IN-59
85775

NASA Technical Memorandum 107568

P.100

A Verified Design of a Fault-Tolerant Clock Synchronization Circuit: Preliminary Investigations

Paul S. Miner

March 1992

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, VA 23665

Abstract

Schneider [1] demonstrates that many fault-tolerant clock synchronization algorithms can be represented as refinements of a single proven correct paradigm. Shankar [2] provides a mechanical proof (using EHDM [3]) that Schneider's schema achieves Byzantine fault-tolerant clock synchronization provided that eleven constraints are satisfied. Some of the constraints are assumptions about physical properties of the system and can not be established formally. Proofs are given (in EHDM) that the fault-tolerant midpoint convergence function satisfies three of these constraints. This paper presents a hardware design, implementing the fault-tolerant midpoint function, which will be shown to satisfy the remaining constraints. The synchronization circuit will recover completely from transient faults provided the maximum fault assumption is not violated. The initialization protocol for the circuit also provides a recovery mechanism from total system failure caused by correlated transient faults.

Contents

1 Introduction

NASA Langley Research Center is currently involved in the development of a formally verified Reliable Computing Platform (RCP) for real-time digital flight control systems [4, 5, 6]. An often quoted requirement for critical systems employed for civil air transport is a probability of catastrophic failure less than 10^{-9} for a 10 hour flight [7]. Since failure rates for digital devices are on the order of 10^{-6} per hour [8], hardware redundancy is required to achieve the desired level of reliability. While there are many ways of incorporating redundant hardware, the approach taken in the RCP is the use of identical redundant channels with exact match voting (see [4, 5] and [6]).

A critical function in a fault-tolerant system is that of synchronizing the clocks of the redundant computing elements. The clocks must be synchronized in order to provide coordinated action among the redundant sites. Although perfect synchronization is not possible, clocks can be synchronized within a small skew. The purpose of this work is to provide a mechanically verified design of a fault-tolerant clock synchronization circuit.

The fault-tolerant clock synchronization circuit is intended to be part of a verified hardware base for the RCP. The primary intent of the RCP is to provide a verified fault-tolerant system which is proven to recover from a bounded number of transient faults. The current model of the system assumes (among other things) that the clocks are synchronized within a bounded skew [5]. It is crucial that the clock synchronization circuitry also be able to recover from transient faults. Originally, Lamport and Melliar-Smith's Interactive Convergence Algorithm (ICA) [9] was to be the basis for the clock synchronization hardware, the primary reason being the existence of a mechanical proof that the algorithm is correct [10]. However, modifications to ICA to achieve transient fault recovery are unnecessarily complicated. The fault-tolerant midpoint algorithm of [11] is more readily adapted to transient recovery.

The synchronization circuit is designed to tolerate arbitrarily malicious permanent, intermittent and transient hardware faults. A fault is defined as a physical perturbation altering the function implemented by a physical device. Intermittent faults are permanent physical faults which do not constantly alter the function of a device (e.g. a loose wire). A transient fault is a one shot short duration physical perturbation of a device (e.g. caused by a cosmic ray or other electromagnetic effect). Once the source of the fault is removed, the device can function correctly.

Most proofs of fault-tolerant clock synchronization algorithms are by induction on the number of synchronization intervals. Usually, the base case of the induction, the initial skew, is assumed. The descriptions in [1, 2, 9, 10] all assume initial synchronization with no mention of how it is achieved. Others, including [11, 12, 13] and [14] address the issue of initial synchronization and give descriptions of how it is achieved in varying degrees of detail. In proving an implementation correct, the details of initial synchronization cannot be ignored. If the initialization scheme is robust enough, it can also serve as a recovery mechanism from multiple correlated transient failures (as is noted in [14]).

Schneider [1] demonstrates that many fault-tolerant clock synchronization algorithms can be represented as refinements of a single proven correct paradigm. Shankar [2] provides a mechanical proof (using EHDM [3]) that Schneider's schema achieves Byzantine fault-tolerant clock synchronization, provided that eleven constraints are satisfied. Some of the constraints are assumptions about physical properties of the system and can not be established formally. This paper proposes a hardware solution to the clock synchronization problem which will be shown to satisfy the remaining constraints.

This paper discusses preliminary results in the verification of the design. The fault-tolerant midpoint function is formally proven (in EHDM) to satisfy the properties of translation invariance, precision enhancement, and accuracy preservation.[1] A register transfer level design is presented which implements the synchronization algorithm. An argument for transient recovery from a single fault is presented and issues relating to the more general case are raised. Finally, the approach for achieving initial synchronization is discussed. The notation used here is from Shankar [2].

2 Description of the Reliable Computing Platform

This section summarizes the key details of the Reliable Computing Platform to establish a context for the clock synchronization circuit. It is included here for completeness. The material in this section is paraphrased from Butler and DiVito [5]. The interested reader should consult [5] for more detailed information.

[1]These properties will be defined in the section describing the fault-tolerant midpoint convergence function.

