A Review of Watchdog Architectures and their Application to Cubesats

Jacob Beningo
AERO590 Winter 2010
Department of Aerospace
University of Michigan
Ann Arbor, Michigan 48109

April 28, 2010

1 Introduction

The push towards "faster, better, cheaper" missions with an emphasis on design reuse has led to the popularity of COTS products that have traditionally relied upon custom software and hardware [22]. The motivation for moving towards COTS hardware is faster, cheaper and more powerful processors with lower power consumption compared to those available in rad hard versions [21]. While moving towards COTS components has many advantages, it also requires developing mechanisms to protect the components from radiation harsh environments which high altitude and space systems are subject to.

There are two primary methods to deal with the effects of radiation on an embedded system. The first, fault avoidance, can be defined as using more robust or radiation hardened components to avoid single event upsets caused by cosmic rays [17] and other high energy particles. While this approach ensures system reliability, it does carry with it several disadvantages. Primarily, fault avoidance increases system power usage, costs and lead times while decreasing computational power. Power requirements cause a trade off between higher voltage components and faster system clocks. Power consumption rises as the components voltage increases and as the system clock increases in frequency. When components are selected that are not in high demand by markets such as the consumer electronics industry, the cost of the components rise due to a lack of demand which also has the effect of increasing the lead time of the components.

The alternative method to fault avoidance is fault detection. Even the most robust fault avoidance systems will eventually have a fault [17]. While fault avoidance systems aim to increase the mean time of failure to beyond the mission life, fault detection methods detect when the fault occurs [17] in order to handle the fault and limit the system downtime. There are a number of advantages to fault tolerant systems such as low power consumption, increased computational power and decreased hardware costs. However these advantages can lead to increased hardware complexity (in order to protect the components) and increased software complexity depending on the type of fault tolerant architectures used.

When developing a system based on the fault detection method, one of the most cost effective methods of detecting and handling faults is the watchdog. A watchdog is a subsystem which monitors the operation of the system and in the event a fault or an unknown state occurs, the watchdog can restart the system or put the system into a known state from which the system can recover. In this paper, we will examine common watchdog architectures which have been used in terrestrial and space systems and develop a systematic approach to selecting the architecture which is most effective and developing a general strategy on how they can be used to improve system reliability in cubesats. We will then examine how this approach was used to develop a fault detection method for the Radio Aurora Explorer (RAX) nanosatellite.

2 System Faults

Every embedded system, whether it is here on earth (in automobiles, cell phones and refrigerators) or out in space (in satellites, rovers and space probes) is vulnerable to the adverse affects of radiation. On Earth, these radiation sources are produced by alpha particles from radioactive material, secondary particles from cosmic rays and thermal neutrons [24]. In space, radiation sources include radiation belts around the Earth (Van Allen Belts), solar winds, and cosmic rays [24][19].

Radiation poses a significant threat to an embedded systems operation. Radiation is responsible for two types of failures; Permanent (hard faults), Transient (also known as soft faults or single event upsets [SEU]) [16]. Permanent faults occur when radiation permanently damages a hardware component such as a transistor. This can result in the state being permanently on, off or somewhere in between. Transients occur when radiation temporarily causes a bit flip to occur within the hardware or a communication signal. Transients can be handled by refreshing the memory or restarting the system. Transient faults are the faults that need to be monitored for and corrected. When a transient fault is detected in a system, there is a typical set of steps which should be followed in order to respond to the fault and handle it dependably [22]:

- Fault detection - recognizes that something unexpected has occurred in the system.
- Fault confinement (containment) - limits the spread of the fault effects to one area of the system through the use of fault-detection circuits.
- Diagnosis - if the fault detection technique does not provide information about the failure location and/or properties the cause of the fault will need to be determined.
- Reconfiguration - When a fault is detected and it is determined to be a permanent failure the system might be able to reconfigure its components either to replace the failed component or to isolate it from the rest of the system.

- Recovery - Utilizes techniques to eliminate the effects of faults through fault masking, retry, reseting or power cycling.

2.1 Hard Faults

Hard faults are the result of permanent damage to the system. There are many potential causes for this type of system failure [13]. A few examples are transistor breakdown due to excessive heat or a crystal failure on an integrated circuit. In the event of a hard fault, the only method to recover the system is through part replacement or system redundancy. For space systems, short of being the Hubble or the International Space Station, part replacement is typically not an option. System redundancy however is a viable solution and will be discussed later in this paper.

The work done in [19] has identified several different types of hardware faults which are listed below:

- Latch-up. Permanent, potentially destructive
- Heavy Ion Induced Burn-out in power MOS
- Single Event Gate Rupture (SEGR)

These hardware faults not only cause system faults but can result in lost functionality of the system or permanent damage to other electrical subsystems, in particular the transistors making up the integrated circuits. For example, when radiation passes through the system, if enough charge is injected into the base of the transistor, the normally high-impedance state can become low and turn the transistor on [18].

2.2 Soft Faults

The three primary causes of soft faults are software bugs, electromagnetic interference (EMI) and transients. Software bugs are the result of the embedded firmware behaving in an unexpected manner. This could be caused by attempting to allocate memory from the stack when it is full or some other unexpected behavior which did not appear when the system was being tested. Software bugs can also be removed from the system through proper software testing and validation procedures which will not be covered in this paper.

Electromagnetic interference has become an all too familiar cause of software faults due primarily to increases in system frequency and the decreasing pitch of pins on components [13]. These advances in technology have left serial interfaces vulnerable to noise which results in incorrectly reading or writing of data [13].

Transients occur when a radiation event causes a charge disturbance that flips the logical state of a transistor [10][18]. Transients are generally temporary and can be cleared by writing to the affected register or power cycling the system [23]. If left unchecked, a transient has the potential to affect critical operations of the spacecraft and cause permanent damage [18]. This is where fault tolerant hardware such as a watchdog can save the day. The main contributors of transients are cosmic rays which bombard the earth constantly and exhibit the following four characteristics [10]:

1. Cosmic Ray intensity increases with altitude
2. Fail rate is scaled directly with cosmic ray intensity
3. As devices become smaller they are more vulnerable to soft faults
4. Transient vulnerability is independent of its individual dimensions, or mounting orientation

3 The Philosophy of Watchdogs

Soft faults are not permanent faults and therefore can be detected in a variety of ways. Soft faults often reveal themselves through flow and data faults. A flow fault occurs when a microprocessor is executing code and then begins executing code out of order. One way this can occur is if a bit flips and causes the instruction that is being executed to be interpreted wrong. Data faults occur when the processor is writing data to ram or drive and writes a value such as 0x55 but instead a bit gets flipped and it becomes another value.

Soft faults can be detected through the use of system monitors and general fault detection schemes such as a watchdog. A watchdog is a system which is simpler and less complex than the system that it is monitoring and can exist as either software or hardware. The watchdog monitors for faults which range from a simple time period monitor to a program flow and data validity checker. When a fault is detected, the watchdog can determine what action can be taken in order to return the system to a known state. Instead of the system running a muck, the watchdog is able to avert the fault and by doing so increases the robustness of the system.

In order for a watchdog to have the most effect on a system, there are a number of characteristics which have been identified in [11] which a good watchdog will have as identified below:

1. Watchdog shall detect all erratic software modes - if the software is able to enter into a software state that is not valid and the watchdog is unable to detect it, then the watchdog is not effective.
2. Watchdog shall put the system back to a known"safe" state no matter what went wrong - Returning to a "safe" state allows the system to reinitialize itself and begin executing code in the proper order.
3. Watchdog shall be independent of the primary system - If the primary system fails and the watchdog is part of that system then the watchdog will also fail and not be able to return the system to a known state.

