Triple Modular Redundancy based on Runtime Reconfiguration and Formal Models of Computation

10th Aerospace Technology Congress, Stockholm

Ricardo Bonna
rbonna@fem.unicamp.br

Denis Loubach
dloubach@ita.br

Ingo Sander
ingo@kth.se

Ingemar Söderquist
ingemar.soderquist@saabgroup.com

8-9 October, 2019

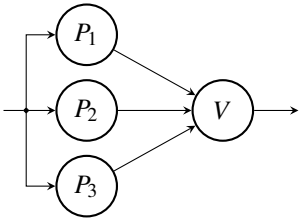
Summary

- Runtime reconfiguration (RTR) is one promising way to mitigate for increased failure rate in future safety-critical avionics systems
- RTR is one of the big challenges for the future generation of integrated modular avionic (IMA) systems
- In the event of a hardware failure, the system is able to reallocate the functionalities from the faulted module into a safe module
- One the most important component of a runtime reconfigurable safety-critical system is the *fault detection mechanism*
- One of such fault detection mechanisms is the Triple modular Redundancy (TMR)

Introduction

- Challenges of the paper
  - Detect when and where a fault occurred
  - Reconfigure a new module to replace the faulted module
  - Do it without stopping the overall process
- Proposed solution
  - TMR to detect faults
  - RTR to reconfigure a new module
  - Shared memory to synchronize newly reconfigured module

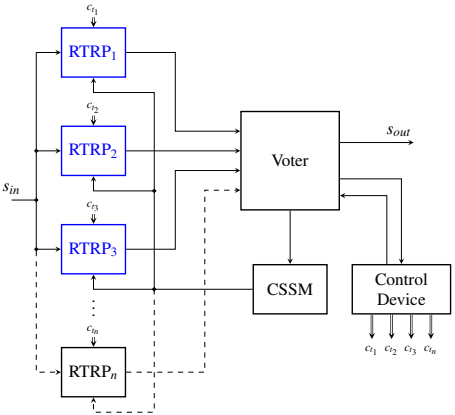
- Three processes P_x executes the same functionality
- A voting mechanism V selects the output that most occur
- The system is immune to a single process failure
- Voter is a single point of failure
- If two processes fails, the system fails



- *Models of computation* (MoCs) are a collection of rules dictating the semantics of execution and concurrency in computational systems.
- A common framework to classify and compare different MoCs is the *tagged signal model*
- A signal $s \in S$ is a set of events $e_i = (t_i, v_i)$ composed by a tag $t_i \in T$ and a value $v_i \in V$
- A process P is a set of possible behaviors that defines relations between input signals $s_i \in S^I$ and output signals $s_o \in S^O$
- MoCs are classified as *timed* or *untimed*
- A *synchronous model* is a timed model in which every event has a matching tag event in every signal
- An event of a synchronous signal s is represented by $s[k]$, where k is its tag

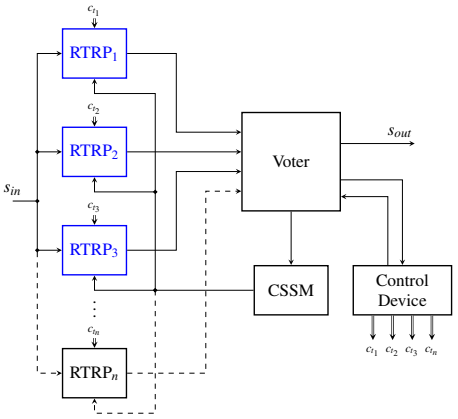
Modeling TMR with RTR

- The triple modular redundancy is used to detect and mask possible faults
- The architecture can mask multiple permanent faults, provided no two faults occur in a small time window defined by the reconfiguration time
- Similar to TMR with spares, but each "spare" processor is initially executing something else



Modeling TMR with RTR

- The elementary processing unit is called Runtime Reconfigurable Process (RTRP)
- A control device is responsible for reconfiguring the RTRPs when needed
- A Voter detects which RTRP is faulted and signalizes the Control Device
- A Current State Shared Memory is used as a high level memory to synchronize the RTRPs after a reconfiguration



