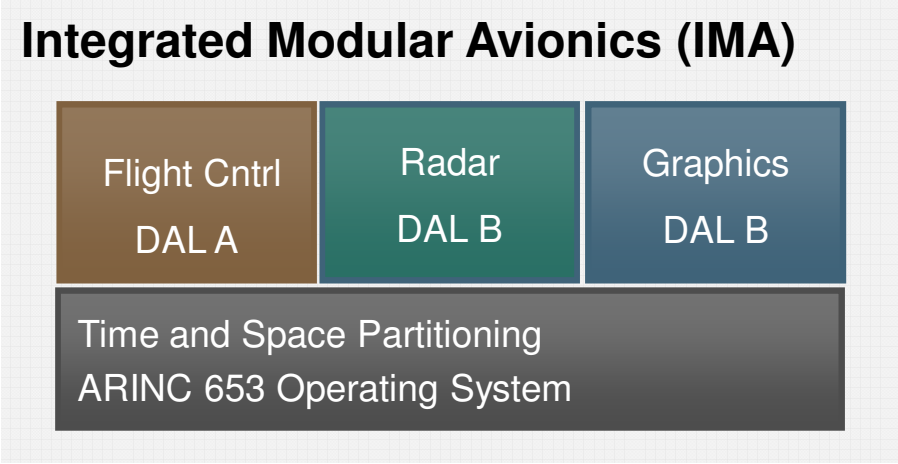
Using ARINC 653 Health Monitoring to Implement Robust Error Management in Avionics Platforms

Larry Kinnan
Principal Technologist for Avionics and Safety Critical Systems

What is ARINC 653?

- Industry specification for Integrated Modular Avionics (IMA)
- Includes an API specification and behavior definition of 56 routines
- ARINC 653 Operating Systems (O/S) and applications are typically certified to DO-178B

