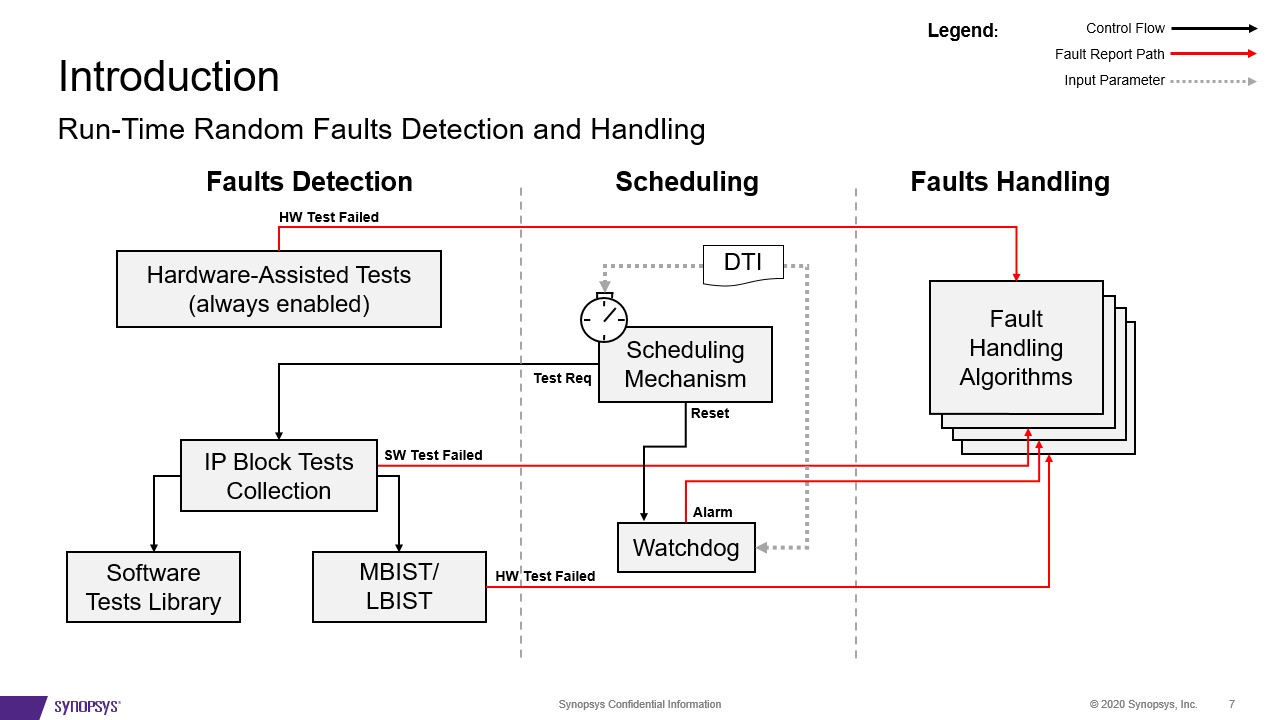
**Safety management Library**

* **FTTI** Fault tolerant time interval: The minimum time spent from the occurrence of fault of an item to the possible occurrence of hazards if the safety mechanism is not activated so you must reach the safe state before this interval
* It also defines diagnostic time interval -> fault reaction time -> safe state and all this before the hazard is even possible
* Fault tolerant time interval may be a function of speed and rounded environment it depends on the block of the design
* Diagram of runt time faults detection

Regarding the HW assisted test this test is for monitoring of voltage and clock

Fault handling algorithms maybe to reset the system or send a report if it’s a watch dog alarm or

To restart a hardware block when its LBIST report etc..