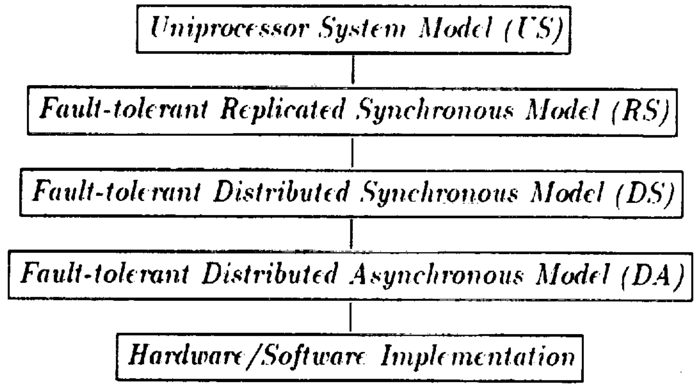
Uniprocessor System Model (US)
Fault-tolerant Replicated Synchronous Model (RS)
Fault-tolerant Distributed Synchronous Model (DS)
Fault-tolerant Distributed Asynchronous Model (DA)
Hardware/Software Implementation

Figure 1: Hierarchical Specification of the Reliable Computing Platform.

NASA-Langley is currently involved in the development of a formally verified Reliable Computing Platform for real-time control [4, 5]. A primary goal is to provide a fault-tolerant computing base that appears to the application programmer as a single ultra-reliable computer. To achieve this, it is necessary to conceal implementation details of the system. Some characteristics of the system are as follows [5]:

- "the system is non-reconfigurable
- the system is frame-synchronous
- the scheduling is static, non-preemptive
- internal voting is used to recover the state of a processor affected by a transient fault"

A hierarchy of models is introduced which provides different levels of abstraction (figure 1, taken from [5]). The top level is the view presented to the applications programmer, i.e. an ultra-reliable uniprocessor system. The details of fault-tolerance are introduced in the lower levels. The next two levels, replicated synchronous and distributed synchronous, introduce the redundancy and voting required for fault-tolerance, but assume perfectly synchronized clocks and an interactive consistency network for reliable distribution of single source data. The fourth level, distributed asynchronous, weakens the assumption of perfect synchrony to one where the clocks are synchronized to within a bounded skew. The details of the hardware/software implementation have yet to be worked out. An abstract view of the assumed

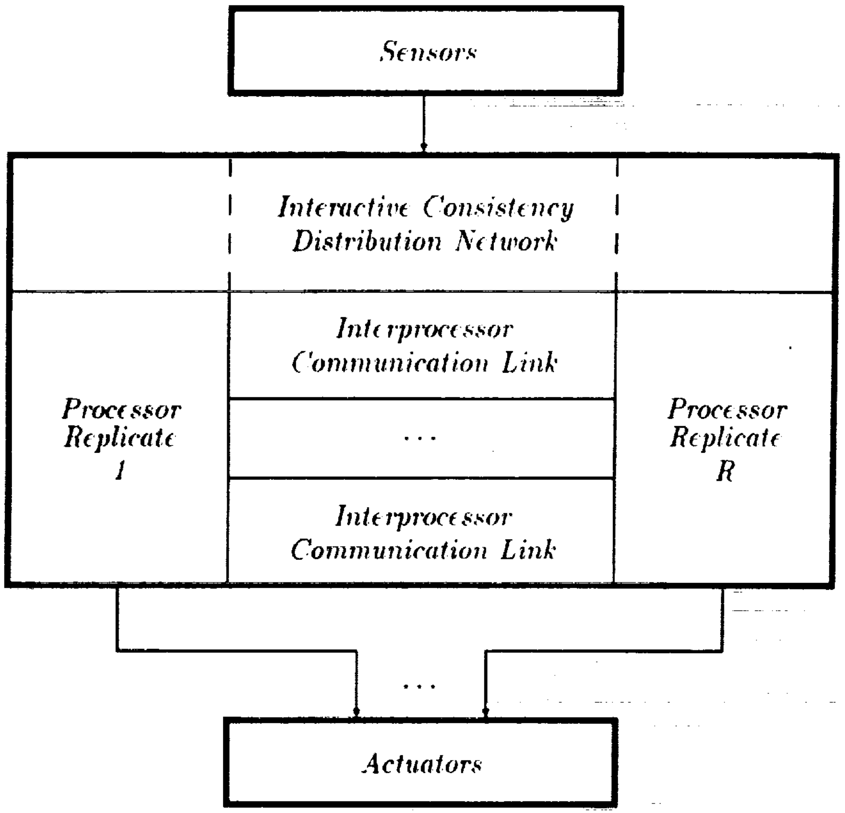
Sensors
Interactive Consistency Distribution Network
Interprocessor Communication Link
Processor Replicate 1
...
Processor Replicate R
Interprocessor Communication Link
...
Actuators

Figure 2: Generic Hardware Architecture

hardware architecture is given in figure 2 (from [5]). The clock synchronization circuit presented here is intended to serve as part of the verified hardware base at the lowest level of the hierarchy.

3 Clock Definitions

This section introduces the notation and assumptions used in Shankar's proof and is largely taken from sections 2.1 and 2.2 of [2]. The conditions enumerated here provide the formal specification for the clock synchronization circuit.