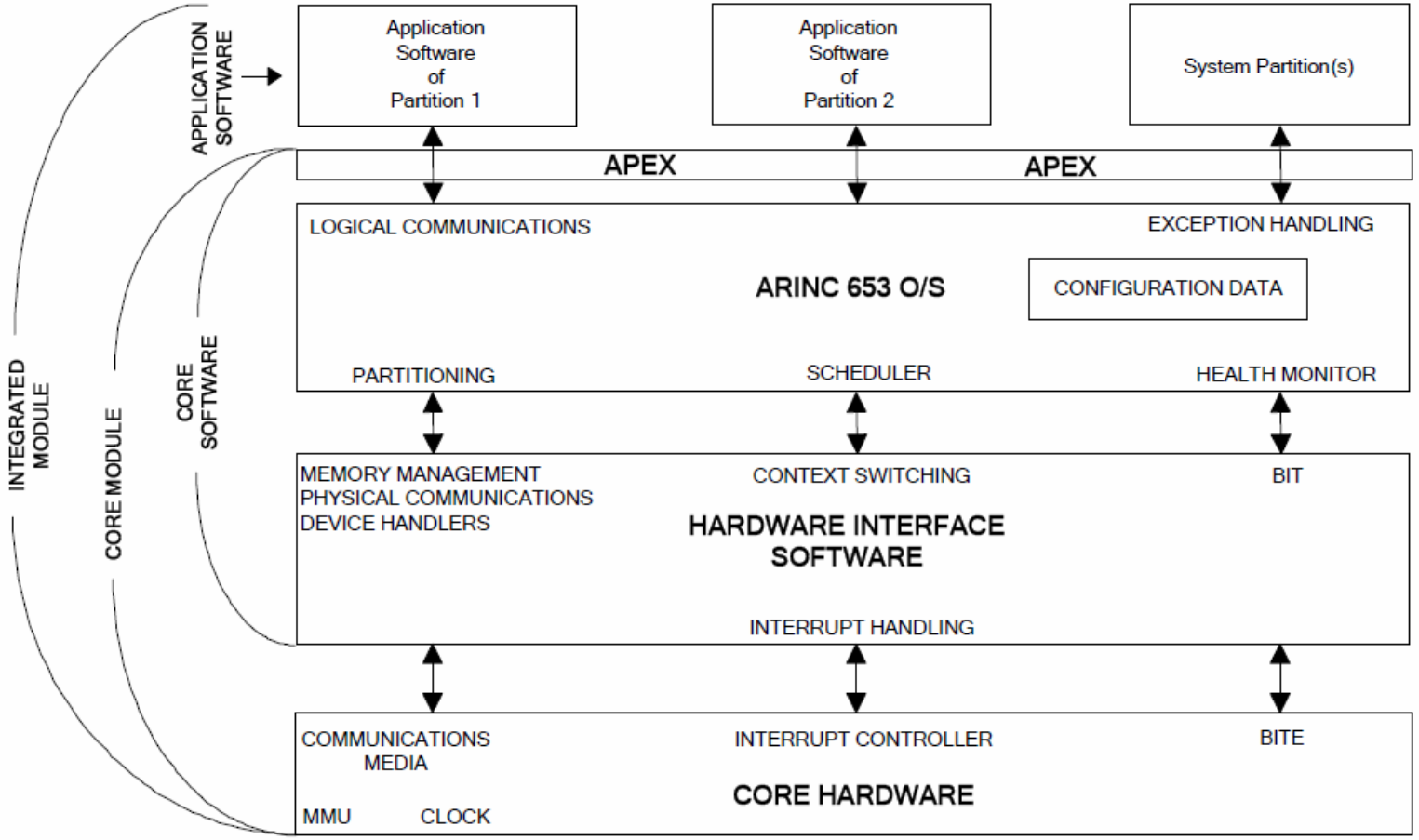


ARINC 653 APEX

- ARINC 653 specification defines a general purpose APEX (APplication / EXecutive) interface
  - Between the O/S and the application software
  - Abstracts away the underlying implementation
  - Allows applications at different safety levels to reside on a single processing element without interference through the use of time and space partitioning

- ARINC 653 is actually multiple specifications
  - Part 1, Supplement 3, Required Services
  - Part 2, Supplement 2 Extended Services
  - Part 3, Conformity Test Specification
  - Part 4, Minimal Profile
  - Additional Parts (0, 5) are underway to incorporate additional technical areas such as multicore.