4. Watchdog shall be capable of providing a hardware reset for all peripherals - Resetting all of the peripherals gets all devices in the system back to a known state.

5. Watchdog shall be woven into the entire embedded system - The watchdog should not be dependent upon just a single bit or function but instead each task within the system should set a flag indicating that the process was successful. In this way if a single task freezes but the rest of the system is still functioning, the watchdog will be able to detect that the task has frozen and then go about putting the system back into a known state.

6. Watchdog shall issue a hardware reset on timeout - When the watchdog times out it determines that something has gone wrong with the system that it is monitoring and should therefore begin by providing the processor with a hardware reset.

7. Watchdog shall be more than a pure software solution (Software Watchdogs are not reliable due to the fact that they can be turned on and off and modified during runtime)

8. Watchdog shall not make assumptions about the state of the hardware or software -

9. Watchdog shall generate debugging information - The watchdog should keep track of what caused it to put the system into a known states. For a software watchdog this could be setting a non-maskable interrupt flag or for hardware it could be relaying a message to an external system that an issue has occurred.

10. Watchdog shall have its own clock, and not share any clock with the system - If the primary system clock freezes or fails in some way and the watchdog is dependent on that clock in order to reset the system, the watchdog will also fail and be unable to return the system to a known operating state.

3.1 Inputs

The primary purpose of a watchdog is to monitor the system for faults and put the system back into a known and normal operating state. In order to do this, the watchdog requires information about the status of the software and hardware of the system. While there are a variety of signals and system parameters that a watchdog can monitor, they can be categorized into three general signal groups: heartbeats [8], bus monitors [2][3][9] and control flow.

The simplest of the three signal groups is the heartbeat. A heartbeat is an "I'm Alive" signal generated by the processor(s) being monitored. It is typically a periodic signal that is generated when the processor reaches a certain known state within the software. It signals the watchdog that the system is functioning and that the code has not hung up. The watchdog will usually have a tolerance for the period at which it expects to receive the heartbeat. If the heartbeat does not occur by the time the watchdog is expecting it to, the watchdog will then take actions to reset the processor or system to put it back into a known state. Figure 1 shows an example of a typical heartbeat signal.

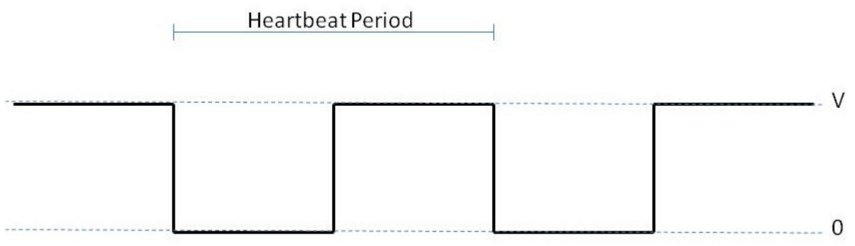


Figure 1: Heartbeat Signal

Within a microprocessor there are a number of ways in which the heartbeat signal can be generated and a limited number of ways in which it can be done correctly and effectively. The first method to generate the heartbeat is through a hardware timer. The timer is setup to toggle an output pin at the frequency required to reset the watchdog. The problem with this approach is that once the heartbeat is setup, the software requires no additional interaction in order to generate the heartbeat. The software could be stuck in an infinite loop but the hardware timer would continue and the watchdog would be unaware that the system was in distress.

The second method to generate the heartbeat is through a function call in the main() of the program. In this method the software generates the heartbeat on each pass through the main loop. While this approach is much better than the first method, there are still a few potential issues. If the processor jumps off and begins executing code out of order or a task freezes and is no longer running, it would still be possible that the heartbeat could be generated. The code could get stuck in the heartbeat function and continually only generate the heartbeat.

In order to prevent this type of failure, the third method which can be used to generate the heartbeat uses a flagging system. Each task within the system is given a flag which it sets when it has successfully completed running. When the heartbeat tasks runs it verifies that each task flag has been set before it generates the heartbeat. At the end of the heartbeat task, the task flags are cleared and each task must once again set its flag in order to generate the heartbeat. By doing this, if the code crashes or a task fails, the heartbeat will not be generated and the watchdog will know that a fault has occurred.

A more complex watchdog will not only monitor the systems heartbeat but also monitor bus activity. Bus monitors are bus input signals that allow the watchdog to monitor any system bus from communications buses to memory buses. The watchdog can monitor for specific fault occurrences on the bus such as checksum errors or commands which require the watchdog to take action. With the addition of bus monitoring, the system and the watchdog begin to become more vulnerable and introduce potential failure modes. For example, the added complexity of bus monitoring requires that the watchdog itself be a processor. If the watchdog is using a general purpose input output pin as an input to monitor the bus, if a fault within the watchdog occurs, the watchdog input pin could be changed to an output and instead of passively watching the bus it could actively write to it. If this were to occur the watchdog could prevent the system from operating properly.

The most complex type of watchdog will also monitor signals known as control flow signals. Control flow signals are sent to the watchdog so that the watchdog knows which process is being executed on the processor. The watchdog monitors that the process is executed properly in addition to verifying that the next process in the processor is executed at the right time and proper sequence. This is typically done by sending a signature to the watchdog. The watchdog recognizes the signature and monitors the activity for faults. If tasks are ran in an inconsistent or improper order the watchdog takes corrective action.

3.2 Outputs

In the event that the system watchdog detects a fault, the watchdog is required to put the system back into a known system state from which task code can begin execution to recover the system. A watchdog has a typical set of tools at its disposal to enter a known state. These include setting an alert, resetting the processor, power cycling the system, and reconfiguring the system.

There are times when a watchdog may detect a fault such as a bus communication fault or a missed heartbeat but the fault is not critical enough at that time to warrant a processor or system reset. In the event of an error that is not critical, it may be useful to simply log or provide an alert that a fault has occurred. This signal can then be used to notify the processor or an operator that a deadline was missed and that system interaction is necessary.

Nearly every watchdog system will have the capable to reset the primary system processor. Resetting the processor restarts code execution from the reset vector of the program. This places the processor into a known startup state from which the watchdog can then monitor for other faults. One of the disadvantages of a processor restart is that peripherals and other subsystems will not necessarily be restarted. If the processor hung while waiting for another subsystem to respond, that subsystem will not have been restarted and the system will eventually reach a fault state again.

It is therefore important that the watchdog have the capability to power cycle the entire system. This will guarantee that the entire system is reinitialized. A standard system will attempt to recover by first resetting the processor and if this fails a determined number of times then the watchdog will power cycle to recover the system. This will bring every peripheral and processor up in a known state and allow the primary processor to reinitialize the entire system.

In the event that a critical system error is detected, an advanced watchdog and system architecture can be capable of reconfiguring the system to work around the detected fault. This would typically be performed by a watchdog that is part of a system that uses subsystem redundancy. When a subsystem or processor fails, the watchdog goes through a procedure (such as resetting the processor and power cycling the system) to try to recover the subsystem. If recovery fails, the watchdog removes the faulty subsystem from the bus and re-tasks processors or redundant hardware to take over the functions of the faulty subsystem.

4 Watchdog Architectures

Watchdog architectures fall into two different classifications, internal and external watchdogs. An internal watchdog is on-chip, typically implemented as a hardware counter. The hardware counter is periodically reset based on conditions occurring within the software. An external watchdog is an off-chip monitoring system which can consist of a simple timer, a system basis chip, or a network of redundant self monitoring nodes.

