XMOSにおける送信待ち状態の回避

マスタ サブタイトルの書式設定

◎ 長谷川侑輝（東洋大学）
磯部祥尚　（産総研）
大蒔和仁　（東洋大学）
森秀樹　　(東洋大学)
土田賢省　（東洋大学）

発表内容

- 1 はじめに
- 2 送信待ち状態の問題と解決案
- 3 アルゴリズムの実装
- 4 まとめ

1. はじめに

マスタ サブタイトルの書式設定

1.1 背景
1.2 目的
1.3 結果

1.1 背景(1/2)

- 組み込みシステム
  - 携帯電話，輸送機械，大規模分散処理，etc…
- プロセッサの集積化，並行システム
- プロセッサ間の通信を保証することが重要
  - 通信における障害を回避する策が必要

1.1 背景(2/2)

- CSPを組み込みシステム開発のベースにする
  - 形式的手法　⇒　検証技術
  - 同期通信　　⇒　正確な伝送
  - 並行処理　　⇒　処理の分散

- CSPだと通信上のボトルネック
  - 送信キャンセルができない
  - ⇒送信相手故障時に送信待ち問題が発生
    - 送信待ちのままデッドロック

1.2 目的(1/2)

- CSPの通信における送信待ち問題の解消

- CSPによる耐故障性の実現

1.2 目的(2/2)

- 解決手法の提案
  - チャネル通信の送信待ちによるデッドロックの回避策
  - "部分故障→全体故障" の回避
- 実機による実装
  - XMOSによる実装
- FDRによる検証
  - モデル検査器によるCSPモデルの検証

1.3 結果

- 通信路に通信用プロセスを設置
- 隣接プロセスの故障時は通信用プロセスごと切断
  - 送信待ちによる連鎖的なデッドロック発生を回避
  - 問題個所を回避して他の正常な通信路で通信の継続
  - 復旧手続きによる故障からの回復

- FDRによる検証
  - 反例として問題の再現
  - 問題の原因究明

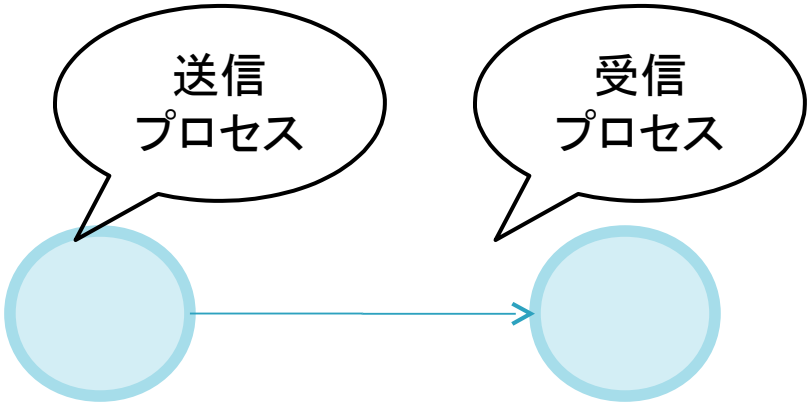
2.送信待ち状態の問題と解決案

マスタ サブタイトルの書式設定

2.1 チャネルによる通信の問題点
2.2 Linkプロセスによる解決案

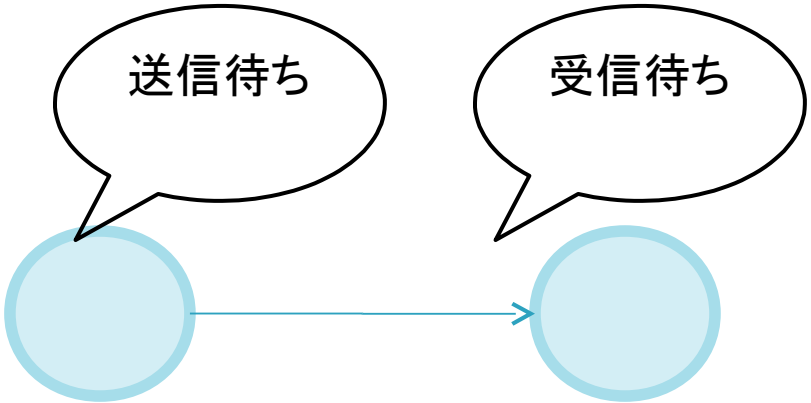
2.1 チャネルによる通信の問題点 (1/5)

- CSPモデル
  - チャネルを用いてプロセス間を同期通信



2.1 チャネルによる通信の問題点 (2/5)

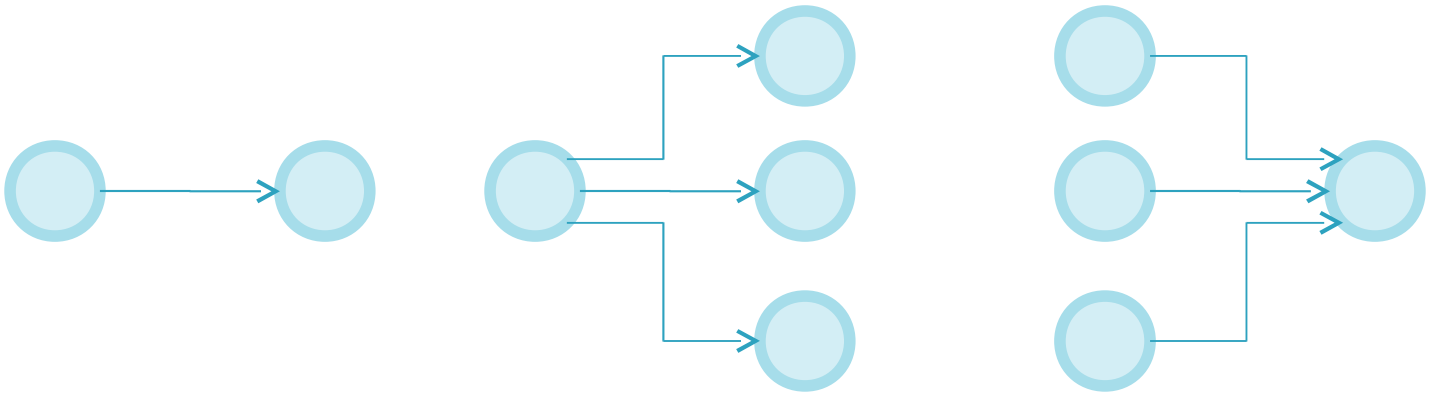
- チャネルによる通信
  - 同期通信
    - 送信待ちと受信待ちで一対となって通信が行われる
    - 送信待ちは相手が受信待ちとなるまで待つ
    - 受信待ちは相手が送信待ちとなるまで待つ



メッセージを送信相手に確実に届ける

2.1 チャネルによる通信の問題点(3/5)

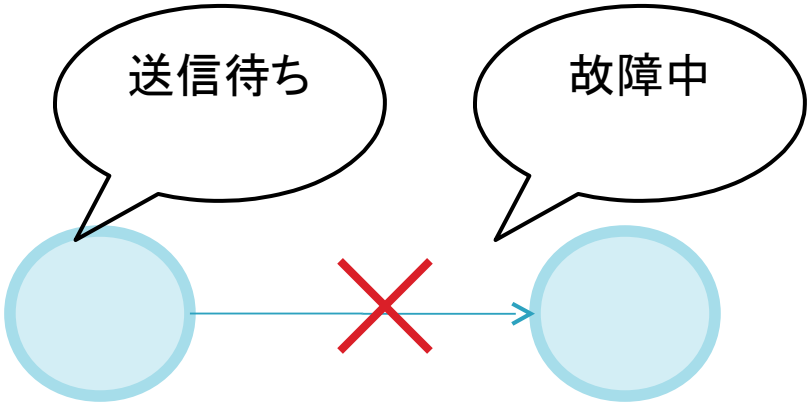
- 通信相手は常に1つ
  - 1対1 or N対1の通信
  - 1対Nの通信はできない