* We are not providing a portable code for any other processors like arm
* Safety management library runs on a master processor and may have slave processor
* For safety master sw runs the fault handling algorithms
* Test manager: optional group of periodic unit tests + Test management has apis to execute a test or group of tests those tests are group of 3

**pre-mission(post Boot-time)** tests and **periodic** optional tests and **pre-shutdown** tests are before the shut down if the system was capable of continuing the execution while having a fault this test will double check and send reports regarding this software crash (send diagnostic data)

* Safety slave must have local software test like stl asil B related to that core or implement software lockstep while the master having the software manager should manage the whole system
* Testing have two parts build time and runtime configuration so user must provide the file with constant data structures of modules for initialization of different components
* To configure manager:

1. User define test procedures which are C function with certain interface define testing sequence like built in memory self-test
2. Assign ID for each test procedure
3. Divide all test procedure into groups using **test** registry (processor specific) which is a constant table that contain properties of test procedures like ID, group at, pointer to C function implementing this test procedure

**Test procedure**: const uint32\_t **(\*test\_procedure)**(​

       uint32\_t **seed**, ​

       const uint32\_t **param**, ​

       uint32\_t\* **signature**​

)

Seed: It’s a built in mechanism to control execution flow to make sure it run completely

Sig: check sum assert whole code is called and no for example a stack corruption

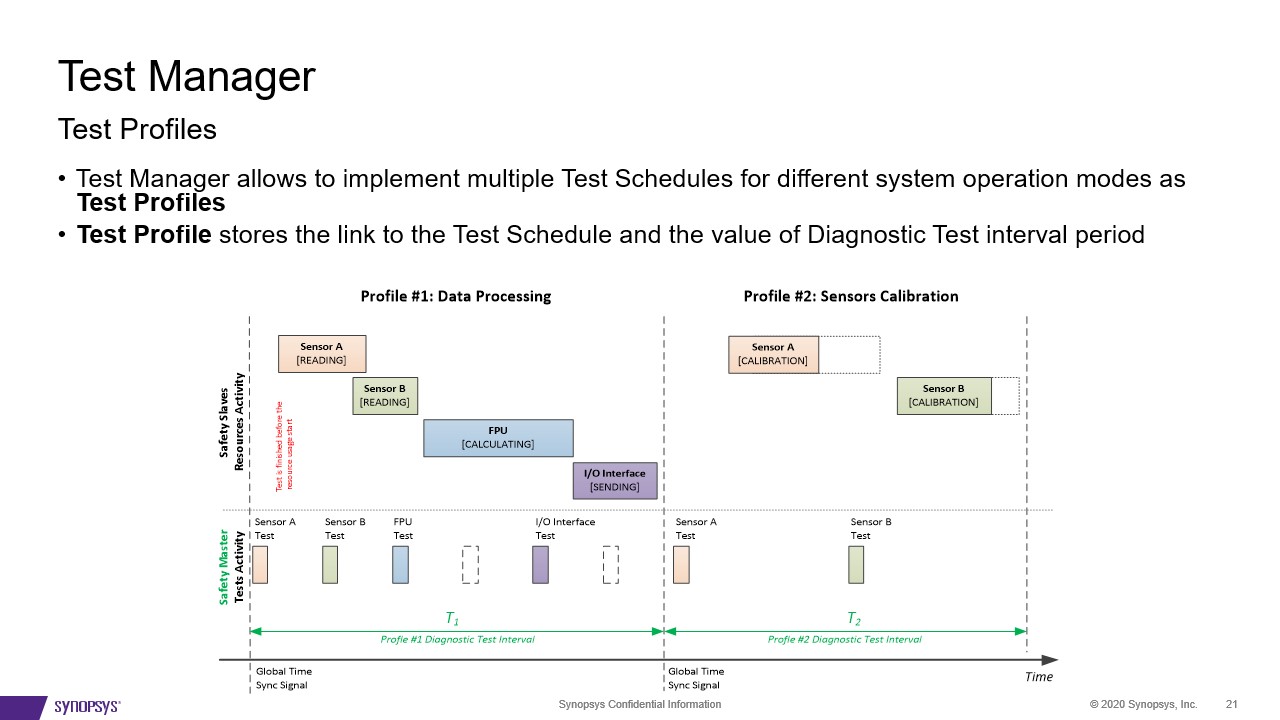
1. If it has run time test (periodic test) user must define **test scheduling table (generalized for master and slaves)**