4.1 Internal Watchdogs

The simplest type of watchdog is a software watchdog. This type of watchdog is included in nearly every, if not every modern processor as can be seen in the block diagram of a simple 8 bit MSP430F2011 in Figure 2. It consists of a timer interrupt service request (ISR) and a refresh function [6]. The timer interrupt is set to run when the watchdog counter reaches a predefined count. Once the count is reached, the ISR runs and restarts the processor. The purpose of the refresh function is to clear the counter before it reaches the predetermined count and runs the ISR. The refresh counter occurs at a regular interval and if it is tripped, signals that the processor has hung up or not executing code along its predetermined timing requirements. Most processors during boot will have a non-maskable interrupt flag which will be set and allow the processor to identify the watchdog as the source of the reset.

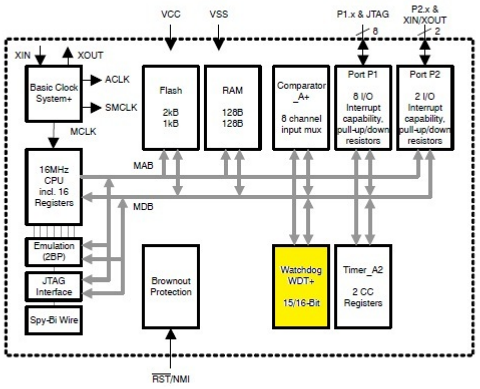


Figure 2: Internal Watchdog Hardware

One of the advantages of an internal watchdog is that it reduces hardware complexity and cost [6]; however, the implementation of only an internal watchdog is not a robust solution. There are two primary issues with using only an internal watchdog. The first, is that internal watchdogs can be turned on, off and period modified during runtime. If the processor has a fault and begins running code that turns off the watchdog, there will be no way to recover the system. A number of processors have to tried to work around this by including very specific timing parameters to turn off the watchdog such as writing to the register two or three times in a specified number of clock cycles but the risk is still there that the watchdog could be turned off. The second issue with only using an internal watchdog is if the watchdogs clock stops functioning. If the watchdog is not getting clock cycles then it is unable to run it's code or hardware to monitor for faults.

There are many different types of software implementations of software watchdogs which can be categorized as follows [6]:

1. Scheduler watchdog - ensures the scheduler or primary program loop is functioning properly.

2. Program Flow watchdog - monitors that the program is being executed in the proper sequence.

3. Oscillator watchdog - detects an oscillator failure on the processor

There are a number of issues which need to be considered when using a software watchdog. The following watchdog features should be considered carefully when working with internal watchdogs [11]:

- Watchdogs which are capable of being modified during run-time can be disabled by crashed code and therefore will never reset the processor.

- Watchdogs must assert a hardware reset to guarantee the processor is restarted.

- Reloading only the program counter may not properly reinitialize the processor

- If a hardware reset is not asserted, every peripheral will have to be reset manually

- A single I/O line is not safe due to a simple toggle or bit flip which can reset the timer

4.2 External Watchdogs

The last general class type of watchdogs is the external watchdog. This type of watchdog is a separate piece of hardware which is independent of the processor. An external watchdog can monitor a single processor or it can be a system monitor and monitor multiple processors and subsystems. Either way, there are a number of advantages to using an external watchdog:

1. Hard reset of the processor

2. More robust system

3. Watchdog is not dependent upon the processors oscillator

4. It is separate from the process which it is monitoring

While these advantages are very important, there are a few drawbacks to using an external watchdog:

1. Cost

2. Added system hardware complexity

Neither of the disadvantages listed here should defer one from using an external watchdog. There are a plethora of different types and architectures of external watchdogs which range from simple monostable vibrators to complex redundant multi-watchdog nodes where every node on the bus monitors every other node. The cost to add a simple monostable vibrator as the external watchdog costs less than $1 and requires very little added complexity or board space. In this section we will look at each type of external watchdog in detail and provide some guidelines on how they are typically implemented.

4.2.1 Watchdog Timers

The watchdog timer is the most simplistic and cost effective of the external watchdogs. The watchdog timer is an external counter that is set to toggle the reset line of the micro-controller when it reaches a preset period. In order to prevent the reset, the processor must reset the watchdog timer before the expiration of the counter. Failure to reset the timer is seen as an indication that the processor has hung up and cannot be recovered [1]. An example interface for a watchdog timer can be found in Figure 3.

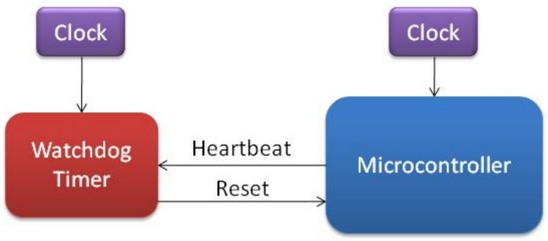


Figure 3: Typical Watchdog Timer Interface

The watchdog timer is typically a simple timer circuit, often a monostable vibrator [4] such as a 555 timer [5]. It provides the same functionality as the internal watchdog except that it provides a more robust solution by providing a redundant hardware timer that is separate from the hardware it is monitoring. In general the watchdog timer monitors the period at which a particular task or event is occurring and if the period is longer than that for which the watchdog is set, the hardware is reset.

4.2.2 Windowed Watchdog Timer

One of the primary disadvantages of a standard watchdog timer is that if the system reaches a fault state in which it continually resets the watchdog timer, the error state will not be detected. The standard watchdog timer can detect a slow fault but will not detect a fast fault which occurs in a period less than that of the watchdog timer [10]. To solve this problem, the windowed watchdog was introduced.

A windowed watchdog consists of two timers. The watchdog refresh must occur after timer 1 expires but before timer 2 expires [4]. In essence, creating a window in which the watchdog must be reset. If a reset occurs before timer 1 or after timer 2, the watchdog will reset the processor. The windowed watchdog therefore overcomes the issue of transient failures that reset the timer early [4]. The timing for a windowed watchdog can be found in Figure 4. By detecting early resets, the windowed watchdog improves the reliability of the system; However, if a fault occurs within the window, the watchdog will not be able to detect it[10]. Therefore the closer the period of timer 1 and timer 2, the less likely it is a fault will occur within the window[4].

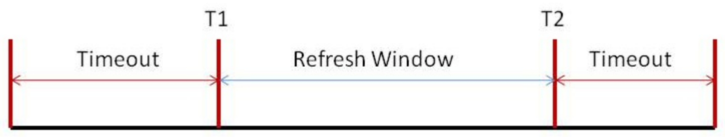


Figure 4: Window Watchdog Timing

4.2.3 Watchdog Processors

A watchdog processor is a small, simple co-processor that detects errors by monitoring the behavior of the system [2][3]. A watchdog processor can monitor a heartbeat, a system bus, shared memory or a combination of these system features. In order to do this, the watchdog processor is no longer a simple piece of hardware but is a micro-controller which also requires software. This inherently increases the vulnerability and risk of the system due to the fact that the watchdog is now susceptible to the same transient faults which it is designed to detect in another processor.

While there is an added risk to using a watchdog processor, it also provides the most flexibility and robustness (if implemented properly) than any other watchdog. It does not need to be overly complex and consume large portions of board space. The watchdog processor can be a simple 8-bit micro-controller from a part family such as the TI MSP430 or the Microchip PIC18FXXXX series. These parts are tiny and consume very little board space. In order to overcome possible faults with the watchdog processor, a simple watchdog timer can be added to it in order protect it from transient faults; thereby increasing it's robustness to properly monitor the system for faults.