○1対1　×1対N　○N対1

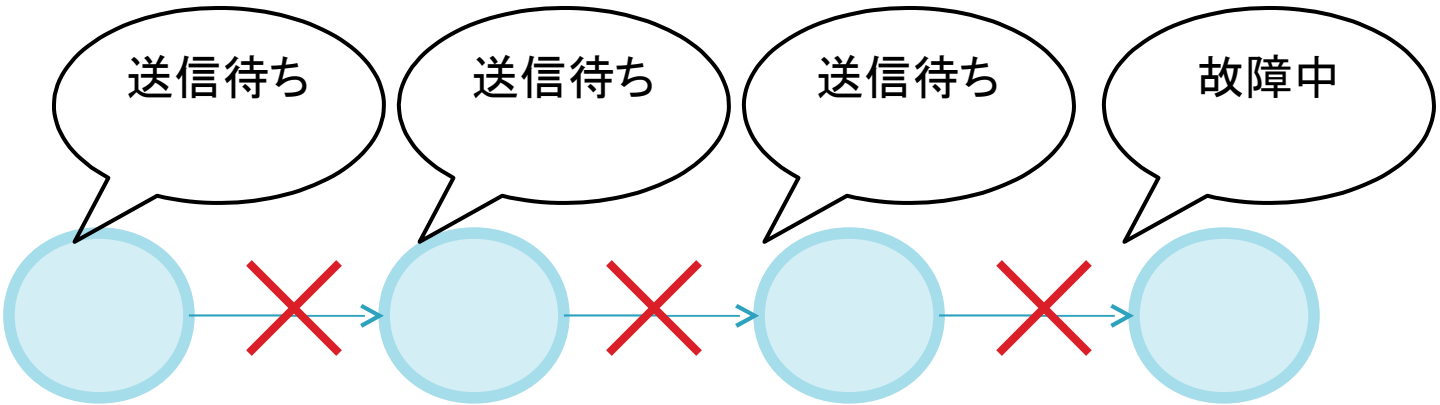
2.1 チャネルによる通信の問題点(4/5)

- チャネルによる送信はキャンセルできない
  - ⇒デッドロック発生の原因
  - ※受信はキャンセルが可能



2.1 チャネルによる通信の問題点 (5/5)

- 送信先のプロセスに異常があると，送信元プロセスにまでデッドロックによる処理停止の危険性
  - 異常プロセスがいつまでも受信待ちにならず，送信元プロセスは送信待ち状態を維持してデッドロックとなる



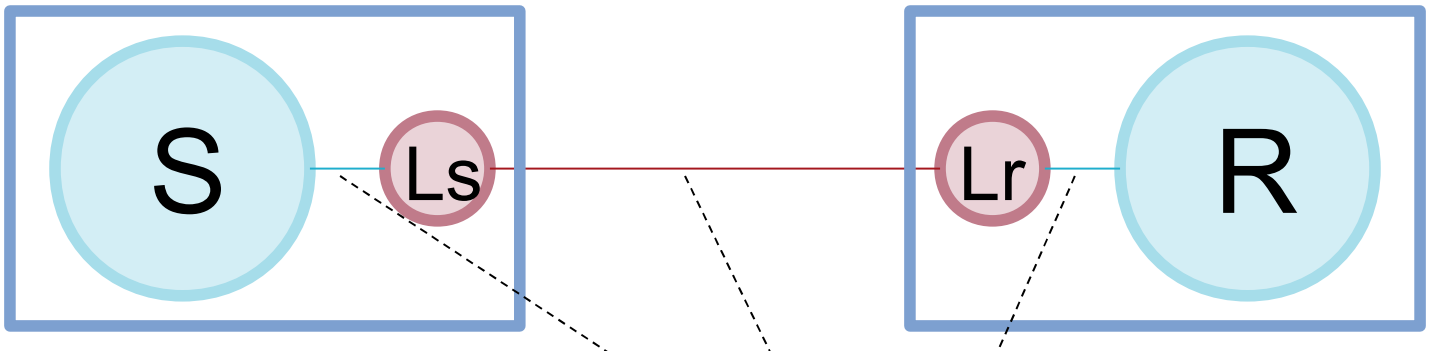
システム全体がデッドロックになる危険性！

2.2 Linkプロセスによる解決案 (1/15)

- 問題
  - 通信相手や通信経路の故障発生時
  - ⇒送信待ち状態でデッドロック
  - ⇒故障範囲がシステム全体に広がる
  - "部分故障→全体故障"の危険性

2.2 Linkプロセスによる解決案 (2/15)

- 通信を行うプロセス間に伝送用プロセスの配置
  - プロセス間の通信を専用のプロセス(Linkプロセス)で代行

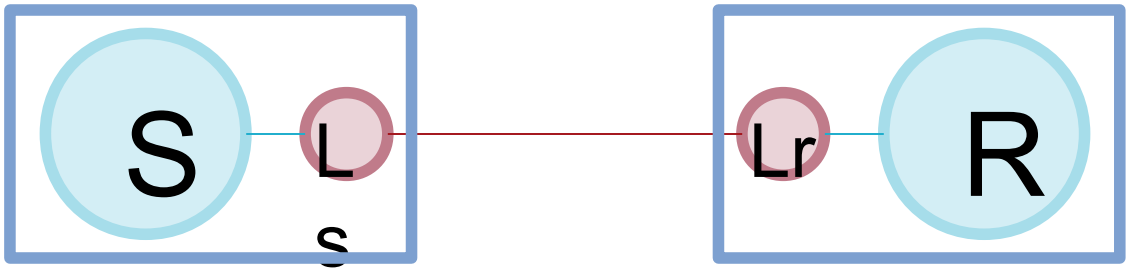


- S ：送信プロセス
- Ls：Linkプロセス
- Lr：Linkプロセス
- R ：受信プロセス

- 内部通信チャネル
- 伝送用チャネル

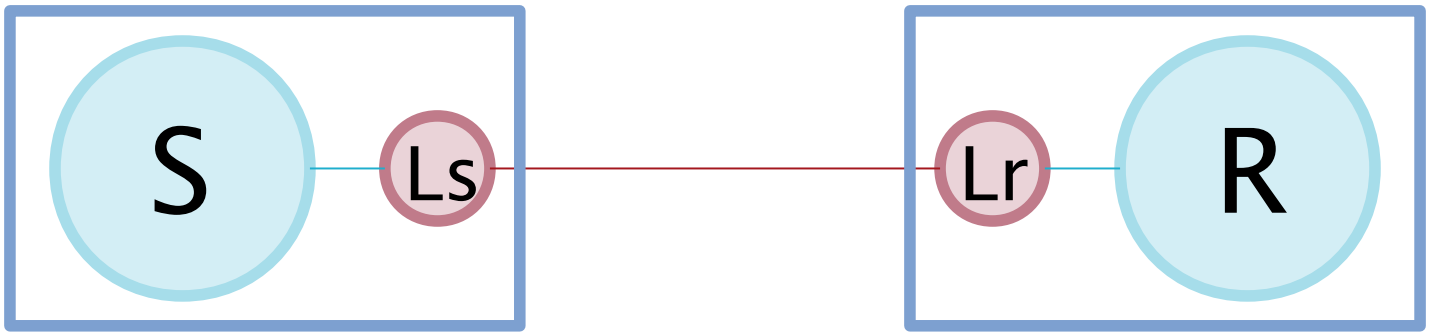
2.2 Linkプロセスによる解決案 (3/15)

- 定義
  - 送信プロセスと受信プロセスは，通信相手1つに対してLinkプロセスを1つ持つ
  - 通信はLinkプロセスを介して行われる
  - 故障発生はSとLsのRとLrの組などで発生するものとする