Probability of system failure

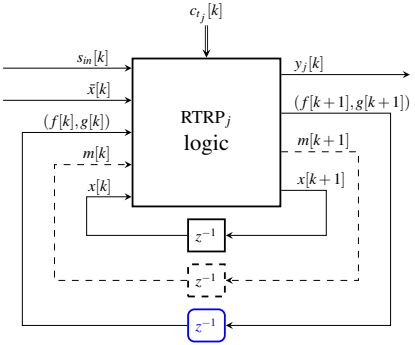
- The system fails if
  - Either the Voter or the Control Device fails
  - Multiple RTRPs fails in a time window smaller than the reconfiguration time
- Suppose that each RTRP has a probability of failure ρ every cycle
- Suppose also that it takes m cycles to reconfigure an RTRP
- The probability of failure of the system is

$$P = \rho(1-(1-\rho)^{2(m+1)})$$

- The faster it is to reconfigure a new RTRP, the less vulnerable the system is

Runtime Reconfigurable Process (RTRP)

- A Runtime Reconfigurable Process (RTRP) is the elementary data processing element
- c_t defines the configuration (f, g), which is stored in a configuration memory
- It takes $\mathbf{m}$ cycles to reconfigure an RTRP
- s_{in} is the input and y is the output
- x is the current state stored in a local (cache) memory
- The first time an RTRP executes, its initial states are inputted through $\bar{x}$



Runtime Reconfigurable Process (RTRP)

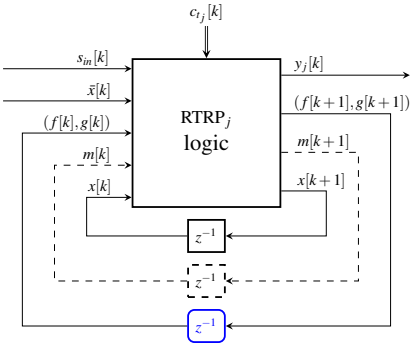
- RTRP internal logic:

$$(f[k+1], g[k+1]) = \begin{cases} (\mathbf{f}, \mathbf{g}) & \text{if } c_t[k] = (\mathbf{f}, \mathbf{g}, \mathbf{m}) \\ (f[k], g[k]) & \text{if } c_t[k] = \perp \end{cases}$$

$$m[k+1] = \begin{cases} \mathbf{m} - 1 & \text{if } c_t[k] = (\mathbf{f}, \mathbf{g}, \mathbf{m}) \\ m[k] - 1 & \text{if } c_t[k] = \perp \text{ and } m[k] > 0 \\ 0 & \text{if } c_t[k] = \perp \text{ and } m[k] = 0 \end{cases}$$

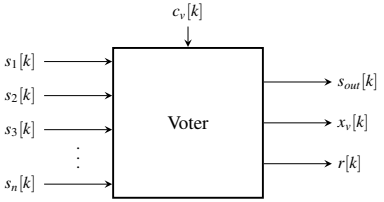
$$x[k+1] = \begin{cases} \perp & \text{if } c_t[k] \neq \perp \text{ or } m[k] > 0 \\ f[k](\bar{x}[k], s_{in}[k]) & \text{else if } x[k] = \perp \\ f[k](x[k], s_{in}[k]) & \text{otherwise} \end{cases}$$

$$y_j[k] = \begin{cases} \perp & \text{if } c_t[k] \neq \perp \text{ or } m[k] > 0 \\ g[k](\bar{x}[k], s_{in}[k]) & \text{else if } x[k] = \perp \\ g[k](x[k], s_{in}[k]) & \text{otherwise} \end{cases}$$



Voter

- The Voter is responsible for detecting and masking faults
- c_v signalizes the Voter the three active RTRPs
- s_{out} is the most occurring outputs from the three RTRPs
- x_v is the state vector to be stored in the CSSM for memory synchronization
- r is the reconfiguration request to signalize the Control Device the faulted RTRP