A watchdog processor is more complex and costly than using a simple timer circuit, but it provides the ability of the watchdog to monitor complex operations being performed by the system which would normally go unchecked and uncorrected if the operation failed. If properly implemented, it can provide a very robust solution for fault detection and recovery.

4.2.4 Peer Watching

Complex embedded systems are often subdivided into subsystems that each perform a specific task or function. In this type of system, where each subsystem contains a processor performing its own tasks, it is possible to have each subsystem act as a watchdog and monitor other subsystems. This type of watchdog monitoring is known as peer watching. There are three types of peer watching; linear, circular and spoke.

Linear peer watching is when every subsystem except for one is monitoring another system. No subsystem monitors more than a single subsystem as can be seen in Figure 5. For example, the watchdog monitors the primary processor then in turn monitors a GPS system. In this system if the GPS system hangs, the primary processor can reset the GPS. If the processor hangs, the watchdog can reset the processor which in turn will reset the GPS subsystem. The advantage of this system is that it is not very complex compared to the other peer monitoring schemes. The disadvantage of linear peer monitoring is when the first node, in our example the watchdog, fails and can no longer reset the subsystem it is monitoring. In this case a cascade failure can occur.

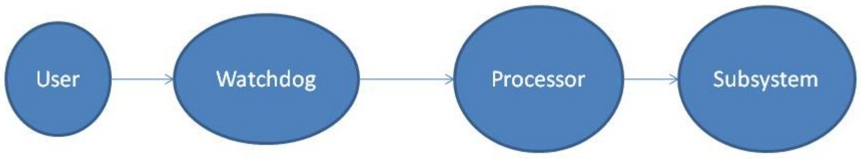


Figure 5: Linear Peer Watching

In order to prevent this, circular peer watching has each subsystem monitor one subsystem with every subsystem being monitored. No subsystem is monitored by more than one subsystem as can be seen in Figure 6. The advantage of circular peer watching is that a separate watchdog processor is not required, therefore an additional cost and hardware complexity is removed from the system. The subsystems that are already there in the hardware to begin with are tasked to monitor heartbeats, buses etc. One of the disadvantages is that if a system wide event occurs which knocks out multiple subsystems, it may not be possible to recover the system. In addition the added software complexity and costs to develop the software outweighs the costs to implement a standard watchdog architecture which was previously discussed.

In spoke peer monitoring, every subsystem monitors every other subsystems as can be seen in Figure 7. One of the primary advantages of this type of system is that if

Figure 6: Circular Peer Watching

multiple failures occur in the system, any subsystem will be able to detect the fault and bring the system back on-line. One of the disadvantages of this type of system is that the complexity of the hardware and the software increase over that of simpler architectures. Safety features also need to be built in to prevent a subsystem with a fault from continually resetting other functioning subsystems. This type of system can provide the most robust type of watchdog monitoring but it comes at a much higher cost.

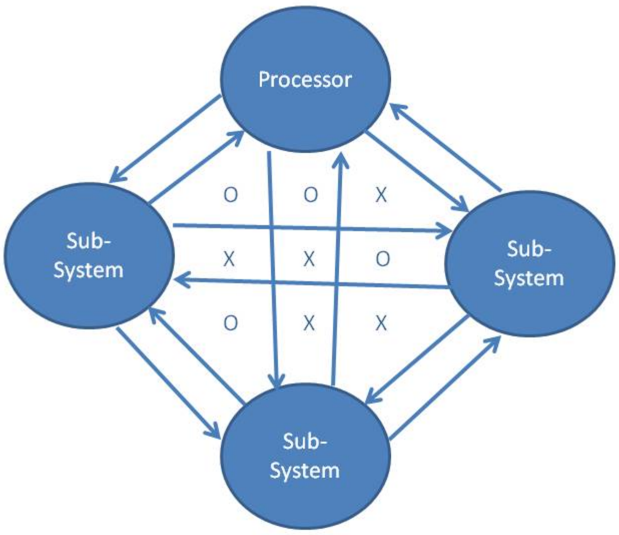


Figure 7: Spoke Peer Watching

In addition to using any of these peer watching architectures, it is also possible to use a combination of them in order to arrive at a system that best balances complexity, cost and software development. It would be possible to develop a circular architecture which then has a single subsystem monitoring a node in a linear peer monitoring fashion.

4.2.5 System Watchdogs

System watchdogs are a watchdog architecture in which the watchdog acts as a monitoring program for failures in redundant subsystems. This type of architecture is used when the system employs dynamic redundancy as a fault tolerant feature. Dynamic Redundancy is a monitoring technique in which each subsystem is duplicated and in the event of a failure, the system (in our case the watchdog) can reconfigure the subsystems to respond to the failures to prevent the faults from affecting the system operation[1].

The system watchdog would monitor the results off the subsystems and when a fault is detected, it would go through a process of resetting and power cycling in order to bring the faulted system back on-line. In the event that the watchdog is unsuccessful, the faulted system can be disconnected from the bus and a backup system connected and brought on-line to take the faulted subsystems place[14]. This type of system is also known as a clustered architecture [7].

The greatest advantage of a system watchdog which can reconfigure the system and remove faulted subsystems is that the system is very robust and fault tolerant. However the major disadvantage is the complexity and cost of the system due to redundant subsystems and potentially due to costs in the development of the fault detection software.

5 Characteristics of a Good Watchdog

A properly designed watchdog is more than a timer that drives a reset pin. It is a last line of defense against faults that will interrupt system operations. In order to create a system which is robust, responsive and capable of handling and recovering from faults, a number of guidelines were identified in [13] which should be followed when designing a proper watchdog and are listed below:

1. The clocks of the watchdog and subsystem being monitored should be separate. If possible, the watchdog should have a dedicated clock generated by an RC or crystal oscillator. If the subsystem clock freezes for some reason this will still allow the watchdog to reset the system.

2. Refreshing the watchdog should be done in such a way that the chances of runaway code accidentally refreshing the watchdog are minimized.

3. Runaway code should be detected quickly in order to prevent the system from entering into an unknown state that can damage the system.

4. The critical control and configuration register bits of the watchdog should have write protection on them so that once set they cannot be accidentally modified.

5. The watchdog should be able to detect that caused a timeout has occurred.

6. Each task within code can set a flag if the task was executed successfully. When all flags check out good, the subsystem can then refresh the watchdog. When the watchdog refresh task is ran, if the flags are not all set, then the watchdog is not refreshed.

6 System Reliability

When designing an embedded system, it is important to have an understanding of the reliability of the system. Whether a system is intended for us on earth or in space, you want to guarantee to some degree that the system will work when you need it to. Reliability can be defined as the probability that the system will not fail for a given period of time under specified operating conditions [15].

There are three characteristics which will determine a systems reliability R(t); failure rate λ (t), cumulative failure probability F(t) and failure probability density f(t).