2.2 Linkプロセスによる解決案 (4/15)

- 送信時：自身のLinkプロセスに送信
  - Linkプロセスは正常時に受信待ちで待機のため，Sendプロセスと Linkプロセス間の通信は即時に行われる



2.2 Linkプロセスによる解決案 (5/15)

- 送信時：自身のLinkプロセスに送信
  - Linkプロセスは送信先のLinkプロセスより送信完了の信号を受け，Sendプロセスへ送信完了の旨を伝える

2.2 Linkプロセスによる解決案 (6/15)

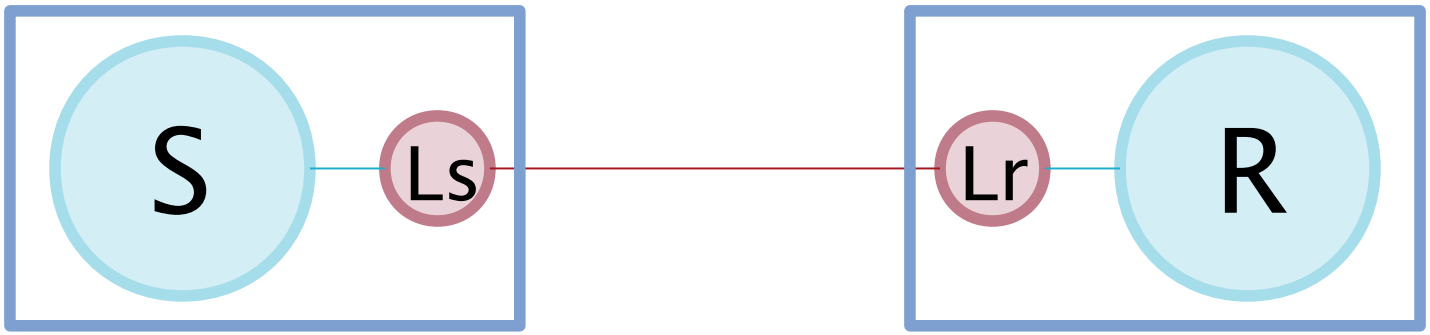
- 送信時：自身のLinkプロセスに送信
  - 送信先が異常などで停止している時，Linkプロセス自身はデッドロックとなるが，Sendプロセスはタイムアウトを用いてLinkプロセスからの受信をキャンセルすればデッドロックにはならない

2.2 Linkプロセスによる解決案 (7/15)

- 送信時：自身のLinkプロセスに送信
  - 前回の送信完了有無を記憶しておけば，デッドロックに陥ったLinkプロセスへの送信は回避することが可能

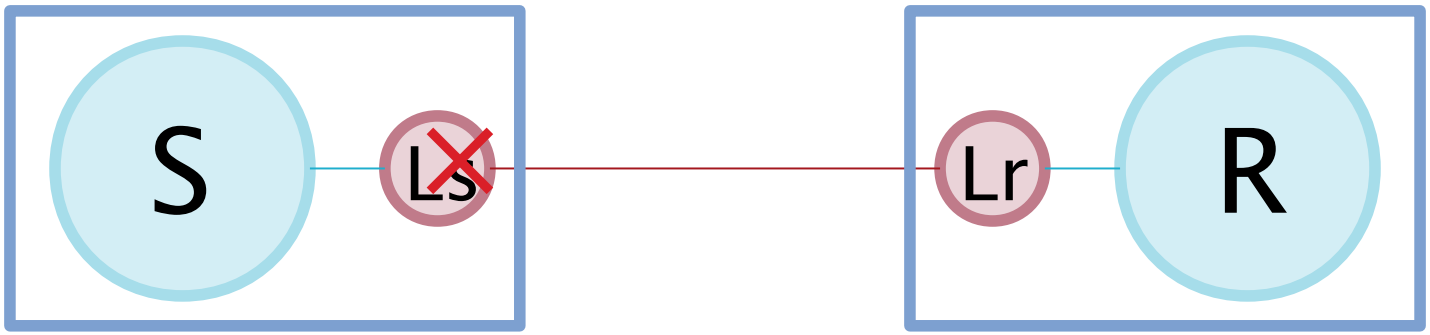
2.2 Linkプロセスによる解決案 (8/15)

- 受信時：自身のLinkプロセスに対して受信待ち
  - Linkプロセスがデータを受信していれば，そのデータをReceiveプロセスへ転送する

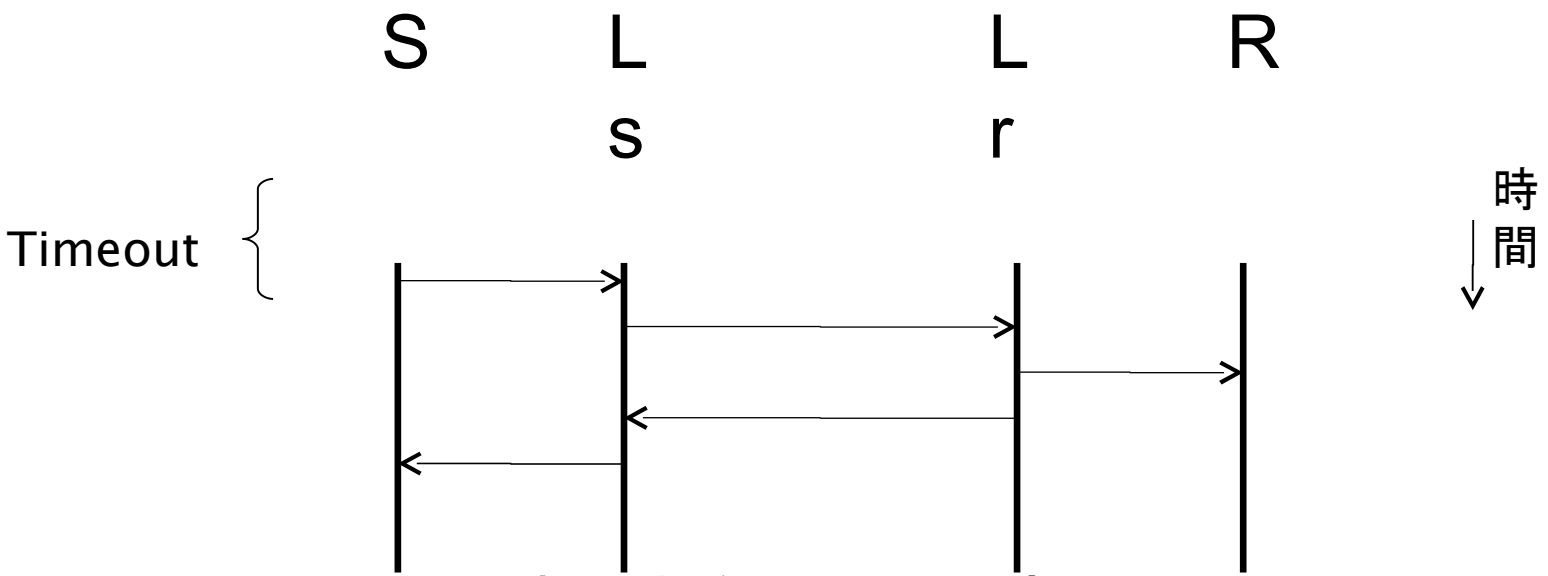


2.2 Linkプロセスによる解決案 (9/15)

- 受信時：自身のLinkプロセスに対して受信待ち
  - 送信側プロセスの異常に対しては，タイムアウトによる受信キャンセルを行えばReceiveプロセスのデッドロックにはならない

