priority	Xambs	Sup_gint	Int_Req

LSU_Reg5	31:16	15:8	7:4	3:0
	Drbll_Info	Hop Count	**FType**	**TType**

LSU_Reg6 (RO)	31	30	29:5	4	3:0
	Busy	Full	RSVD	LCB	LTID

LSU_Reg6 (WO)	31:28	27	26:6	5:2	1	0
	PrivID	CBUSY	RSVD	SrcID_MAP	Restart	Flush

Tx Operation: Manual Trigger Mode



Tx Operation: EDMA Trigger Mode

- In this mode, the EDMA programs the shadow registers.
  - The LSU_SETUP_REG1 is set for a specific LSU to be used with EDMA.
  - The EDMA programs LSU registers Reg0 to Reg5.
  - LSU sends the packet out and the completion generates an interrupt, which triggers the EDMA once again.
- The pre-requisite is that the LSU used by EDMA must not be used by any other master:
  - This eliminates the possibility that the LSU becomes busy by another master. So reading the busy bit is not required.
  - EDMA will be able to use only one shadow register. So full-bit checking is also not required.
  - So LSU Reg6 read is not required for EDMA mode of operation.

Message Passing Operation

- SRIO Overview
- DirectIO Operation
- Message Passing Operation
- RapidIO Interrupts
- Understanding SRIO Implementation
- Configuring the Demo Setup
- Summary

Message Passing Operations

16 Transmit & 16 Receive Channels

Queue Range	Number of Queues	Purpose
0 to 511	512	Low priority accumulation. These 512 queues are divided into 16 groups, each group with 32 continuous queues. Each group is monitored with one interrupt.
512 to 639	128	AIF Tx queues. Each queue has a dedicated queue pending signal which drives a CDMA Tx channel.
640 to 671	32	PA Tx queues. Each queue has a dedicated queue pending signal which drives a CDMA Tx channel.
672 to 687	16	SRIO Tx queues. Each queue has a dedicated queue pending signal which drives a CDMA Tx channel.
688 to 691	4	FFTC Tx queues. Each queue has a dedicated queue pending signal which drives a CDMA Tx channel.
692 to 703	12	General purpose.
704 to 735	32	High priority accumulation. Each high priority queue can be monitored based on watermark, and each queue has an interrupt signals.
736 to 799	64	Queues with starvation counters readable by the host. Starvation counters increment each time a pop is performed on an empty queue, and reset when the queue is not empty (or when the starvation count is read).
800 to 831	32	QMSS Tx queues. Used for Infrastructure (core to core) DMA copies and notification.
832 to 863	32	Generic queues reserved for traffic shaping, if it is configured in firmware to support this feature.
864 to 8191	7328	General purpose. It is not safe to use queue 8191 however, because some CDMA override functions use 0xFFF in the low 12 bits to specify non-override conditions.

16 dedicated Tx queues for 16 Tx channels

Queues for 16 Rx channels are assigned from this range.

-

	QMSS	SRIO	PA	FFTC	AIF
Global Control	0x02a6c000	0x02901000	0x02004000	0x021f0200	0x01f17200
Tx Channel Config	0x02a6c400	0x02901400	0x02004400	0x021f0300	0x01f14000
Rx Channel Config	0x02a6c800	0x02901800	0x02004800	0x021f0500	0x01f15000
Tx Scheduler Config	0x02a6cc00	0x02901c00	0x02004c00	0x021f0400	0x01f17000
Rx Flow Config	0x02a6d000	0x02901e00	0x02004e00	0x021f0600	0x01f16000