ARINC 653 Notional Architecture



Source: ARINC Specification 653 Part 1, Supplement 3, Figure 2.1

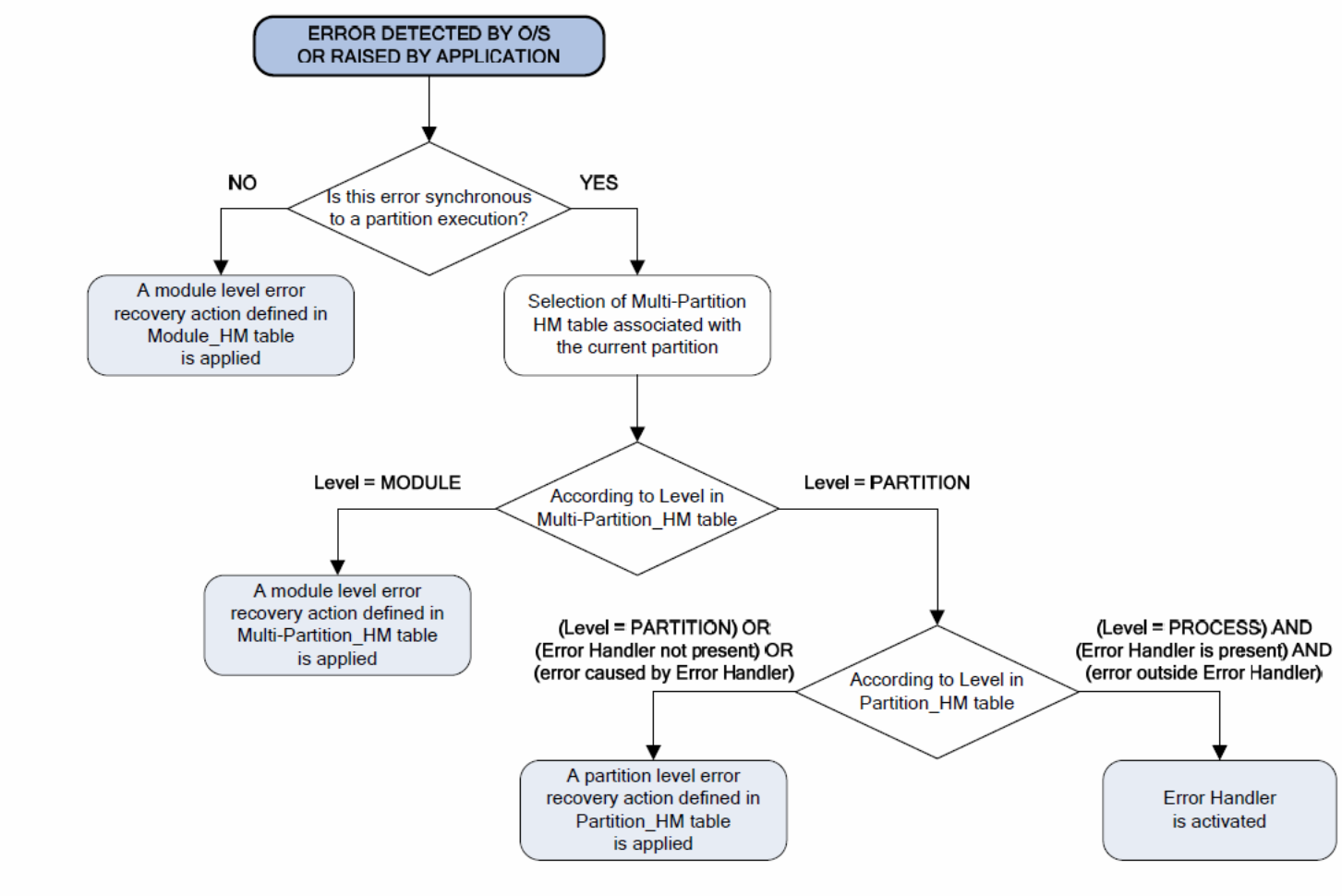
ARINC 653 APEX API

- ARINC 653 APEX API provides the following services:
  - Process management
  - Time management
  - Partition management
  - Sampling port management
  - Queuing port management
  - Buffer management
  - Blackboard management
  - Semaphore management
  - Event management
  - Error management
- Includes API specifications for C and Ada
- XML Schema defined to be used to define the configuration of the system
  - It does not cover the compositional elements of the system

Health Monitoring

- Provides a framework to raise and handle alarms in a system
- Functions at 3 levels in a hierarchal fashion
  - Process level: controlled by the Application Supplier(s)
  - Partition level: controlled by the System Integrator
  - Module level: controlled by the Platform Supplier
- Table-driven fault responses and recovery actions occur at partition and module level using XML to define an error code to handler relationship
  - Provides support for cold and warm restarts for partitions and module
- Application driven responses and recovery actions occur at process level in user mode

HM Decision Tree



Source: ARINC Specification 653 Part 1, Supplement 3, Figure 2.4

Process Level Errors

- A process level error impacts one or more processes in the partition, or the entire partition

- The HM will not violate partitioning when handling process level errors
  - All processing occurs within the partition's time window

- Fault responses to process level errors are determined by the application programmer using a error handler process.
  - The error handler process will be the highest priority process within the partition
  - The error handler process is active in **`NORMAL`** mode only

- The process level is optional. If not present, partition level error processing is invoked by the O/S

Partition Level Errors

- A partition level error impacts only one partition

- Examples of partition level errors are:
  - Partition configuration error during partition initialization
  - Partition initialization error
  - Errors that occur during process management
  - Errors that occur during error handler process

- The HM will not violate partitioning when handling partition level errors
  - Processing occurs within the partition's time window
  - This includes warm or cold restarts of the partition