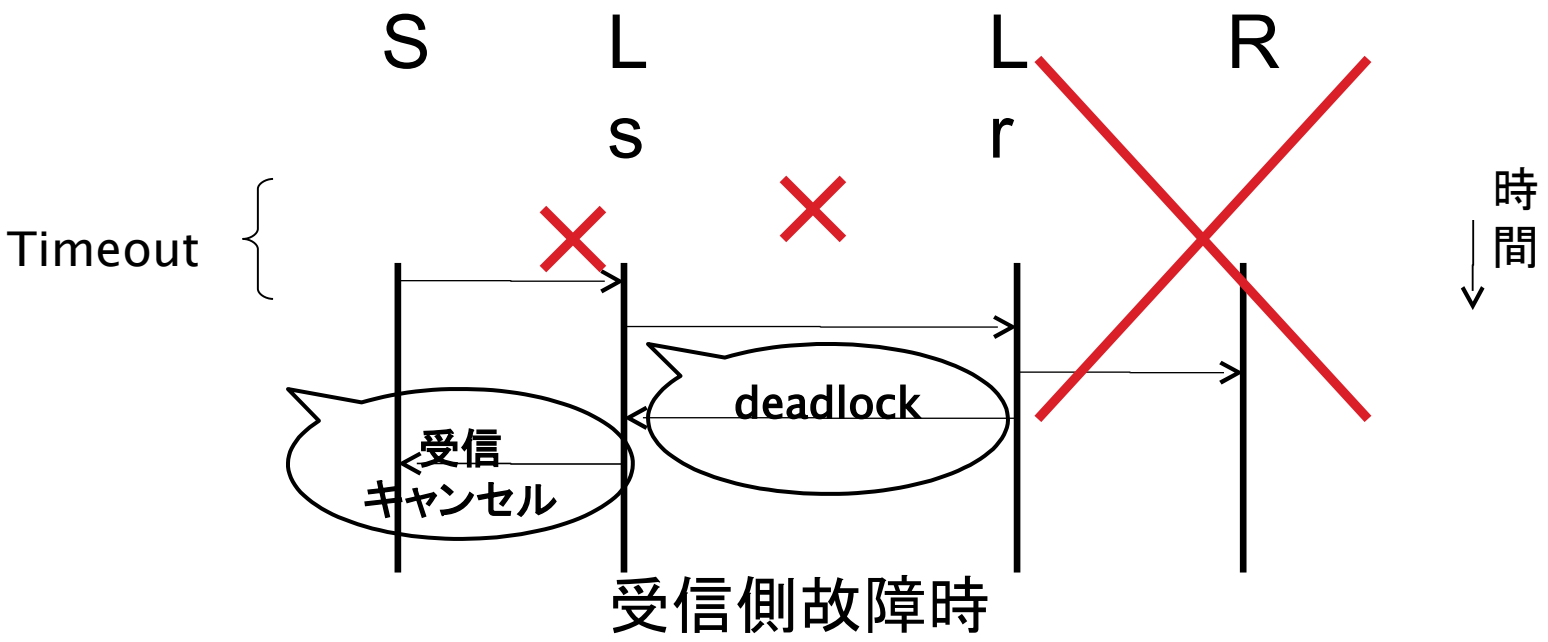


2.2 Linkプロセスによる解決案 (10/15)

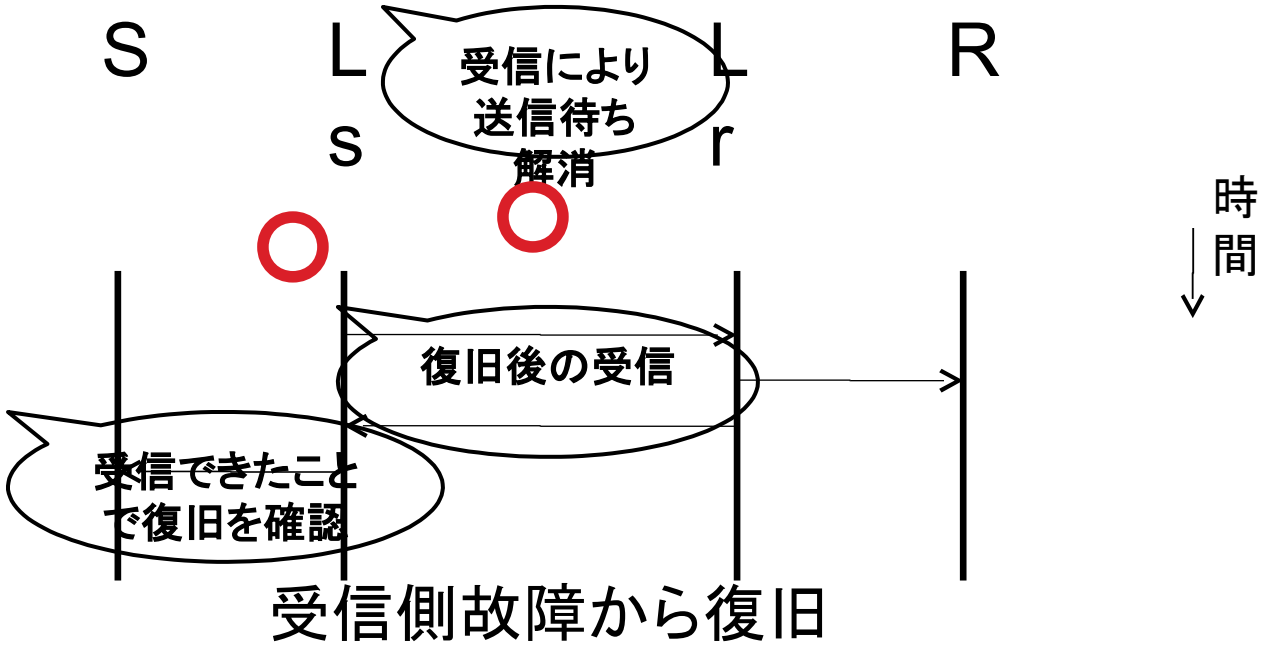


時間方向と通信順序の関係

2.2 Linkプロセスによる解決案 (11/15)

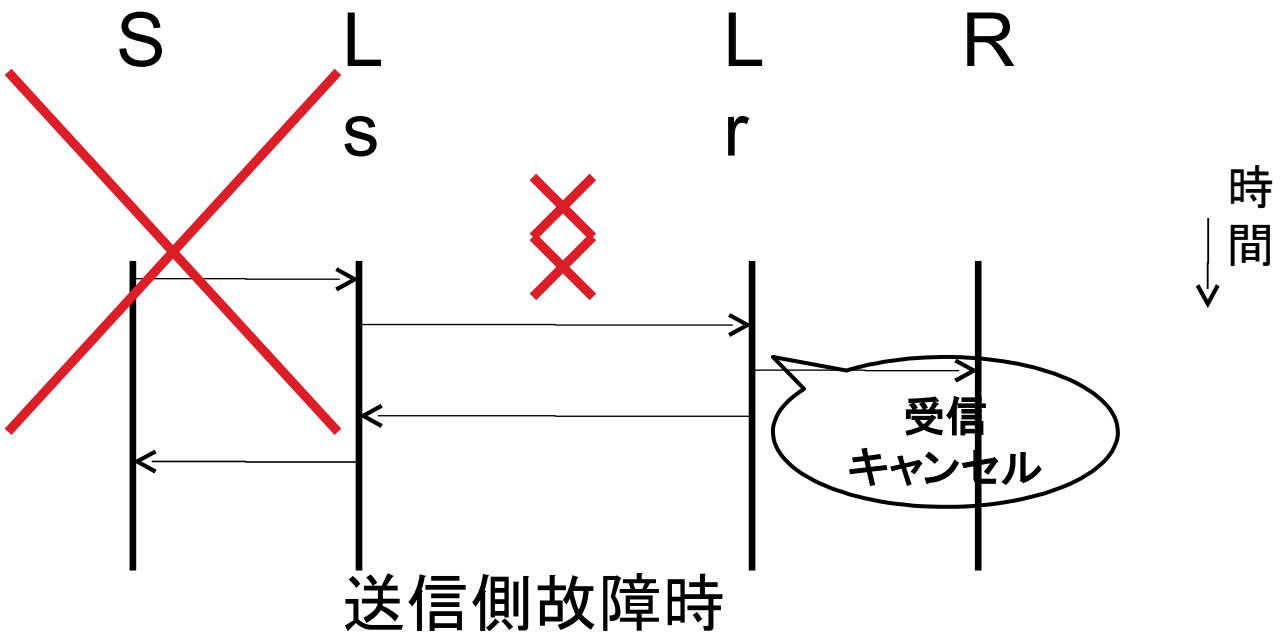


2.2 Linkプロセスによる解決案 (12/15)



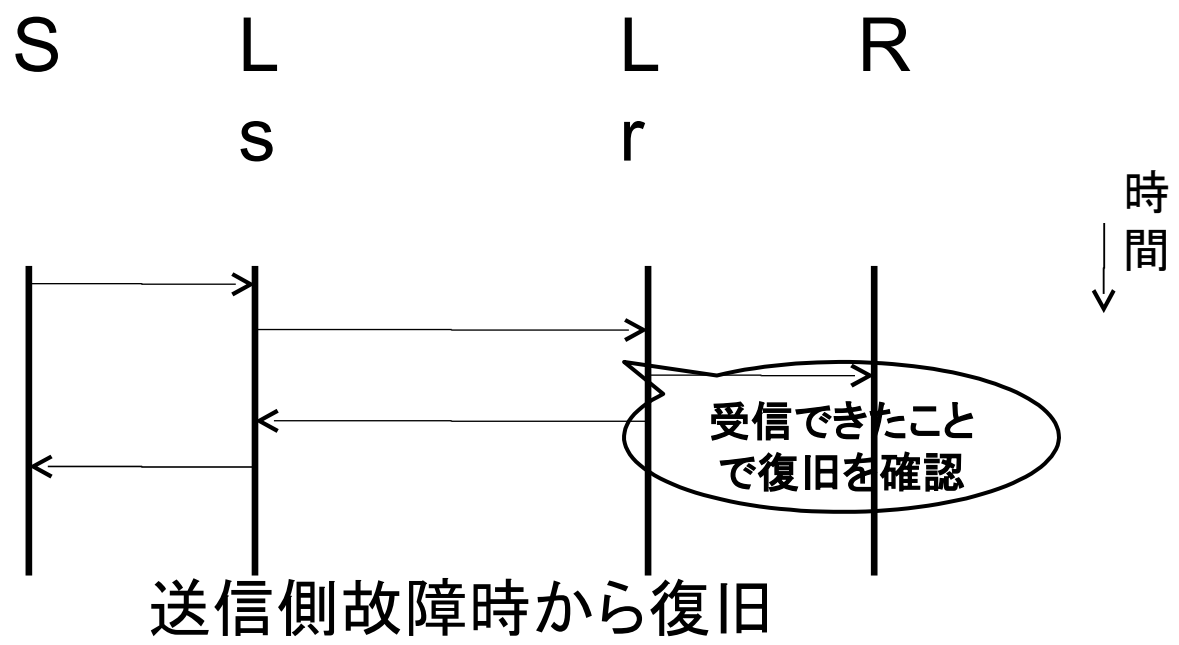
受信側故障から復旧

2.2 Linkプロセスによる解決案 (13/15)



送信側故障時

2.2 Linkプロセスによる解決案 (14/15)



送信側故障時から復旧

2.2 Linkプロセスによる解決案 (15/15)

S a_ch1 ! data; a_ch1 ? data; タイムアウトによる 受信キャンセル	R a_ch2 ? data;