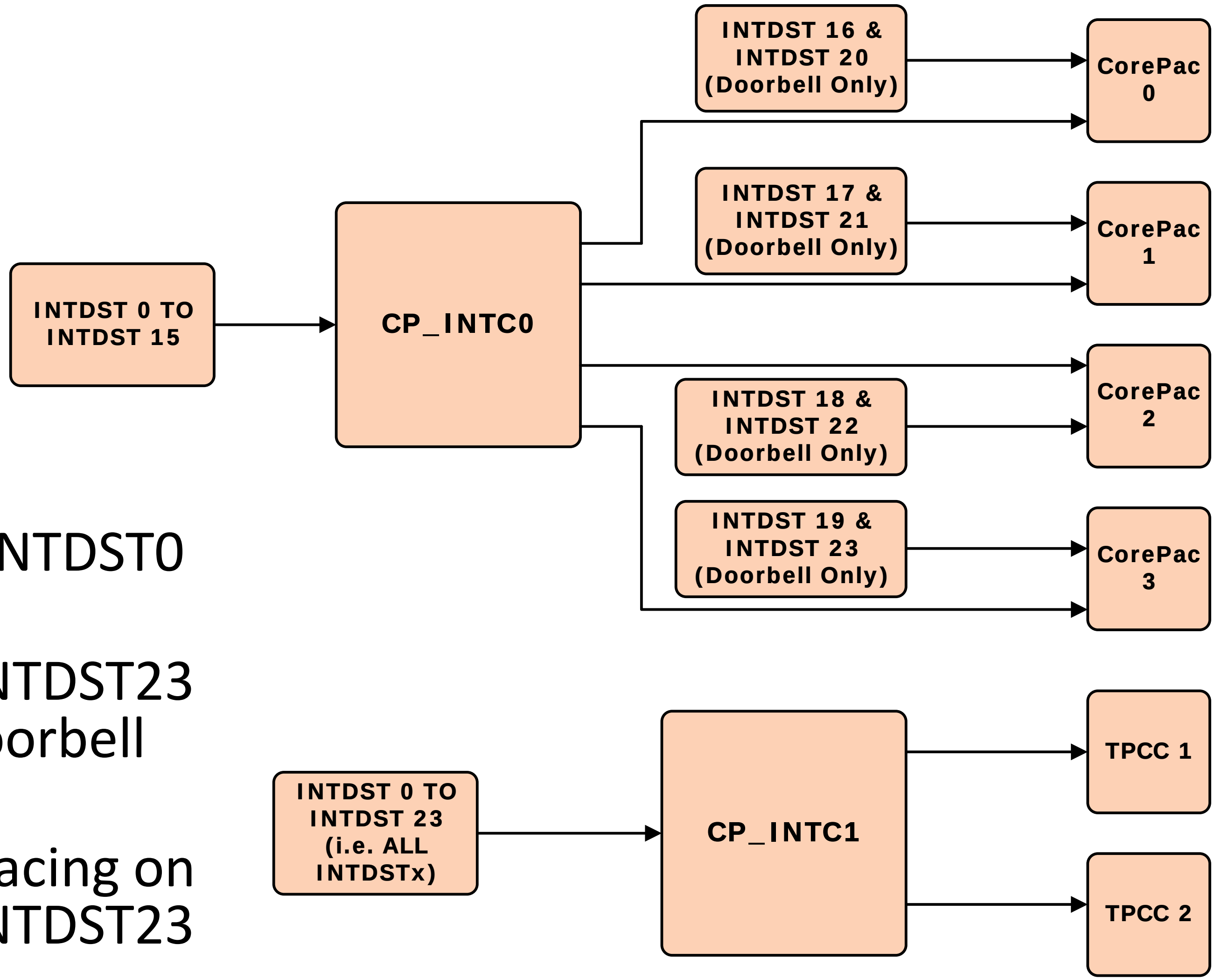
Message Passing Features

- Maximum 4 KB message size
- Maximum of 16 segments per message

RapidIO Interrupts

- SRIO Overview
- DirectIO Operation
- Message Passing Operation
- RapidIO Interrupts
- Understanding SRIO Implementation
- Configuring the Demo Setup
- Summary