Module Level Errors

- A module level error impacts all the partitions within the module

- Examples of module level errors are:
  - Module configuration error during module initialization
  - Other errors during module initialization
  - Errors during system-specific function execution
  - Errors during partition switching
  - Power fail

- Module level error processing ignores the temporal partitioning. Most module level errors are fatal and require immediate action including restart (warm or cold)

HM Error Codes

- **`DEADLINE_MISSED`**
  - Process deadline violation
- **`APPLICATION_ERROR`**
  - Error raised by an application process
- **`NUMERIC_ERROR`**
  - During process execution, error types of overflow, divide by zero, floating-point error
  - Numeric errors may be used by the Ada runtime in its operation
- **`ILLEGAL_REQUEST`**
  - Illegal O/S request by a process

HM Error Codes

- **`STACK_OVERFLOW`**
  - Process stack overflow
- **`MEMORY_VIOLATION`**
  - During process execution, error types of memory protection, supervisor privilege violation
- **`HARDWARE_FAULT`**
  - During process execution, error types of memory parity, I/O access error
- **`POWER_FAIL`**
  - Notification of power interruption

Invoking the HM

- The Health Monitor (HM) is invoked by
  - An application calling the **`RAISE_APPLICATION_ERROR`** service
  - By the O/S or hardware detecting a fault

- The recovery action is dependent on the error level
  - Recovery actions for each partition/module level error are specified in the Module HM and Partition HM tables (by the system integrator)
  - Recovery actions for process level errors are defined by the application programmer

Process Level Error Recovery Actions

- The application developer can optionally install an error handler process
  - CREATE_ERROR_HANDLER

- Possible recovery actions can be
  - Ignore, log the failure but take no action
  - Confirm the error (n times) before action recovery
  - Stop faulty process and re-initialize it from entry address
  - Stop faulty process and start another process
  - Stop faulty process (assume partition detects and recovers)
  - Restart the partition (cold or warm restart)
  - Stop the partition (**`IDLE`**)

Partition Level Error Recovery Actions

- Each partition can have its own HM configuration table or share a common one across multiple partitions

- Recovery actions are based on responding to specific error types and can be one of the following
  - Ignore
  - Stop the partition (IDLE)
  - Stop and restart the partition (cold or warm restart)

Module Level Error Recovery Actions

- The Module HM configuration specifies the recovery action based on the system state

- Recovery actions are based on responding to specific error types and can be one of the following
  - Ignore
  - Shutdown the module (fail)
  - Reset the module (cold or warm restart)
  - Recovery actions as defined by the implementation

Partition/Module HM Configuration

- System HM table
  - Defines the level of an error (Module, Partition, Process) based on the error and the state of the system
  - For process level errors only, the error codes are indicated in this table
  - These error codes are implementation independent
- Module HM table
  - Defines the recovery action (e.g., shut down the module, reset the module) associated with the detected module level error
- Partition HM table
  - Defines the recovery action (e.g., stop the partition, restart the partition in warm or cold mode) associated each detected partition level error
  - Each partition has a separate table or can share a common table

HM Services

- **`REPORT_APPLICATION_MESSAGE`**
  - If an error is detected, allows the current partition to transmit a message to the HM function
  - Can also be used to record an event ,for logging purposes

- **`CREATE_ERROR_HANDLER`**
  - For the current partition, creates an error handler process, that
    - Has no identifier (ID)
    - Cannot be accessed by the other processes of the partition
    - Is a special aperiodic process, with the highest priority
    - Cannot call blocking services
    - Transmits the error context to the HM function via the **`REPORT_APPLICATION_MESSAGE`** service

HM Services

- **`GET_ERROR_STATUS`**
  - Used by the error handler process to determine the error code, the identifier of the faulty process, the address at which the error occurs, and the message associated with the fault

- **`RAISE_APPLICATION_ERROR`**
  - Allows the current partition to invoke the error handler process for the specific error code **`APPLICATION_ERROR`**