Ls a_ch1 ? data; link_ch ! data; link_ch ? data; a_ch1 ! data;	Lr link_ch ? data; a_ch2 ! data; link_ch ! data;

故障復旧のCSP記述

3.アルゴリズムの実装

マスタ サブタイトルの書式設定

3.1概要
3.2 アルゴリズムの仕様
3.3 実装結果
3.4 FDRによる検証

3.1 概要(1/2)

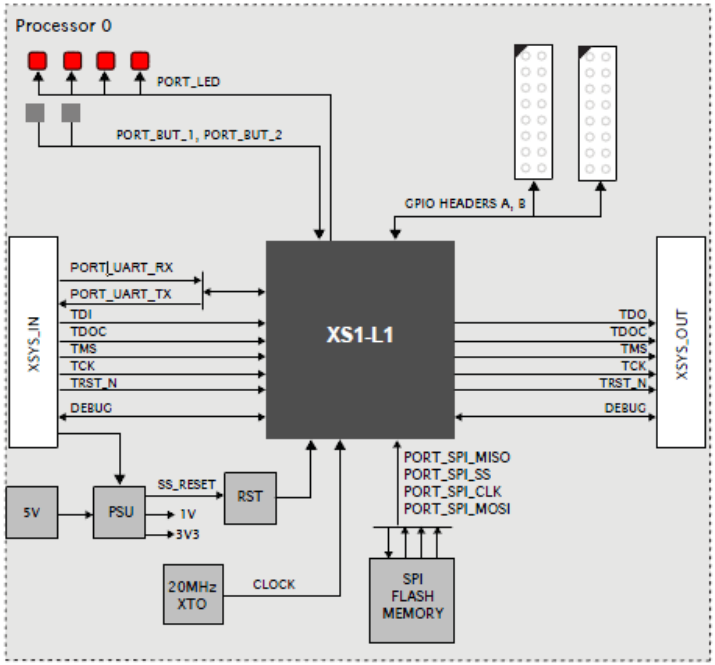
- XMOSでLinkプロセスを用いたプロセス間通信の実装を行う
  - CSPモデルによるモデル化
  - XMOSの評価ボードXK-1で実行

  - 送受信を行うプロセスとLinkプロセスは同一コア内で実行
  - 故障はコア単位で発生することを想定

3.1 概要(2/2)

- XMOSチップを搭載した評価ボードXK-1を使用

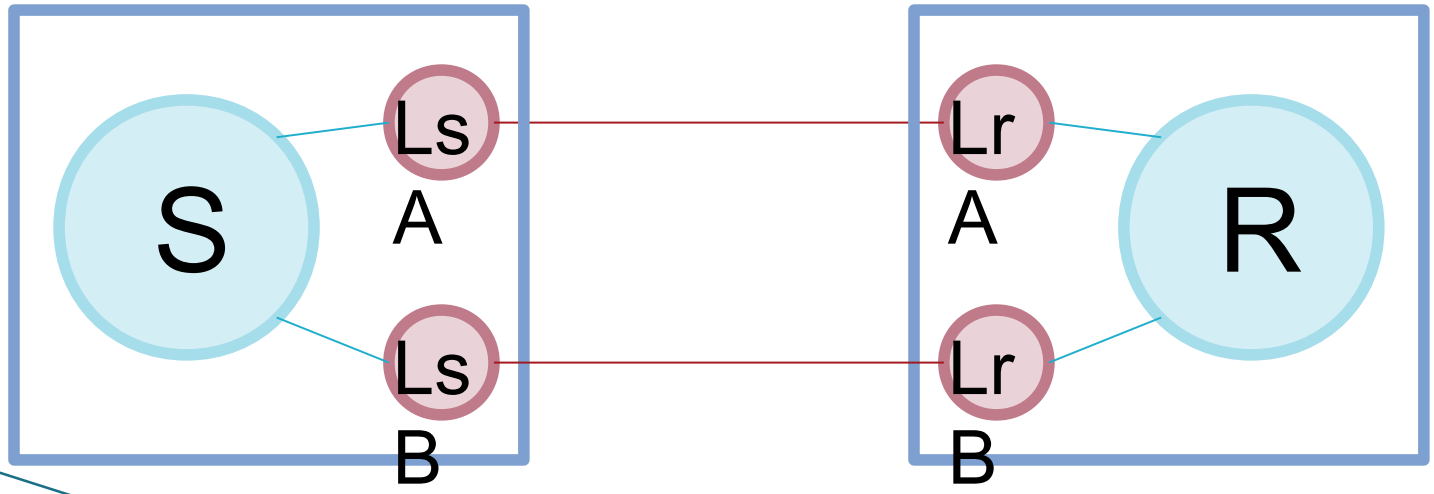
XK-1のブロック図



参考文献:XMOS Limited, "XMOS - Revolutionising Electronics", http://www.xmos.com/.