The failure rate of the system is the rate at which failures occur during a specified amount of time. It can be defined mathematically as

$$lamda(t) = -(1/R)dR/dt \tag{1}$$

or

$$lamda(t) = f(t)/(1 - F(t)) \tag{2}$$

The cumulative failure rate of the system is the probability that the system will fail. It can be defined mathematically as

$$F(t) = \int_0^t f(\lambda)d\lambda \tag{3}$$

or

$$F(t) = 1 - R(t) \tag{4}$$

The failure probability density is the probability that a failure will occur in a specified amount of time. It can be defined mathematically as

$$f(t) = -dR(t)/dt \tag{5}$$

or

$$f(t) = \lambda(t)R(t) \tag{6}$$

Reliability can be defined mathematically as

$$R(t) = \int_t^\infty f(\lambda)d\lambda \tag{7}$$

or

$$R(t) = 1 - F(t) \tag{8}$$

When developing a fault tolerant system, the system designer is interested in knowing the Mean Time To Failure (also known as Mean Time Between Failures) and the Mean Time to Repair [20]. The Mean Time to Failure (MTTF) can be defined as the time it takes between starting the system (from a cold start or from a reset condition) to when a fault occurs. The Mean Time to Repair (MTTR) is the amount of time it takes to detect and correct the fault (possibly by a watchdog resetting the system).

While Mean-Time-To-Failure and Mean-Time-To-Repair are of interest, the system designer really wants to understand how much of the time the system will be working properly. Availability (A) is the average time the system is working properly [20] as defined below:

$$A = MTTF/(MTTF + MTTR) \tag{9}$$

7 Designing a Watchdog

Before beginning to design a watchdog, there are a number of parameters which first need to be identified in order to develop the most cost effective, robust watchdog. The first step is to determine the required level of reliability. There are five levels of reliability to which a system can operate as follow [7]:

1. Basic automatic fault detection by watchdog, no automatic recovery, no data consistency
2. Basic automatic fault detection by watchdog, automatic fault recovery, no data consistency
3. Level 2 reliability plus enhanced automatic fault detection by watchdog with periodic check pointing, logging and recovering internal state
4. Level 3 reliability with persistent data recovery
5. Continuous operation without interruption

Choosing which of these reliability levels the system needs to operate to will determine which watchdog architecture is used and determine the complexity and cost of the watchdog.

Once the watchdog architecture has been determined, the characteristics of the system which are going to be monitored will need to be selected. At this point, the possible inputs which are referred to earlier in this paper need to be examined. A detailed understanding of the timing of the software will need to be evaluated in order to properly select a timeout for the watchdog [12]. This includes the frequency of the interrupts. For a system where the exact timing of the system is variable due to asynchronous communication, the watchdog timeout can be set to a few seconds or longer, just to verify that the entire system hasn't hung up. Another important factor in determining the timeout is the time period in which damage to the system could happen if errant code begins to run.

When the signals that the watchdog is going to monitor have been selected and the general timing has been determined, it's time to look at how the watchdog will recover the system. The possible outputs of the watchdog were discussed in detail earlier in the paper and should be reexamined to understand possible options. It is not unusual for a watchdog to employ multiple options for its output. A watchdog processor may attempt communication with the system it is monitoring. If that fails it may move to a higher level of recovery by resetting the processor. If the watchdog determines that this was not effective it can move to restarting the entire system before deciding to remove the subsystem from the bus and enable a redundant subsystem.

7.1 Assessment

As discussed earlier in the paper there are two methods used to protect a space system, fault avoidance and fault tolerance. Each technique offers unique advantages and disadvantages and the selection of which technique should be used is dependent on a variety of factors which include development time, budget, mass, size and power budget to name a few.

In a comparison between fault avoidance and fault tolerance hardware development (seen in Table 1 below), there are a number of interesting parameters which reveal themselves. Fault tolerant hardware is more expensive and takes longer to develop than that of hardware for a fault tolerant system. A standard Virtex 5 FPGA costs approximately $10,000 where the rad hard version which would be used in a fault avoidance system costs in excess of $100,000. The lead-time for these components was also on the order 12 weeks for the off the shelf version verse a year(?) for the rad hard version.

The hardware for a fault tolerant system is more complex due to the addition of the watchdog and additional fault tolerant circuitry. This adds to the component and design costs. However, based on the desired level of reliability, it is likely possible that the hardware for the fault tolerant design will overall be less expensive and less time consuming to develop and launch than a system based on the fault avoidance techniques. There is also the advantage to fault tolerant design in that the clock frequencies operate at higher frequencies and use less power than the fault avoidance counterparts.

Hardware	
Fault Avoidance	Fault Tolerance
Higher Costs	Lower Costs
Fewer Components	More Components
Less Complex	More Complex
Long Lead Components	Available Components
Longer Design Cycle	Shorter Design Cycle
Lower Operating Frequency	Higher Operating Frequency
Higher Power	Lower Power

Table 1: Comparison of Fault Avoidance and Tolerance Hardware Design

While fault tolerance appears to have an advantage over hardware design, it is not necessarily the case for the software development. Fault tolerant techniques require additional software complexity and design time depending on the level of reliability required. Firmware costs on average \$15 - \$30 per line of code with extreme cases such as the space shuttle where the cost is \$1000 per line of code [25]. The higher the reliability level that is required for the application, the more complex the code will become which equates to additional overhead.

In the time domain, fault tolerant systems using COTS components operate at faster system frequencies than fault avoidance systems. This can allow an algorithm or task to run multiple times and then vote on the result in order to avoid a fault. While repeating calculations and tasks increases the in which things are performed, cost and complexity of the system have been saved. This causes time and power to increase but this is the trade for using COTS and running on a lower budget (can we compare frequency and power of COTS vs RAD parts?) data for system costs?

In an environment where resources and time are abundant, the preferred method of system development would be to use fault avoidance. Fault avoidance is useful for projects with long development schedules, deep pockets, and a spacecraft with sufficient size to support the power budget. Fault avoidance systems typically have less processing power than COTS components. In a fault avoidance system, errors will occur but the mean time to failure has been increased beyond the usable life of the mission.

When budgets, time, and resources are constrained, the most effective design is to use fault tolerance. A fault tolerant system is built knowing that faults are going to occur but that the system will detect and respond to the faults. In order to do this, the watchdog is used. In fault tolerant systems, there are two critical axises which need to be examined, time and space. As is implied, time is the rate at which a task or work is done while the space is the amount of volume that the system takes. Fault tolerant systems are less complex, costly and application size if often smaller as well [6]. By using COTS parts power, volume and time to develop are drastically decreased.

Cubesats are not only budget, time and resource but also have to adhere to strict space and power requirements. They are also limited to dimensions as small as 10 cm by 10 cm by 10 cm with a mass of 1 to 3 kilograms. This makes Cubesat missions the perfect candidate for using fault tolerant techniques. With a limited budget, a cubesat cannot afford to purchase rad hard components and the mass and space requirements also limit the use of redundant components and complex fault avoidance systems. However, a cubesat can take advantage of the latest technologies in data storage to store redundant data and perform operations multiple times in order to reduce the risk of a fault.

Figure 8 attempts to quantify the selection of a specific type of fault tolerant architecture based on reliability parameters for the system. As can be seen from the graph, architectures and sub-features can be overlapped in order to improve the reliability of an individual architecture. The goal is to define the reliability requirements and then based on those requirements use Figure 8 to determine the architecture and architecture features to guide the development of the watchdog.