Process Level Error Handler Example

```
void processLevelHandler (int param)
{
   ERROR_STATUS_TYPE errorStatus;
   RETURN_CODE_TYPE  returnCode;

   while (1)
   {
        GET_ERROR_STATUS (&errorStatus,&returnCode);
        if (returnCode == NO_ERROR)
         {
           switch (errorStatus.ERROR_CODE)
             {
             case DEADLINE_MISSED :
                logMsg ("ARINC DEADLINE_MISSED has occurred\n\r", 1, 2, 3,4,5,6);
                break;
             case APPLICATION_ERROR :
                logMsg ("ARINC APPLICATION_ERROR has occurred\n\r", 1, 2, 3,4,5,6);
                break;
             default :
               /* this error could be from the partition OS so the partition may need to be restarted    */
                break;
             }
         }
        else
        {
         /* Failure of GET_ERROR_STATUS indicates  a possible OS problem */
         /* possible actions are log error or restart the partition                    */
         break;
         }
    }

    STOP_SELF ();       /* the error handler process stops itself */
}
```

All possible ARINC errors should be handled here. Only 2 shown for brevity

Error handler only runs in response to a HM error being injected by the O/S, the application or a hardware induced fault

Partition HM Table Example

```
<PartitionHMTable Name="part1Hm">
    <SystemState>
        <ErrorIDAction
            ErrorIdentifier="UNKNOWN"
            ErrorAction=“IDLE"/>
        <ErrorIDAction
            ErrorIdentifier="NUMERIC_ERROR"
            ErrorAction=“IGNORE"/>
        <ErrorIDAction
            ErrorIdentifier="POWER_FAIL"
            ErrorAction=“SHUTDOWN"/>
        <ErrorIDAction
            ErrorIdentifier="HME_KERNEL"
            ErrorAction="KERNEL_ERROR"/>
        <ErrorIDAction
            ErrorIdentifier="HME_CONFIG_ERROR"
            ErrorAction=“RESTART"/>
    </SystemState>
</PartitionHMTable>
```

Ignore Error which is typically not wise to do as the results may be unpredictable

The Partition HM Table can be extremely long depending on the requirements for error handling or it could be one entry that restarts the module.