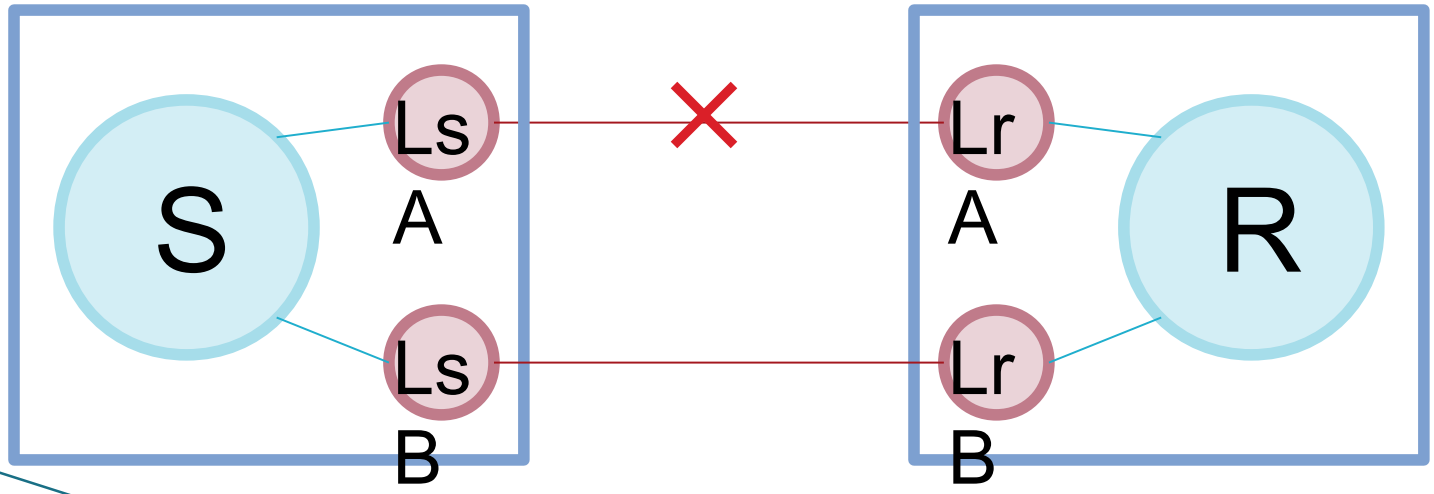
3.2 アルゴリズムの仕様(1/4)

- 送信プロセスSはLinkプロセスLsAとLsBを介し，受信プロセスRへ二重送信する
  - データは非負の整数を送信側で生成するものとする



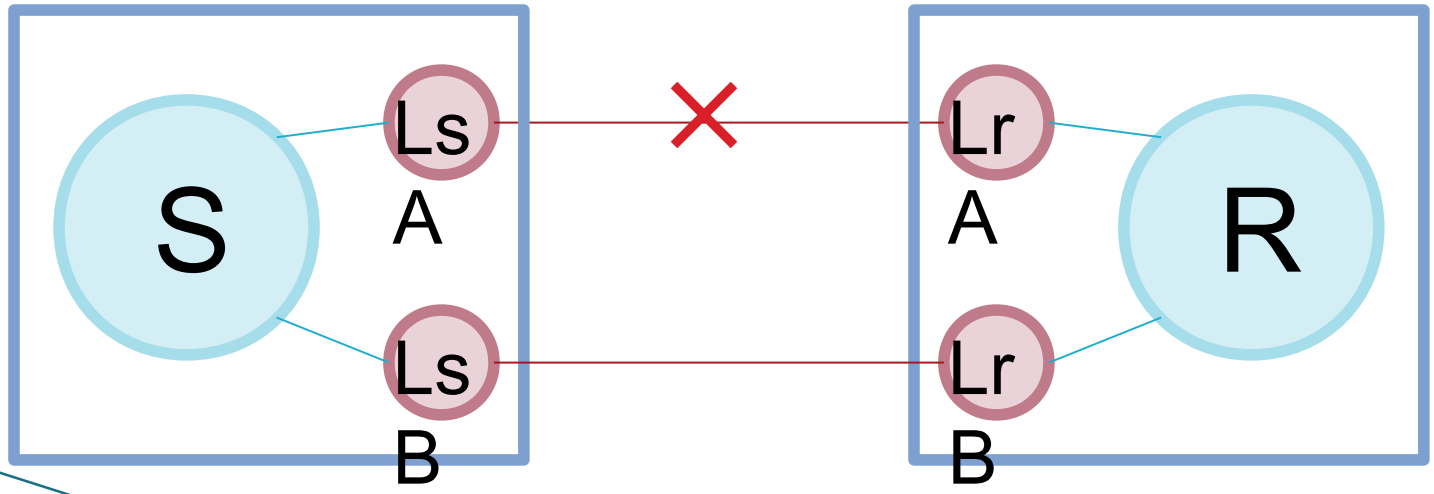
3.2 アルゴリズムの仕様(2/4)

- 通信において送信側にはタイムアウト時間が設定されており，時間内に通信が完了しなければ通信相手の故障とみなす
- 復旧するまではその相手との通信を行わない



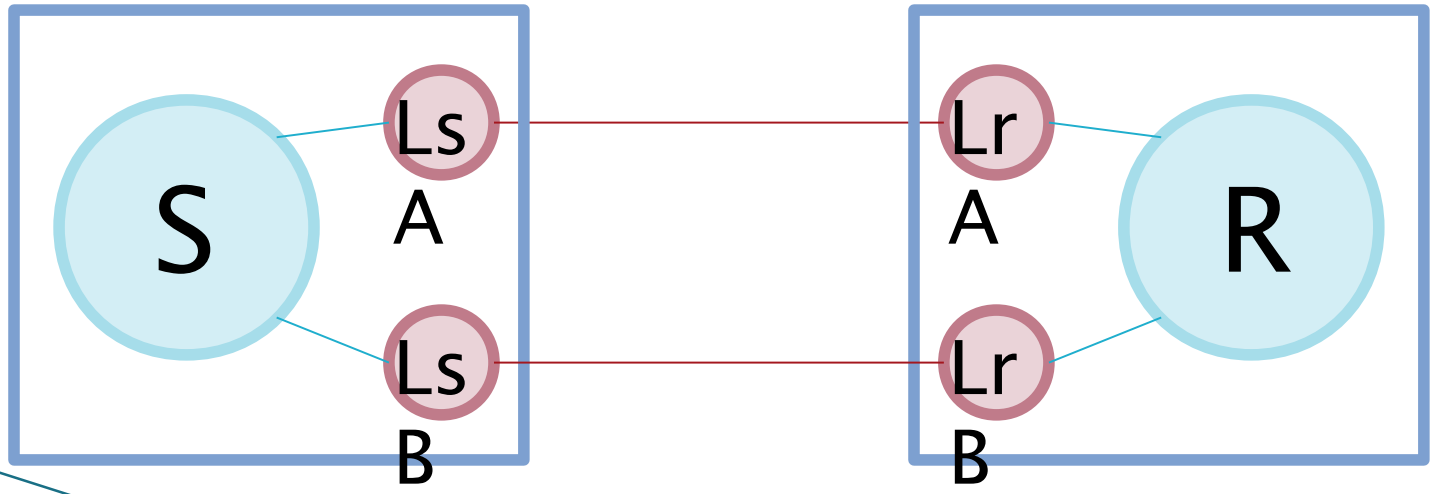
3.2 アルゴリズムの仕様(3/4)