Interrupt Destinations

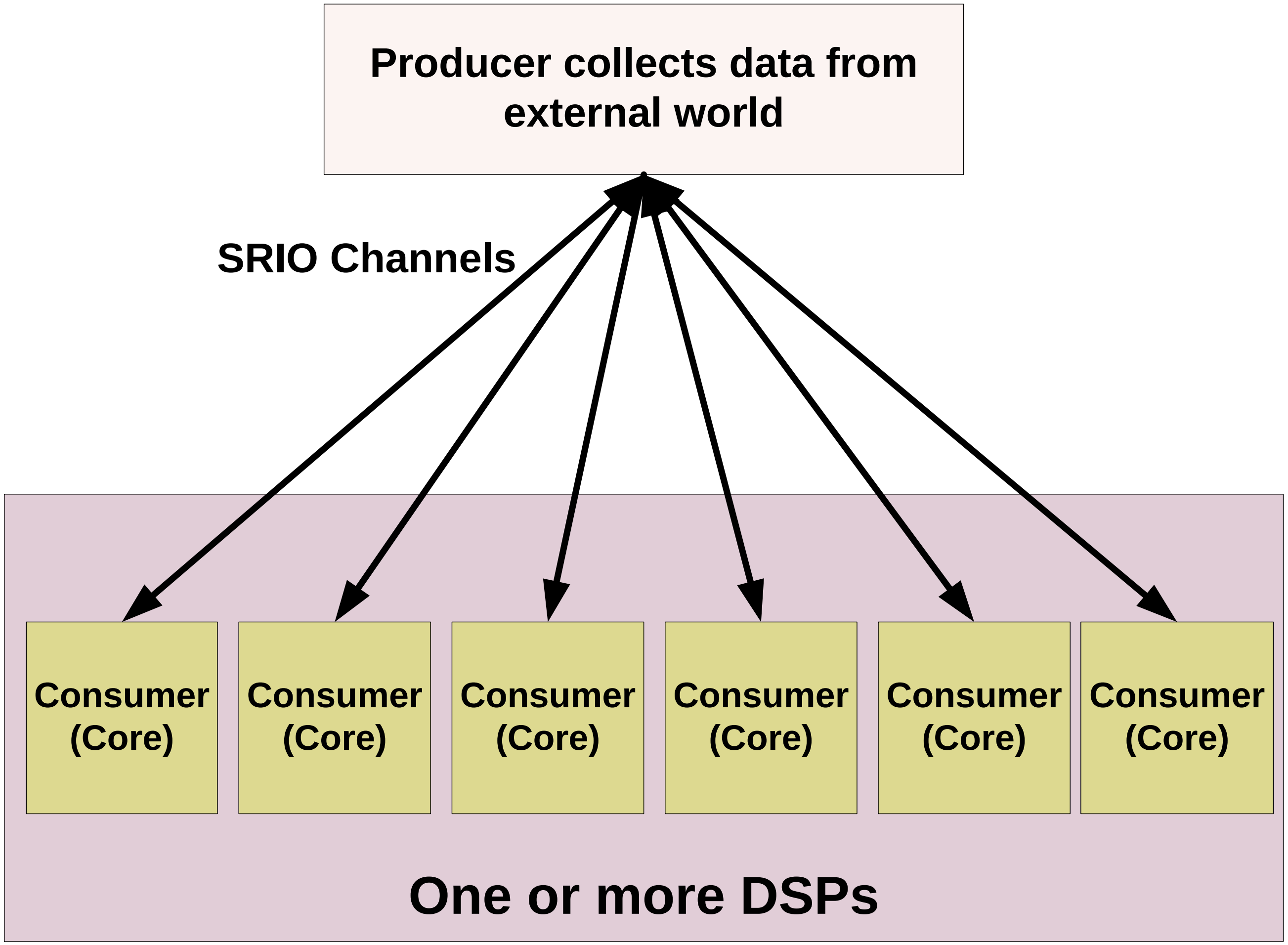


- 24 Interrupt Destinations (INTDST0 to INTDST23)
- INTDST16 to INTDST23 are only for Doorbell interrupts
- No Interrupt pacing on INTDST16 to INTDST23

Understanding SRIO Implementation

- SRIO Overview
- DirectIO Operation
- Message Passing Operation
- RapidIO Interrupts
- Understanding SRIO Implementation
- Configuring the Demo Setup
- Summary

SRIO Implementation Model

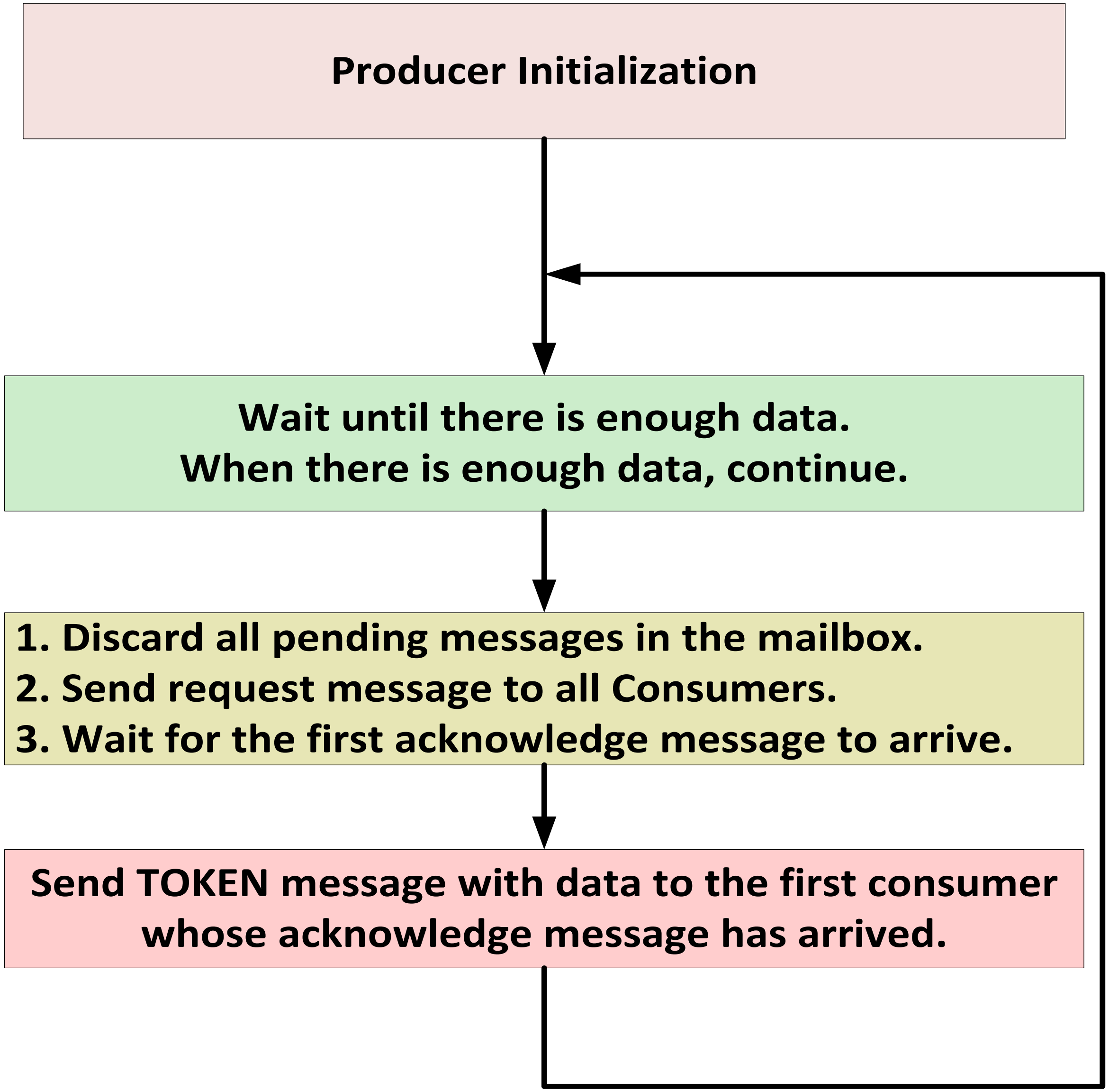


Requirements:

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

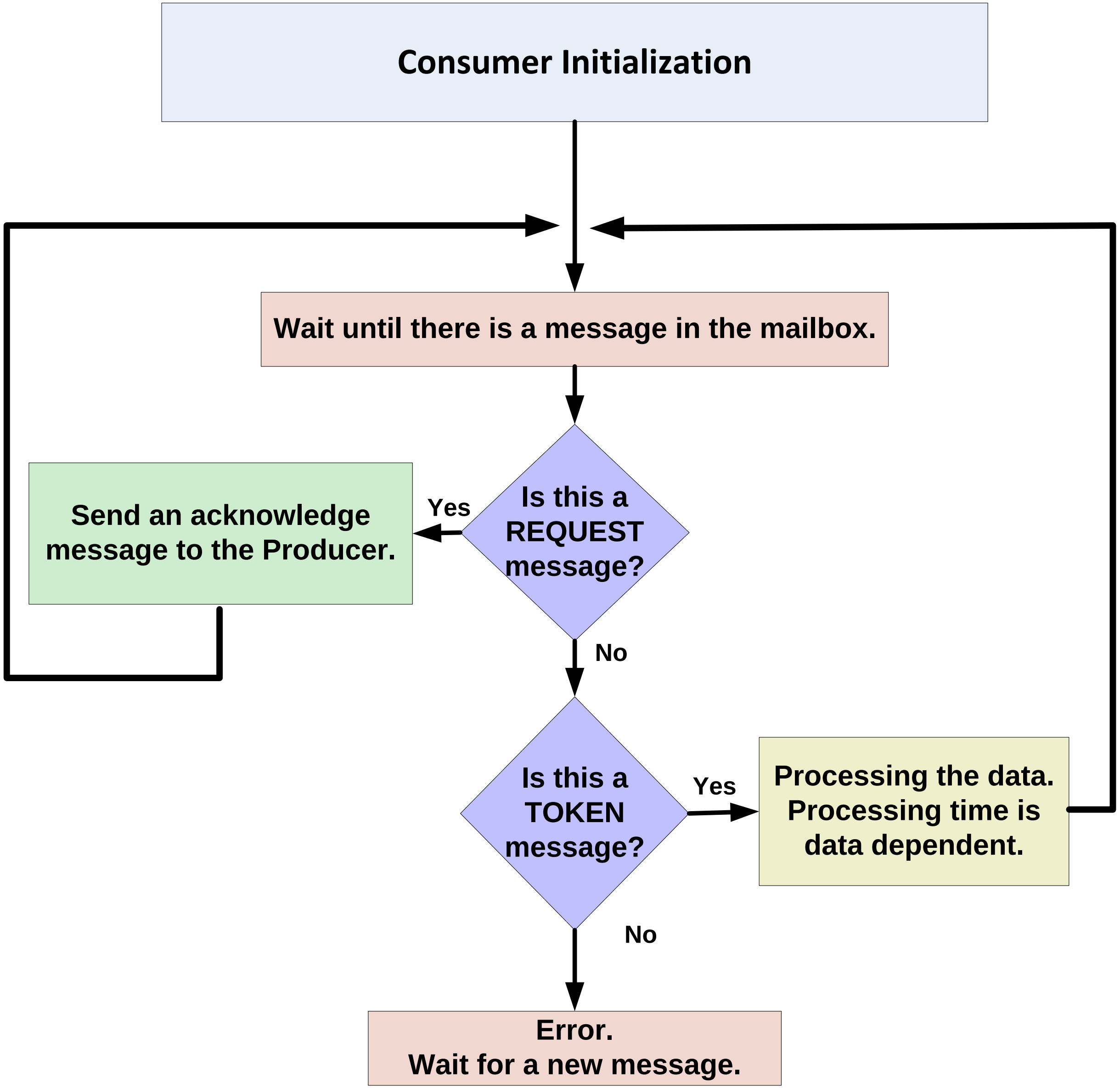
Producer = Master
Consumer = Slave

Producer (Master) Protocol



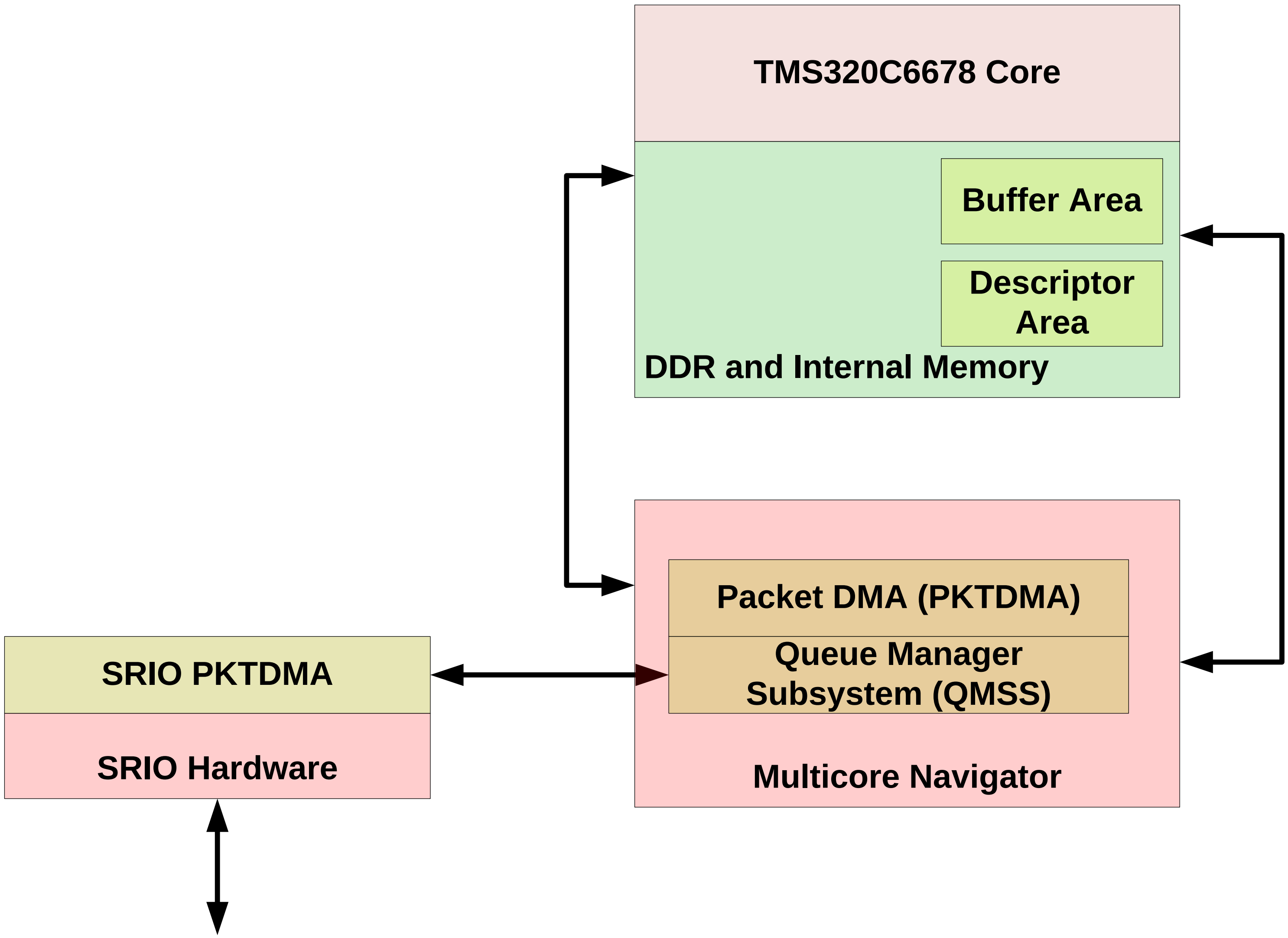
Producer = Master