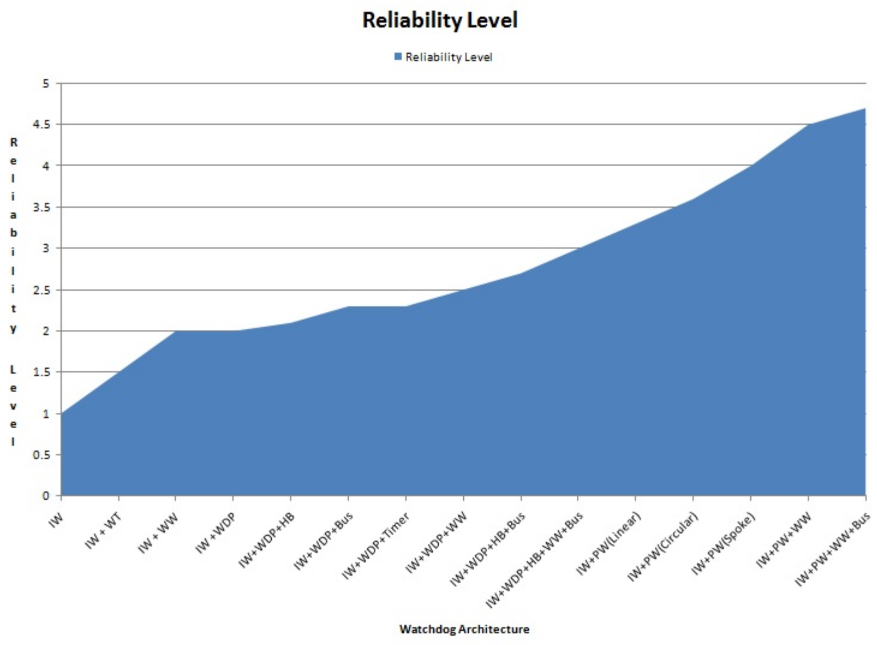


Figure 8: Selecting a Watchdog Architecture based on Reliability Requirements
IW - Internal Watchdog
WT - Watchdog Timer
WW - Windowed Watchdog
WDP - Watchdog Processor
Bus - Bus Monitor
HB - Heartbeat
PW - Peer Watching

8 Watchdog Case Study: RAX Cubesat

The Radio Aurora eXplorer(RAX) is a 3U (3 unit) CubeSat with a volume of 10 cm x 10 cm x 30 cm and a mass of approximately three kilograms. The RAX mission is a perfect example of a space system suited for COTS components and a fault tolerant system architecture due to restrictions in development time, size, mass, cost and power budget.

RAX was built using common off the shelf components. The flight computer (FCPU), consists of a TI MSP430F1611 16 bit micro-controller running at 8 MHz and consuming less than 5 mA in active mode. The FCPU performs all primary tasks on the spacecraft such as communications, task scheduling and experiment setup. The FCPU controls which subsystems are turned on and connected to the communications bus at various stages throughout the mission.

In order to minimize system down time and risks due to transient faults, the RAX FCPU is monitored by a watchdog system. In this section we will investigate the design process used to determine the best watchdog architecture for use on RAX and how the watchdog system was implemented in order to improve overall system reliability and mission success.

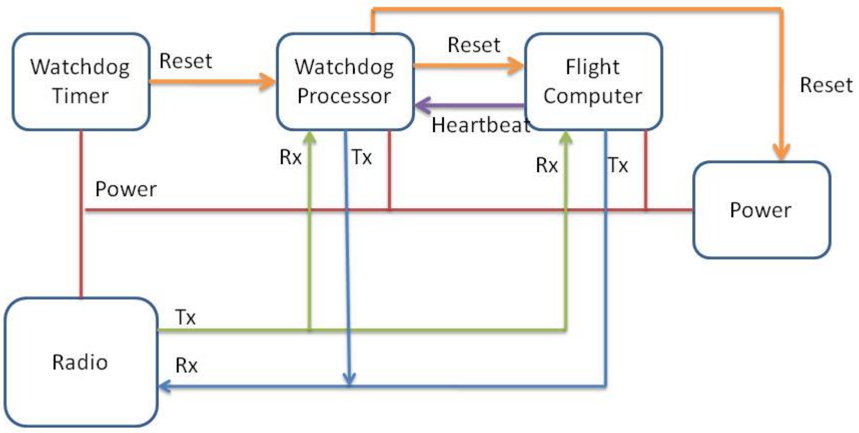
8.1 Flight Computer Watchdog Overview

During the development of RAX, it was determined that the satellite would need to meet a system reliability level of three in order to successfully carry out the mission. As can be recalled from section 7, a system reliability level of three states that the watchdog will be able to detect faults, automatically recover from the faults, perform periodic check pointing, logging and recovery of the internal state.

The minimum watchdog architecture that can successfully implement a reliability level of three can be found using the figure "Required Reliability and Watchdog Architecture Selection for fault tolerant Systems" in section 7.1. It was determined that the RAX watchdog would consist of an internal watchdog to the FCPU in addition to an external watchdog processor. The watchdog processor selected was an 8 bit TI MSP4302011 processor. This particular processor offered a number of advantages such as small footprint, low power and familiarity due to the fact that it is from the same architecture family as the FCPU processor. However, there is a disadvantage to this processor in that being the same architecture, a large scale transient event that affects the FCPU could potentially also affect the watchdog. This however was minimized by placing the watchdog on a separate board within the stack and by offsetting the watchdog from a direct line with the FCPU processor so that the two processors had no x,y,z coordinates in common.

The RAX watchdog monitors two system parameters for faults; a heartbeat and a communication bus. The FCPU generates a heartbeat at an interval of 1 Hz. In the event that the watchdog does not detect a heartbeat for four minutes, corrective actions are taken to put the FCPU back into a known state. Details of how the heartbeat is generated, monitored and what corrective actions are taken to recover the system will be covered in detail in section 8.2. In addition to monitoring the heartbeat from the FCPU, the watchdog also monitors the communication bus between the FCPU and the on-board Lithium radio. When commands are transmitted to the spacecraft, the watchdog parses the data and looks for a command designated for the watchdog such as resetting the FCPU or power cycling the spacecraft. Details of how the bus is monitored will be covered in detail in section 8.3.

In the event that a fault is detected, the watchdog has the ability to detect the fault and handle it by resetting the FCPU and by power cycling the spacecraft. In either case, when the FCPU is reset, all subsystems are powered off while the FCPU goes through its initialization sequence. The FCPU stores its internal state and a list of commands internally on a space rated SD card. In the event of a system restart, the FCPU is able to recover the commands which were being executed at the time of the fault and then execute those commands in order to put each subsystem back into a known state.



1) Watchdog Timer set to reset Watchdog Processor periodically.
2) Watchdog Processor monitors heartbeat. If heartbeat not received in 5 minutes the flight computer is reset.
3) Watchdog Processor monitors communication bus for flight computer reset command.
4) Watchdog Processor monitors communication bus for spacecraft reset command.
5) Internal Watchdog Timer in the Watchdog Processor monitors for flow errors.

Figure 9: Selecting a Watchdog Architecture based on Reliability Requirements

The RAX watchdog timer has five states which can be seen in Figure 10; Watchdog Init, Watchdog Monitor, Watchdog Transmit, Reset FCPU and Reset Spacecraft. Watchdog Init is when the watchdogs watchdog timer resets the watchdog and must reinitialize itself. From this state, the watchdog moves into a state where it monitors ground communications and the heartbeats generated by the FCPU. If a ground command is received the watchdog moves into the watchdog transmit state to transmit an acknowledgment before returning to the monitor state and either moving to the Reset FCPU or Reset Spacecraft states. Missed heartbeats will also move the watchdog from the Watchdog Monitor state to the Reset FCPU state.

8.2 Heartbeat Monitoring of the Flight Computer

The FCPU generates a standard heartbeat signal which has a frequency of 1 Hz. The heartbeat is generated using the recommendation from section 3.1. The heartbeat is NOT generated using an internal hardware timer but is instead generated by a watchdog function within the primary task. This decreases the likely hood that the watchdog function will be ran during a fault. At this time, the heartbeat does not use a task flagging system to generate the heartbeat but only runs if the main program task has executed successfully. While a flagging system increases the reliability of the system, the flight software timing is complex and difficult to ascertain due to its changing nature.