- 二重通信のうち一方が正常ならば，通信は継続
- 故障個所を復旧させることで再び二重通信が行える

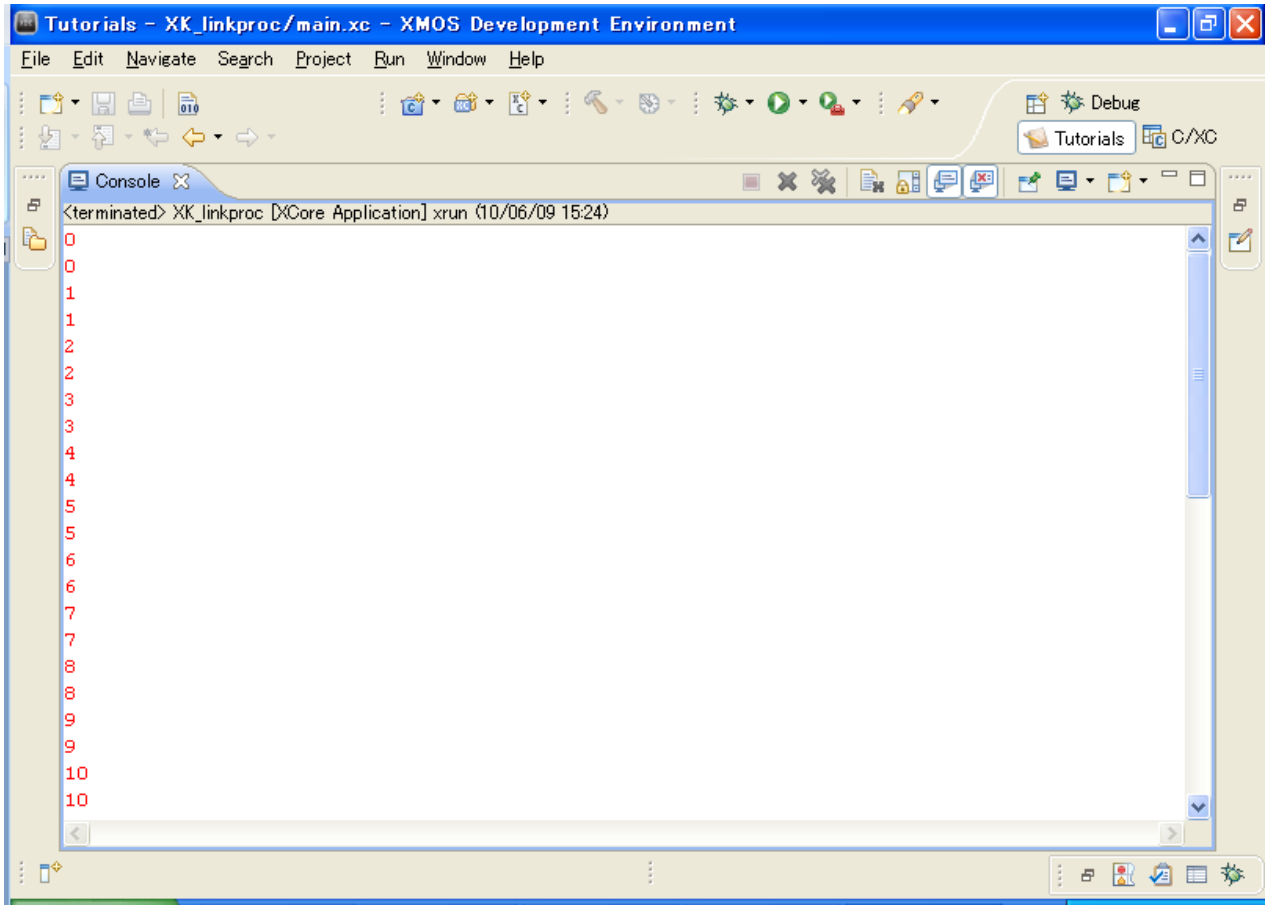


3.2 アルゴリズムの仕様(4/4)

- 追加仕様
  - 意図的に異常状態にするべく，今回はボタンをトリガとして内部で無限ループに入る状態を作成
    - 意図的なデッドロックの発生
  - 再度ボタンを押すことで復旧



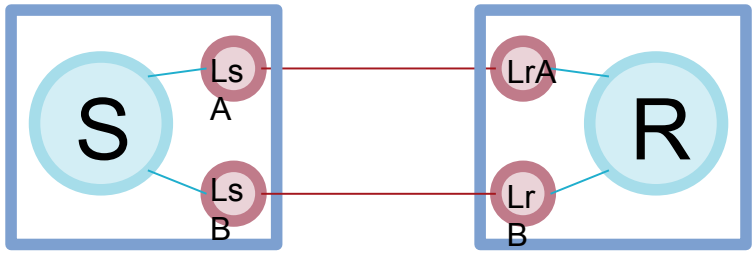
3.3 実装結果(1/2)



二重通信の実装結果

3.3 実装結果(2/2)

- 動作確認
  - Linkプロセスを用いた二重通信
    - A(LsA→LrA)とB(LsB→LrB)
  - 通信路故障時に送受信プロセスのデッドロック回避
    - ボタンを押しても送信プロセスはデッドロックにならない
  - 正常な通信路による通信の継続
    - もう一方の通信路で通信を継続
  - 復旧手続きによる故障からの回復
    - 再度ボタンを押すことで二重通信の再開