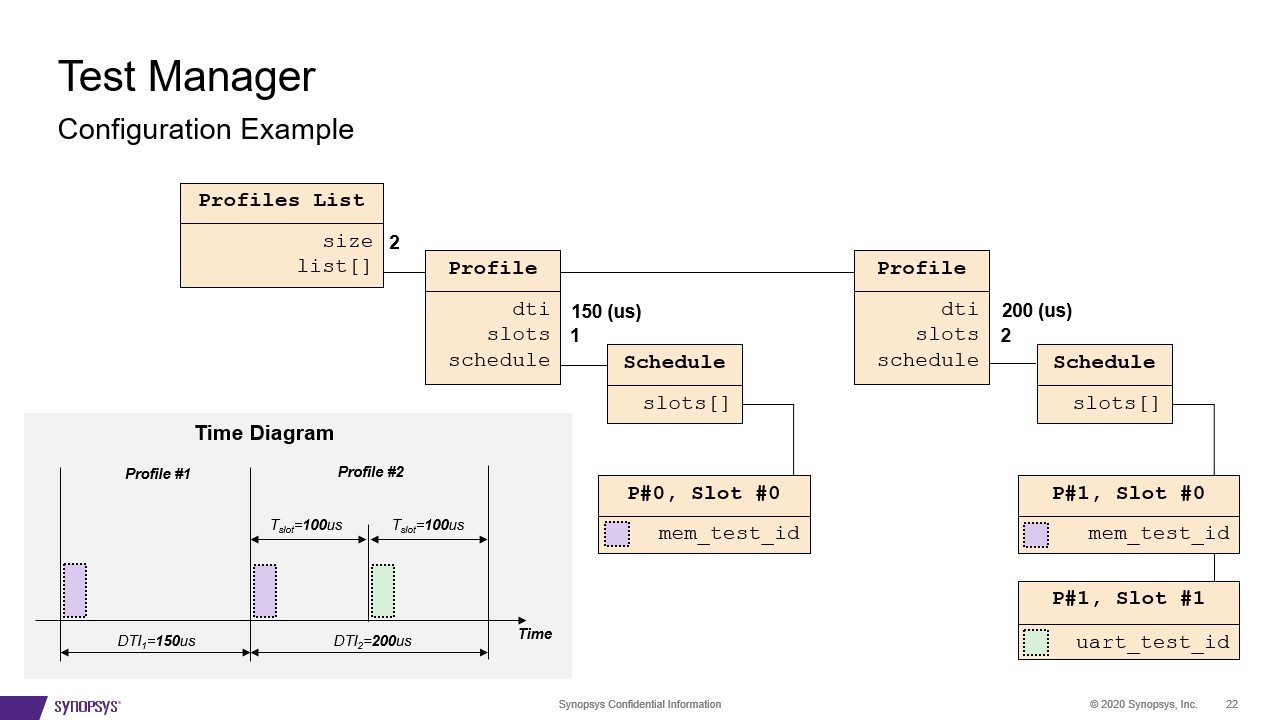
**Test Schedule :** It’s a constant C structure it consists of slots(small subset of tests) it may be not applicable to interrupt the system for considerable amount of time(100+test) so this increases the interrupt times It takes shorter slots of time each time so those decisions are taken by system designer and hard-coded in the schedule Note : Those whole slots have to run within **DTI (Diagnostic Test Interval)**

**Explanation:**

Imagine having a serial port sending safety critical data over a channel and communicate with some sensor and we need periodic tests so safety master control this test it has some diagnostic time interval and it stops the function of the driver and allocates the serial port for the testing for the loop back test and then reconfigure the resource back if the testing passes We need the scheduling to wait or check if the resource is free

----------------------------------------------------------------------------------------------

1. **Test profiles**



1. **Fault Manager**

* Types of faults: Software initiated; Hardware initiated
* Fault manager injects the fault in internal queues so that test manager can report and error
* Fault manager also deals with HW initiated faults that are signaled using interrupt controller of ARC core, it implements the generic IRQ handler to catch the faults

1. **Fault reaction**

* It’s a c function provided by system designer
* Should implement specific fault handling like problematic block recovery and start the system shutdown process
* Its executed within the fault reaction time interval
* Its outcome is either reporting a problem to switch to safe state or error correction with staying in mission mode

1. **Alarms and Fault aggregation**

* Faults don’t have to be always taking an action immediately
* There are some types of alarms:

**The pass through:** default type calls the reaction immediately after receiving the fault

**Join:** allows to configure a reaction in case of getting several faults during the preconfigured period

**Rate:** Calculates the rate of a fault signals and invokes a reaction after passing a certain threshold

1. **Fault list**

Is a C structure that describes the list of faults that should be handled on the core it provides the following

* Unique fault ID
* Fault source
* Priority and severity