Voter

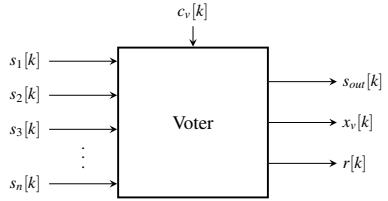
- Defining c_v and s_j as:

$$c_v[k] = (a,b,c), \quad a,b,c \in \{1,2,\ldots,n\}$$
$$s_j[k] = (y_j[k], x_j[k]), \quad j \in \{1,2,\ldots,n\}$$

- Outputs s_{out}, x_v and r:

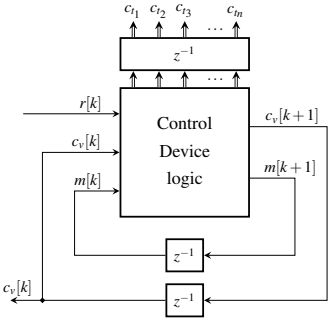
$$(s_{out}[k], x_v[k]) = \begin{cases} (y_a[k], x_a[k]) & \text{if } y_a[k] = y_b[k] \\ & \text{or } y_a[k] = y_c[k] \\ (y_b[k], x_b[k]) & \text{if } y_b[k] = y_c[k] \end{cases}$$

$$r[k] = \begin{cases} \perp & \text{if } y_a[k] = y_b[k] = y_c[k] \\ a & \text{if } y_a[k] \neq y_b[k] = y_c[k] \\ b & \text{if } y_b[k] \neq y_a[k] = y_c[k] \\ c & \text{if } y_c[k] \neq y_a[k] = y_b[k] \end{cases}$$



Control Device

- The Control Device is responsible for reconfiguring the faulted RTRP signalized by the Voter
- r signalizes the faulted RTRP
- c_v signalizes the Voter the current three active RTRPs
- m is a countdown timer to mark the reconfiguration time of a new RTRP
- c_{t_j} sends configuration functions f and g to a new RTRP



Control Device

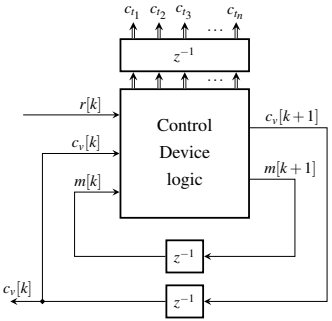
- Control Device's logic:

$$m[k+1] = \begin{cases} \mathbf{m} & \text{if } r[k] \neq \perp \text{ and } m[k] = 0 \\ m[k]-1 & \text{if } m[k] > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$c_v[k+1] = \begin{cases} c_v[k] & \text{if } r[k] = \perp \text{ or } m[k] > 0 \\ h(c_v[k], r[k]) & \text{if } r[k] \neq \perp \text{ and } m[k] = 0 \end{cases}$$

with $h((a,b,c),r)$ given by

$$h((a,b,c),r) = \begin{cases} (\max(a,b,c)+1, b, c) & \text{if } r = a \\ (a, \max(a,b,c)+1, c) & \text{if } r = b \\ (a, b, \max(a,b,c)+1) & \text{if } r = c \end{cases}$$

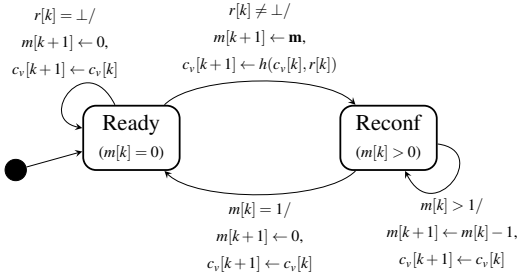


Control Device

- Control Device's logic: (continuing)

$$c_{t_j}[k+1] = \begin{cases} (\mathbf{f}, \mathbf{g}, \mathbf{m}) & \text{if } r[k] \neq \perp \text{ and } m[k] = 0 \\ & \quad \text{and } \max(c_v[k]) + 1 = j \\ \perp & \text{otherwise} \end{cases}$$