3.4 FDRによる検証(1/7)

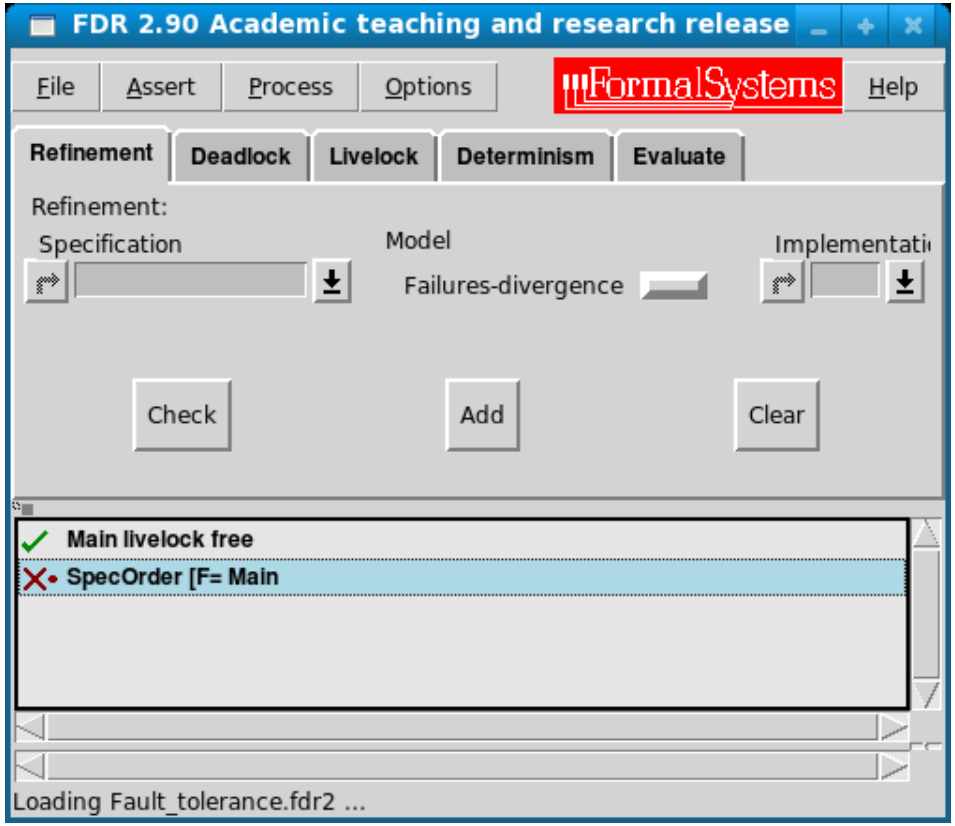
- 通信順序によって送信順序と受信順序のタイミングがずれる現象を確認

- ⇒人力で原因を探すことは難しい

3.4 FDRによる検証(2/7)

- CSPモデルをモデル検査器FDRで検証
- ⇒反例として現象を再現

3.4 FDRによる検証(3/7)



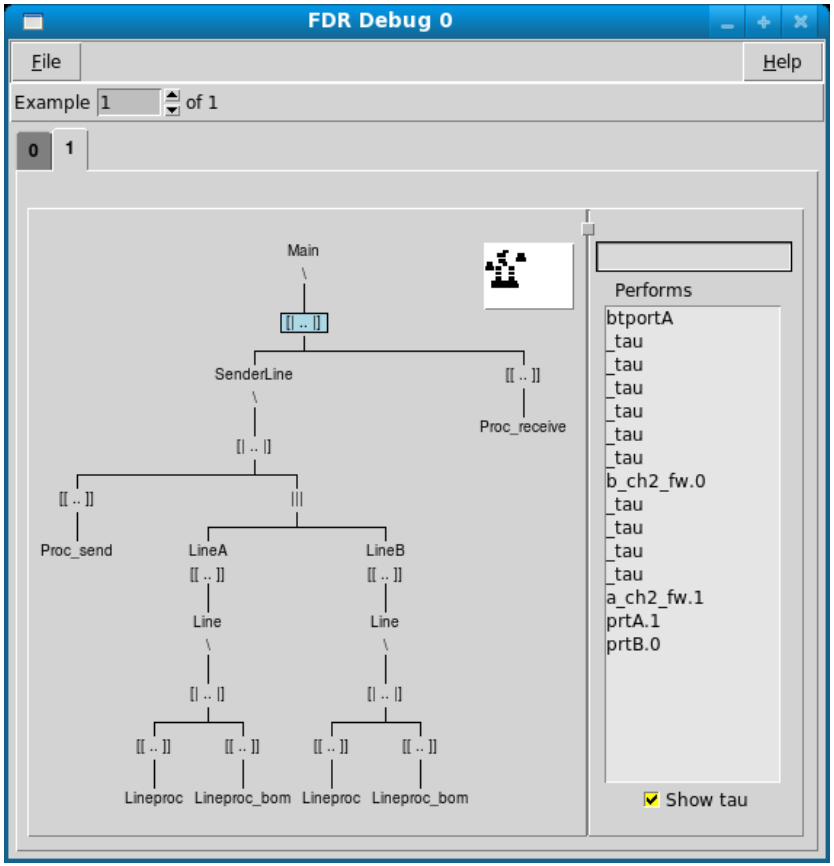
- FDRによる検証

3.4 FDRによる検証(4/7)

- FDRによって偽の判定が出た場合、デバッガにて解析を行う

- ⇒仕様を満たさない反例の出力

3.4 FDRによる検証(5/7)

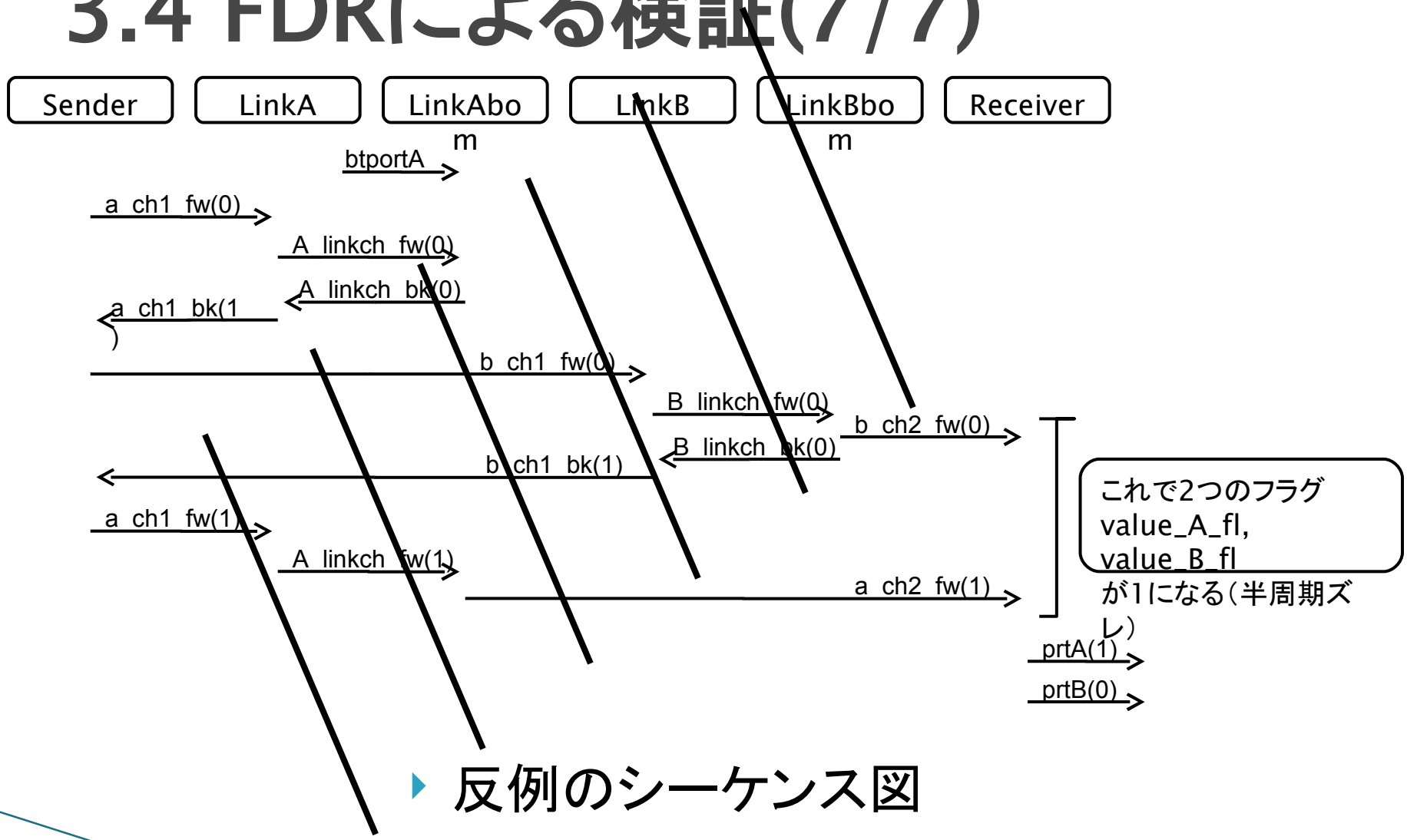


- FDRのデバッガによる反例の表示

3.4 FDRによる検証(6/7)

- シーケンス図より反例を調べる
- ⇒受信側のタイミングが半期ずれる可能性があることを確認

3.4 FDRによる検証(7/7)



- 反例のシーケンス図

4.まとめ

マスタ サブタイトルの書式設定

4.1 考察
4.2 まとめ

4.1 考察(1/2)

- 評価ボードを用いたアルゴリズムの提案と実装
- 送信先の故障発生時も送信プロセスはデッドロックによる停止することなく動作

- ⇒送信キャンセルができない環境でも、送信待ちによるデッドロックを回避

耐故障性

4.1 考察(2/2)

- 仕様で曖昧であった送信順序に関する障害
- CSPモデルを検証できるFDRで検証
- ⇒反例により原因の究明

システム開発において
検証の効果は大きい

4.2 まとめ(1 / 3)

- CSPの通信における送信待ち問題の解決アルゴリズムを提案
- 提案アルゴリズムをXMOSを用いて実装実験
- FDRによるCSPモデルの検証

4.2 まとめ(2/3)

- 送信待ちによるデッドロックの回避
- 部分故障から送信待ちの連鎖による全体故障を抑制
- 部分的な故障を隔離することでシステム処理の継続

4.2 まとめ(3/3)

- 今後の課題
  - さらに複雑なトポロジにおける提案手法の適応
  - 故障とみなすまでのタイムアウト時間に関する評価と最適化
  - ハードウェア記述言語による耐故障チャネルの実装
  - etc…

おわり

- ご静聴ありがとうございました.