Consumer = Slave

Consumer (Slave) Protocol

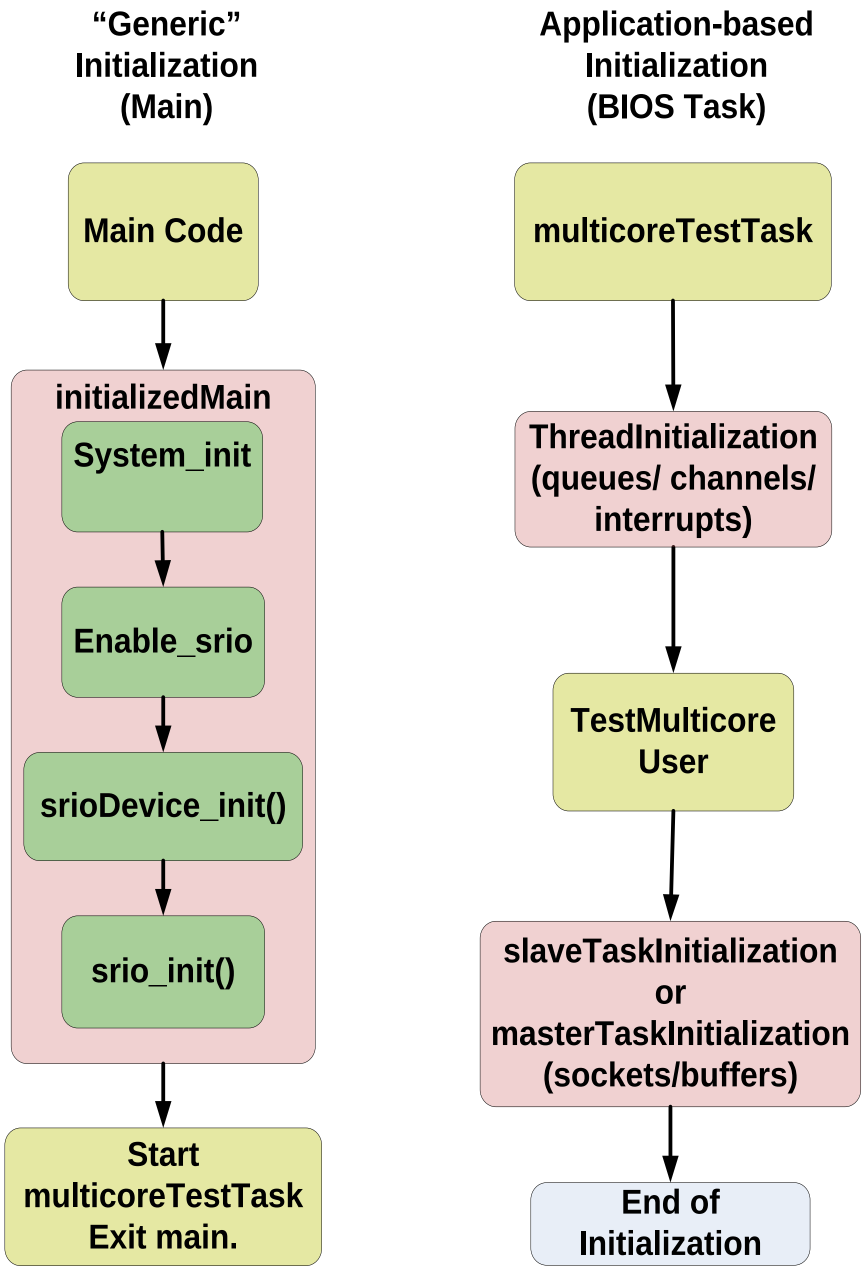


Producer = Master
Consumer = Slave

Hardware Components



Configuration/Initialization Flow



Configuration Steps:

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

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

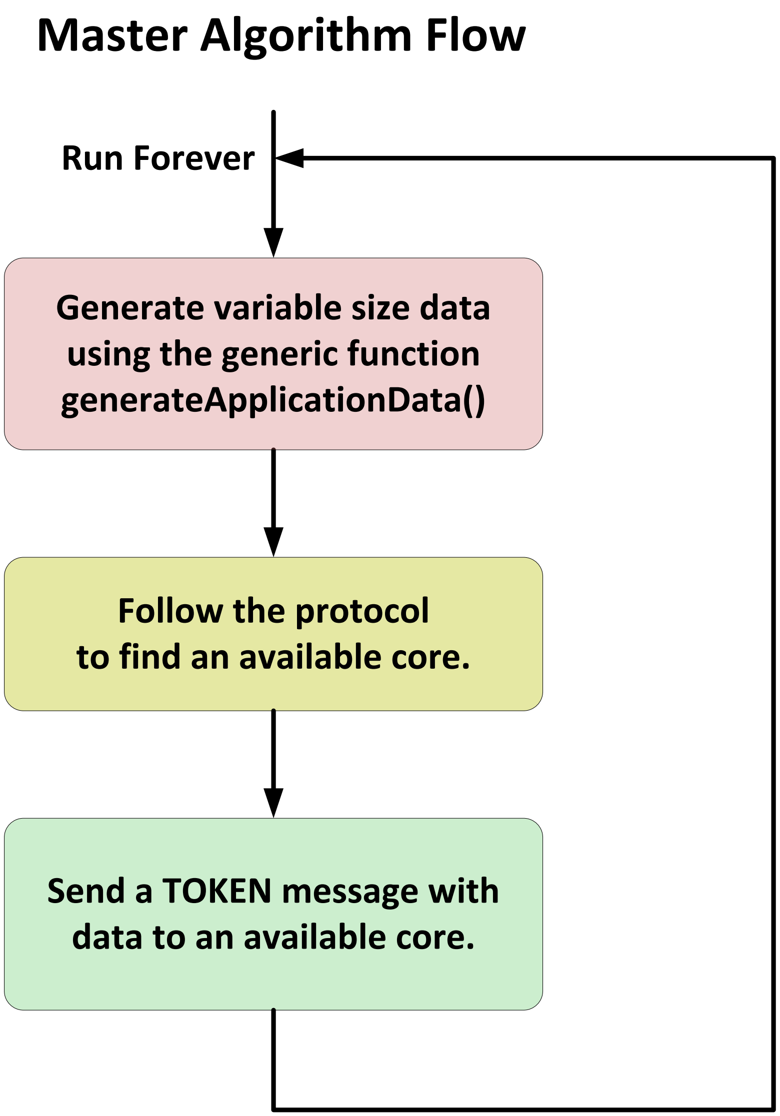
“All Cores” Initialization

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

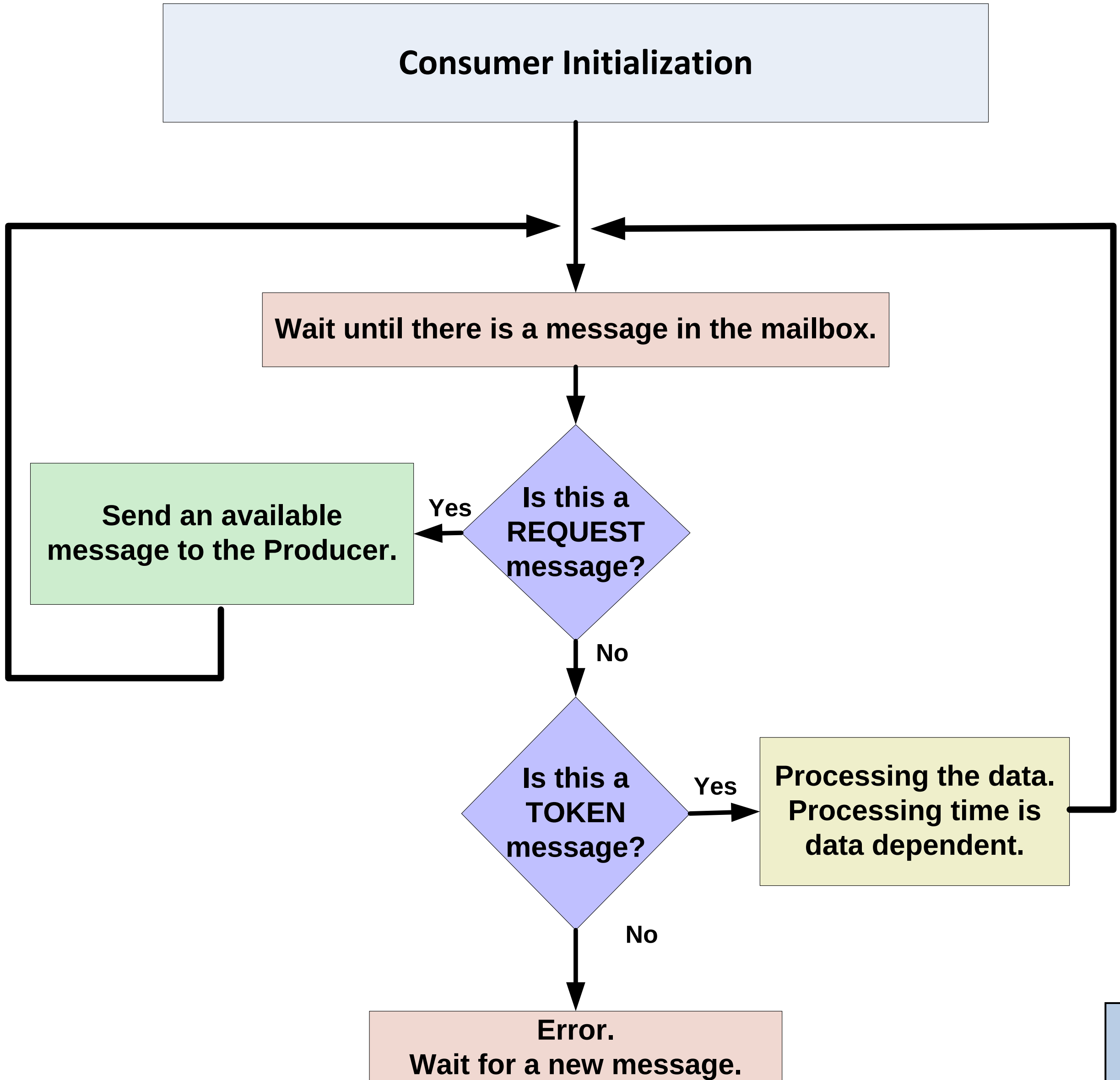
- **Srio_sockOpen()** opens a socket
- **Srio_sockBind()** binds the opened socket to routing
  - Segmentation mapping