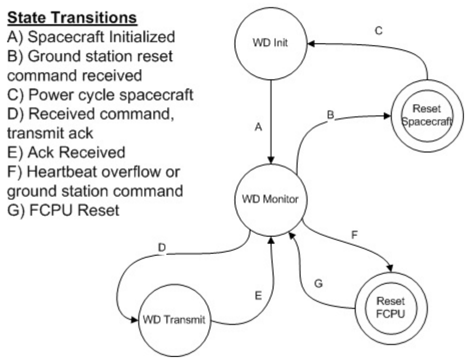

State Transitions
A) Spacecraft Initialized
B) Ground station reset command received
C) Power cycle spacecraft
D) Received command, transmit ack
E) Ack Received
F) Heartbeat overflow or ground station command
G) FCPU Reset
WD Init
Reset Spacecraft
WD Monitor
WD Transmit
Reset FCPU
A
B
C
D
E
F
G

Figure 10: Watchdog State Diagram

The reason for this is the number of asynchronous data communication tasks and the modularity of the system which allows tasks to be turned on and turned off. Developing and maintaining a flagging system for the watchdog would increase software complexity and decrease system performance in this particular instance.

The watchdog processor which monitors the heartbeat is setup to behave in a similar manner as a standard watchdog timer. However, when the internal timer expires and a heartbeat has not been received, instead of resetting the FCPU immediately the watchdog records that the heartbeat was missed, reloads the counter and continues to wait for a heartbeat. If a heartbeat is not received in six minutes, then the watchdog takes corrective actions.

There are a couple of reasons why the watchdog does not immediately act upon a missed heartbeat. First, each experiment performed on RAX lasts for approximately five minutes. Once the experiment is started, even if the flight computer locks up, each subsystem will still be able to collect data for the experiment. If the flight computer is immediately reset, it will take seconds for the system to reinitialize and prepare for the experiment which will result in lost data. By allowing six minutes of missed heartbeats to occur will allow any subsystems collecting data to complete the data collection process and go into an idle state before taking corrective actions. There are also communication events such as file transfers from the ground which can allow the heartbeat to take longer than one second to execute which further justifies counting missed heartbeats and resetting after a defined number of heartbeats are missed.

Using the watchdog to count missed heartbeats and only take corrective actions once heartbeats have been missed for a period of time does introduce the possibility for a case in which the watchdog will not be able to detect that a fault has occurred in the flight computer. For example, if the flight software has moved into an unknown state and is still sending heartbeats once every minute or 500 times per minute, the watchdog would not reach the six minute limit set and would not attempt to recover the system. In order to avoid this, additional logic has been built into the watchdog to monitor for sporadic heartbeat generation.

This sporadic heartbeat monitor consists of counting the number of heartbeats which occur in a six minute window. In a six minute window the watchdog would expect to receive 360 heartbeats. This monitor creates a window in which the watchdog expects to see between 300 and 420 heartbeats per six minute interval. If more or less heartbeats are detected, then the watchdog will recognize that a fault has occurred and begin operations to recover the system and put it back into a known state. This window was selected in order to account for missed heartbeats that could be the result of file transfers and ground communications.

8.3 Watchdog Processor Watchdog Timer

The watchdog processor improves the overall system reliability but like any other processor it is just as susceptible to transient events as any other processor. So if the watchdog is watching the flight computer then who is watching the watchdog? For RAX, the watchdog is monitored by a simple space rated watchdog timer.

The watchdog timer consists of a 555 timer whose output is then fed into a 24 stage frequency divider. The frequency divider is setup to latch an output when a specified number of pulses from the watchdog timer have been received. When the output latches, the watchdog processor is reset in addition to the watchdog timer circuitry being reset to begin the count again.

The watchdog timer circuit was designed to reset the watchdog period every hour, but can be set to any time period from seconds to weeks. The period is set in hardware through the adjustment of an RC circuit, and therefore cannot be turned off or modified by any software. The watchdog timer does not monitor a heartbeat from the watchdog processor but instead provides a method to guarantee that the watchdog processor is in a known state once every hour. The reasons for this was to have a simple design that consumed the least number of pins on the watchdog processor so that the watchdog itself could remain a simple system. Adding additional software to the watchdog processor which would not in itself be considered complex would still add to the complexity and size of the code and provide additional avenues in which bugs could creep into the design. A simple periodic reset is sufficient.

The watchdog timer was configured for a reset every hour for a number of reasons. First, the reset period should be larger than the window in which the watchdog processor is monitoring the FCPU, which in this case is 6 minutes. This allows for a simpler watchdog processor that does not have to save its state flash. Secondly, due to the simplicity of the watchdog processor code, it is not as likely for it to reach a non-recoverable state as the flight computer due to its limited flash size and simple functions. Therefore the watchdog timer was added as a last line defense to make sure that the watchdog processor gets a periodic refresh.

8.4 Ground Communication Bus Monitoring

In addition to monitoring the heartbeat from the flight computer, the watchdog processor also has two of its general purpose input output pins connected to the transmit and receive lines of the radio communication bus. By doing this, the watchdog processor is capable of not only monitoring incoming radio packets but also generating radio packets of its own. The primary reason for this is that it allows the watchdog processor to accept commands from the ground and transmit status and command acknowledgments.

The watchdog processor monitors a standard UART communications bus; However, the watchdog processor does not have an internal hardware UART. The watchdog processor was selected without a hardware UART in order to reduce part complexity, cost and power consumption. In order to receive and transmit commands, a software UART was developed to receive and transmit data. The software UART consists of a simple state machine which allows for Receiving, Transmitting and Processing of byte data.

When data is received from the ground, the watchdog reads in each byte and processes it in accordance with the command set that is stored in the watchdog. The watchdog monitors for two commands. The first is a FCPU reset command. If a fault goes undetected or mission operators decide they want to reset the flight computer, this command will do exactly that when the watchdog decodes it. The second, is a spacecraft reset command. When the watchdog receives this command the entire spacecraft is power cycled and brought up fresh from a known state. Before each command is executed, the watchdog transmits an acknowledgment.

The communication bus monitoring can also be viewed as a peer watching architecture. In this case, the mission operators are monitoring the watchdog which is then monitoring the flight computer. This architecture is representative of the simplest of the peer watching architectures known as linear peer watching.

8.5 Watchdog Processor Internal Watchdog Architecture

In addition to the external watchdog timer, the watchdog processor also has its own internal watchdog. The purpose of this watchdog is to detect internal faults and put the watchdog back into a known state. There are two conditions that must be met in order for the internal watchdog to be refreshed.

The first, is that the function to test if there is a byte in the UART buffer that needs to be processed must have ran. When this function runs, its set a flag which indicates that the task ran successfully. Second, the function that tests to see if the FCPU needs to be reset or the spacecraft power cycled needs to run. When this function is ran, it sets a flag as well. When both flags are set, the watchdog task knows that both executed and refreshes the watchdog. In the event that only a single flag or neither flag is set, the watchdog task knows that the code has faulted and will reset itself. If both flags are set, the watchdog will refresh the watchdog timer and clear the flags.

8.6 Watchdog Recovery

The watchdog on RAX has two methods which it uses to recover the spacecraft when faults are detected. Resetting the flight computer and power cycling the entire spacecraft. Each fault detection method goes about implementing these two methods in its own way.