ARINC 653 Adoption and Future

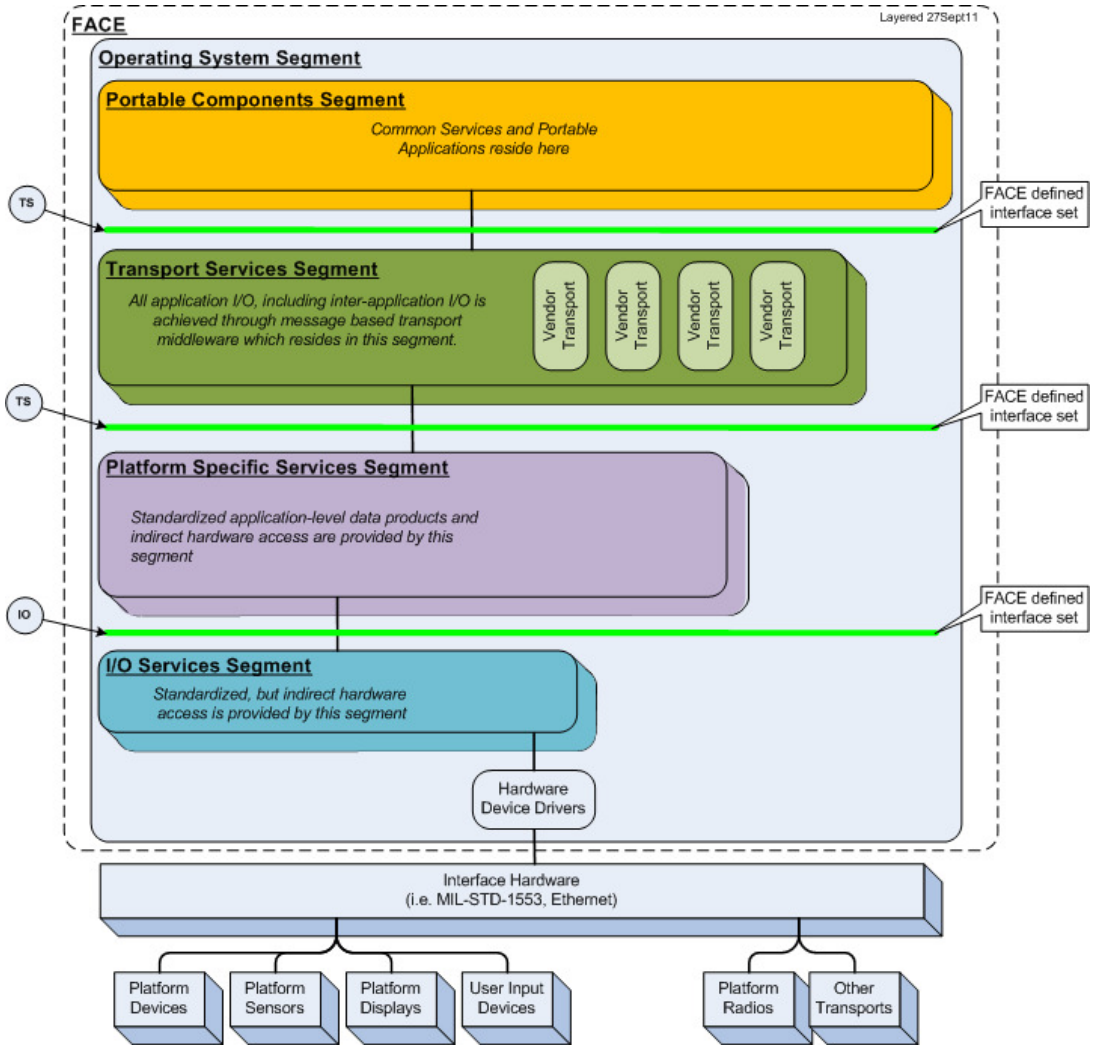
- ARINC 653 has been used successfully in numerous commercial and military aircraft
  - Airbus 330 MRTT ARBS
  - Boeing 787 Common Core System
  - Boeing P8A MMA
  - C130 AMP (Tech Refresh)
  - Boeing KC-767
  - nEUROn UCAV Demonstrator
  - Northrop X-47B UCAS Flight Control and Mission Computer

- ARINC 653 has a proven track record and is now being adopted into other emerging standards

FACE - Purpose

- Delivers an open architecture that enables rapid deployment and re-use of software across platforms
  - Includes both a technical specification and a business model
  - Includes 4 O/S segment Profiles which use ARINC 653 Part 1 or POSIX
- Enables more capability sooner, on more platforms
  - Expands software supplier choices and enables interoperability
  - Creates a platform for integrating both future and legacy systems
- Provides an industry library of conformant software and supporting safety / security evidence to accelerate usage

FACE Architectural Segments



- FACE Portable Components Segment
  - Portable Applications
  - Portable Common Services
- Transport Services Segment
- Platform Specific Services Segment
  - Platform Device Services
  - Platform Common Services
  - Graphics Services
- I/O Services Segment
- Drivers
- Operating System Segment
  - 4 Profiles (General, Safety Basic, Safety Extended and Secure)
    - ARINC 653 APIs
    - POSIX APIs

Thank You

Author contact information:

Larry Kinnan

Principal Technologist for Avionics and Safety Critical Systems

larry.kinnan@windriver.com

Definitions

- AMP – Avionics Modernization Program
- ARBS – Aerial Refueling Boom System
- APEX - APplication EXecutive
- DAL - Design Assurance Level
- FACE – Future Airborne Capability Environment
- HM – Health Monitoring/Management
- IMA - Integrated Modular Avionics
- O/S - Operating System
- UCAV - Unmanned Combat Air Vehicle
- UCAS - Unmanned Combat Air System