Producer (Master) Application Algorithm



Producer = Master
Consumer = Slave

Consumer (Slave) Application Algorithm

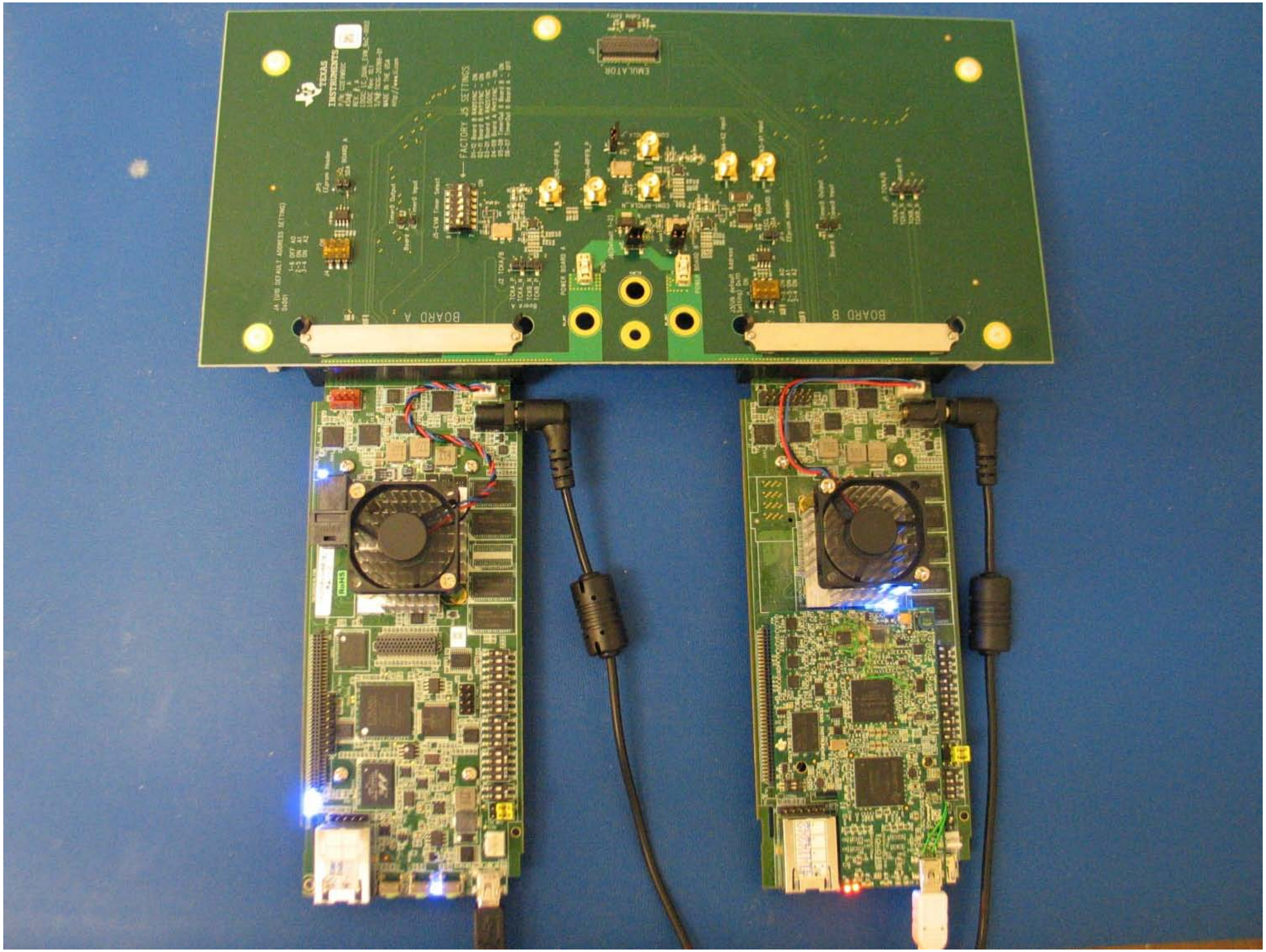


Producer = Master
Consumer = Slave

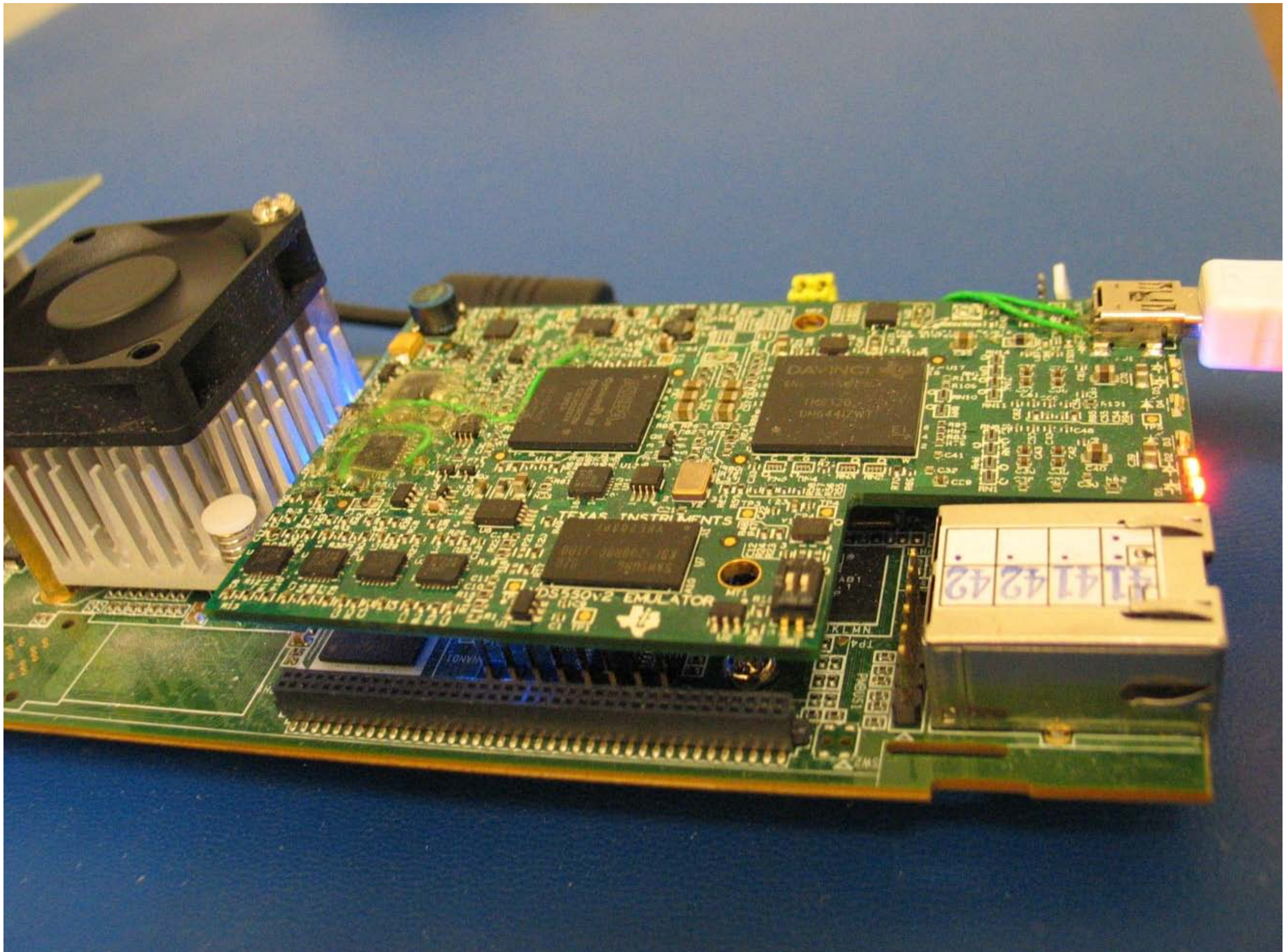
Configuring the Demo Setup

- SRIO Overview
- DirectIO Operation
- Message Passing Operation
- RapidIO Interrupts
- Understanding SRIO Implementation
- Configuring the Demo Setup
- Summary

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

Summary

- SRIO Overview
- DirectIO Operation
- Message Passing Operation
- RapidIO Interrupts
- Understanding SRIO Implementation
- Configuring the Demo Setup
- Summary

Summary

- C66x SRIO helps to deliver:
  - Higher performance
  - New transaction types
  - Less required CPU interaction per transaction
  - Better deterministic scheduling
  - More flexibility and system support with increased number of IDs
- For more information:
  - Serial RapidIO (SRIO) for KeyStone Devices User Guide
  - SRIO Online Training
  - Support forums at the TI E2E Community website