- State chart representing the Control Device's behavior



Selecting Modeling Framework

- We choose ForSyDe to model and simulate such architecture
- ForSyDe is implemented in Haskell
- Haskell benefits:
  - Functions are pure $\rightarrow$ no side effects
  - High order functions $\rightarrow$ possible to exchange functions as normal data
  - Strong type system
- ForSyDe has a synchronous library with process constructors
- A few process constructors: `comb`n`SY`, `unzip`m`SY` and `delaySY`

Architecture model in ForSyDe

- RTRP model in ForSyDe

```
rtrp (f0,g0,m0,x0) ct s_in x' = out
    where (out, fb) = unzipSY $ comb4SY rtrpFunc ct s_in x' fb'
          fb' = delaySY (f0,g0,m0,x0) fb
```

- Voter model in ForSyDe

```
voter cv s1 s2 s3 s4 s5 = unzip3SY $ comb2SY voterFunc cv (zip5SY s1 s2 s3 s4 s5)
```

- Control Device model in ForSyDe

```
ctrlDev r = (cv, cts)
    where (cv, m, ct) = unzip3SY $ comb3SY (ctrlDevLogic prosopon1) r m' cv'
          cts = unzip5SY ct
          m' = delaySY 0 m
          cv' = delaySY (1,2,3) cv
```

Architecture model in ForSyDe

- Architecture process network in ForSyDe

```
tmrPN s_in = (r, s_out, out2, out4)
    where out1 = rtrp1 ct1' s_in x'
          out2 = rtrp2 ct2' s_in x'
          out3 = rtrp3 ct3' s_in x'
          out4 = rtrp4 ct4' s_in x'
          out5 = rtrp5 ct5' s_in x'
          (s_out, x, r) = voter cv' out1 out2 out3 out4 out5
          x' = delaySY (Prst 0) x
          (cv, (ct1,ct2,ct3,ct4,ct5)) = ctrlDev r
          cv' = delaySY (1,2,3) cv
          ct1' = delaySY Abst ct1
          ct2' = delaySY Abst ct2
          ct3' = delaySY Abst ct3
          ct4' = delaySY Abst ct4
          ct5' = delaySY Abst ct5
```

Test Case

- RTRPs 1, 2 and 3 are configured with the accumulator function:

$$y[k+1] = y[k] + s_{in}[k] \qquad y[0] = 0$$

- $RTRP_2$ has a flaw:

$$y[k+1] = y[k] - s_{in}[k] \qquad \text{when } y[k] = 3$$

- RTRPs 4 and 5 are configured with the negative accumulator function:

$$y[k+1] = y[k] - s_{in}[k] \qquad y[0] = 0$$

- When $k = 4$, $RTRP_2$ will output the wrong result
- It takes 2 cycles to reconfigure a new RTRP

- **Expected result**:
  - RTRP$_2$ fails when $k = 4$
  - When $k = 7$, RTRP$_4$ assumes RTRP$_2$ task

Simulation results in ForSyDe

k	0	1	2	3	4	5	6	7	8	9
s_{in}	1	1	1	1	1	1	1	1	1	1
s_{out}	0	1	2	3	4	5	6	7	8	9
y_1	0	1	2	3	4	5	6	7	8	9
y_2	0	1	2	3	2	3	2	3	2	3
y_3	0	1	2	3	4	5	6	7	8	9
y_4	0	-1	-2	-3	-4	$\perp$	$\perp$	7	8	9
y_5	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
r	$\perp$	$\perp$	$\perp$	$\perp$	2	4	4	$\perp$	$\perp$	$\perp$

Conclusion

- An RTR architecture was presented for safety critical systems
- In the event of a hardware failure a new and safe module can be allocated to replace the faulted one
- TMR was used to detect and mask faults
- The system is able to withstand multiple permanent faults, provided no two faults occur in a small time window
- "Spare" processors can be initially used to perform non critical data processing
- The Voter and the Control Device are single points of failure of the architecture

Acknowledgments

- Research Development Foundation (*Fundação de Desenvolvimento da Pesquisa*) – FUNDEP/MCTIC/MDIC.
- Swedish-Brazilian Research and Innovation Centre – CISB

Thank you

- Corresponding author: Ricardo Bonna
- E-mail: rbonna@fem.unicamp.br