The heartbeat monitor only has the ability to reset the flight computer. While it is possible to have the watchdog track the number of consecutive faults caused by missed heartbeats and then move to a more aggressive reset of power cycling the spacecraft, this adds complexity to the watchdog software. In order to track its state, the watchdog would need to be able to store its state to flash. This is possible with the watchdog processor that has been selected; However, the code space required to properly write flash segments and maintain the state would require moving to a more capable processor. This had the impact of increasing costs, power usage and increasing the software complexity and design time. The primary goal of the watchdog is to be as simple as possible. It was therefore decided that the watchdog would only reset the processor based on heartbeat faults.

The communications bus monitor however is capable of both resetting the flight computer and power cycling the spacecraft. It is up to the mission operators to determine when it is necessary to perform these functions. It would be standard procedure for them to attempt to reset the flight computer before moving to the aggressive method of power cycling the entire spacecraft.

8.7 System Reliability

It is useful to gain insight into the expected system reliability of the system and determine the probability of a permanent or transient failure occurring in addition to an understanding of how the watchdog improved the system reliability. According to [22], the permanent fault rate for a space based system is $1x10^{-8}$ per hour and the transient fault rate is 3 (3 - 18 worst case during solar flare). For this discussion it will be assumed that the orbit is 90 minutes which results in the permanent and transient fault rates of $1.5x10^{-8}$ and 5 respectively.

Using the mathematical functions in section 6, we can show that the probability of an orbit without a permanent failure is $Psuccess = e^{-1*(1.5x10^{-8})} = 0.999999985$. A two year mission consists of approximately 116,800 orbits. The probability of successfully completing a two year mission without a permanent failure is $Psuccess = e^{-116800*(1.5x10^{-8})} = 0.998249$. The chance of surviving for a 10 year mission is $Psuccess = e^{-1168000*(1.5x10^{-8})} = 0.9826$.

Now the RAX radio, watchdog and flight computer are all based on similar microcontroller architectures. All three systems are required in order to operate successfully. The combined probability that the entire system can survive for a two year mission is $Psuccess = P(1) * P(2) * P(3) = 0.998249^3 = 0.9947$. The success rate of a ten year mission would be $Psuccess = P(1) * P(2) * P(3) = 0.9826^3 = 0.9487$

The event that is most likely to affect the system are transient faults. Using the 5 events per orbit which is on the low end of worst case occurance, for a single orbit the chance that the flight computer will not be affected by a transient fault is $Psuccess = e^{-1*(5)} = 0.00673$. However, a properly implemented internal watchdog has a minimum effect of catching and recovering the system 85% of the time. The combined probability of successfully completing an orbit without an upset now becomes $Psuccess = P(1) * P(2) = 0.00673 * 0.85 = 0.85$

Adding a watchdog processor with a similar architecture yields a probability of 0.85 that the watchdog will operate over the course of an orbit. When combining the probabilities that the system will function during an orbit yields $Psuccess = P(1) + P(2) - P(1) * P(2) = 0.85 + 0.85 - (0.85)(0.85) = 0.97$. While adding a radio reset to the system adds an additional method to reset and power cycle the system, the probability of this successfully is lower due to its added linear probability. This can be shown by $Psuccess = P(1) * P(2) = 0.85 * 0.97 = 0.83$

9 Conclusions

While there are a number of methods which exist to protect a space based system from transient faults produced by the space environment, a combination of fault tolerant and watchdog architecture effectively protect space systems. Based on the mission requirements for reliability, it is possible to select an architecture which will minimize cost and complexity while optimizing the probability for successful operation.

References

[1] Vinod B. Prasad, *Fault tolerant digital systems.* IEEE, 1989.

[2] Aamer Mahmood and E.J. McCluskey, *Concurrent Error Detection Using Watchdog Processors - A Survey.* IEEE, 1988.

[3] Alfredo Benso and Stefano Di Carlo and Giorgio Di Natale and Paolo Prinetto, *A Watchdog Processor to Detect Data and Control Flow Errors.* Proceedings of the 9th IEEE International On-Line Testing Symposium, 2003.

[4] Ashraf M. El-Attar and Gamal Fahmy, *A study of fault coverage of standard and windowed watchdog timers.* IEEE International Conference on Signal Processing and Communications, 2007.

[5] Mohamed S. Hefny and Hassanein H. Amer, *Design of an improved watchdog circuit for microcontroller-based systems.* IEEE, 1989.

[6] Michael J. Pont and Royan H.L. Ong, *Using watchdog timers to improve the reliability of single-processor embedded systems: Seven new patterns and a case study.* Proceedings of the First Nordic Conference on Pattern Languages of Programs, 2002.

[7] Michael R. Lyu and Veena B. Mendiratta, *Software Fault Tolerance in a Clustered Architecture: Techniques and Reliability Modeling.* IEEE, 1999.

[8] Lars Grunske *Transformational Patterns for the Improvement of Safety Properties in Architectural Specification.* Proceedings of The Second Nordic Conference on Pattern Languages of Programs, 2003.

[9] Christian Webel and Ingmar Fliege and Alexander Geraldy and Reinhard Gotzhein *Developing Reliable Systems with SDL Design Patterns and Design Components.* WITUL, 2004.

[10] Ashraf M. El-Attar and Gamal Fahmy, *An Improved Watchdog Timer to Enhance Image System Reliability In The Presence Of Soft Errors.* IEEE International Symposium on Signal Processing and Information Technology, 2007.

[11] Jack Ganssle *Great Watchdogs.* The Ganssle Group, 2004.

[12] Niall Murphy *Watchdog Timers.* Embedded.com, 2000.

[13] Suahs Chakravarty and Rohit Tomar and Mohit Arora *Need a watchdog for improved system fault tolerance?.* commsdesign.com, 2008.

[14] G. Ken Hunter and Neil C. Rowe *Software Design For a Fault-Tolerant Communications Satellite.* Naval Security Group Activity, MD, 2000.

[15] Lisa Guerra *Reliability Module.* NASA's Exploration Systems Mission Directorate , 2008.

[16] A. Christy Presya and T.R. Gopalakrishnan Nair *Fault Tolerant Real Time Systems.* International Conference on Managing Next Generation Software Application , 2008.

[17] Savio N. Chau and Leon Alkalai and Ann T. Tai *The Design of a Fault-Tolerant COTS-Based Bus Architecture for Space Application.* IEEE , 1999.

[18] Audra Bullock *SCI Team PDR Report.* UH Cubesat Project, Hawii , 2002.

[19] Len Adams *Space Radiation Effects in Electronic Components.* Brunell University, 2003.

[20] Sverre Vigander *Evolutionary Fault Repair of Electronics in Space Applications.* IEEE , 2005.

[21] I V McLoughlin and V Gupta and G S Sandhu and S Lim and T R Bretschneider *Fault tolerance through redundant COTS components for satellite processing applications.* IEEE Information, Communications and Signal Processing , 2003.

[22] A. M. Finn *System Effects of Single Event Upsets.* Computers in Aerospace VII Conference, CA , 1989.

[23] James W. Cutler and Armando Fox and Kul Bhasin *Applying the Lessons of Internet Services to Space Systems.* Proceedings of the IEEE Aerospace Conference, Big Sky, Montana, 2001.

[24] Philip P. Shirvani *COTS Technology and Issues - Space Environments.* 44th Meeting of IFIP Working Group 10.4, 2003.

[25] Jack Ganssle *The Art of Designing Embedded Systems Second Edition* . Newnes Burlington, MA, 2008.