$PC_p(t)$	The reading of p's physical clock at real time t.
$VC_p(t)$	The reading of p's virtual clock at time t. This is the logical time used by the system.
$IC_p^i(t)$	The reading of p's ith interval clock at real time t (Only sensible if $t_p^i \leq t$).
t_p^i	The real time that processor p begins the ith synchronization interval.
adj_p^i	Cumulative adjustment to p's physical clock up to and including t_p^i.
Θ_p^i	An array of clock readings (local to p) such that (for $i > 0$) $\Theta_p^i(q)$ is p's reading of q's clock at t_p^i.
$cfn(p, \Theta_p^i)$	Convergence function executed by p to establish correct $VC_p(t_p^i)$.

Table 1: Clock Notation

3.1 Shankar's Notation

In general, clocks will be represented by different abstractions. Each redundant clock will incorporate a physical oscillator which marks passage of time. Each oscillator will drift with respect to real time by a small amount. Physical clocks derived from these oscillators will similarly drift with respect to each other. The purpose of a clock synchronization algorithm is to make periodic adjustments to local (virtual) clocks to keep redundant clocks within a bounded skew of each other. This periodic adjustment makes analysis difficult, so an interval clock abstraction is used in the proofs. This interval clock is indexed by the number of elapsed intervals since the beginning of the protocol. An interval corresponds to the elapsed time between adjustments to the virtual clock. The proof that synchronization is maintained is by induction on intervals.

Table 1 introduces the notation for the key elements required for a verified clock synchronization algorithm. Shankar outlines the following set of relationships between these values,

$$\begin{aligned} adj_p^{i+1} &= cfn(p, \Theta_p^{i+1}) - PC_p(t_p^{i+1}) \\ adj_p^0 &= 0 \\ IC_p^i(t) &= PC_p(t) + adj_p^i \end{aligned}$$

$$VC_p(t) \;=\; IC_p^i(t), \text{ for } t_p^i \leq t < t_p^{i+1}$$

presuming the presence of PC and VC, with an abstraction for IC used in the proofs. The following can be simply derived.

$$\begin{aligned} VC_p(t_p^{i+1}) &= IC_p^{i+1}(t_p^{i+1}) = cfn(p, \Theta_p^{i+1}) \\ IC_p^{i+1}(t) &= cfn(p, \Theta_p^{i+1}) + PC_p(t) - PC_p(t_p^{i+1}) \end{aligned}$$

Using these equations and the eleven conditions outlined in the next section, Shankar mechanically verified Schneider's paradigm. Some of the conditions will need to be modified in order to reason about transient recovery. It will then be necessary to rerun the EHDM proofs of the main theorem of [2] (below).

Any implementation which satisfies the constraints in Shankar's report will provide the following guarantee.

Theorem 1 (bounded skew) *For any two clocks p and q that are non-faulty at time t,*

$$|VC_p(t) - VC_q(t)| \leq \delta$$

That is, the difference in time observed by two non-faulty clocks is bounded by a small amount. This gives the leverage needed to reliably build a fault-tolerant system. The next section enumerates the conditions to be met to guarantee this result.

3.2 Shankar's Conditions

The first condition is initial skew, δ_S, which is a bound on the difference between good clocks at the beginning of the protocol.

Condition 1 (initial skew) *For nonfaulty processors p and q*

$$|PC_p(0) - PC_q(0)| \leq \delta_S$$

The rate at which a good clock can drift from real-time is bounded by a small constant ρ.[2]

[2]Notice that in this formulation a good clock must have been good continually since time 0. This condition will need to be modified in order to reason about recovery from transient faults.

Condition 2 (bounded drift) *There is a nonnegative constant ρ such that if clock p is nonfaulty at time $s, s \geq t$, then*

$$(1-\rho)(s-t) \leq PC_p(s) - PC_p(t) \leq (1+\rho)(s-t)$$

Shankar notes the following corallary to *bounded drift* which limits the amount two good clocks can drift with respect to each other during interval from t to s.

$$|PC_p(s) - PC_q(s)| \leq |PC_p(t) - PC_q(t)| + 2\rho(s-t)$$

The next four conditions describe some constraints upon the synchronization interval as related to initial conditions of the protocol.

Condition 3 (bounded interval) *For nonfaulty clock p*

$$0 < r_{min} \leq t_p^{i+1} - t_p^i \leq r_{max}$$

Condition 4 (bounded delay) *For nonfaulty clocks p and q*

$$|t_q^i - t_p^i| \leq \beta$$

Condition 5 (initial synchronization) *For nonfaulty clock p*

$$t_p^0 = 0$$

Since we do not want process q to start its $(i+1)$th clock before process p starts its ith we state a nonoverlap condition

Condition 6 (nonoverlap)

$$\beta \leq r_{min}$$

This, with *bounded interval* and *bounded delay*, ensures that for good clocks p and q, $t_p^i \leq t_q^{i+1}$.

All clock synchronization protocols require each process to obtain an estimate of the clock values for other processes within the system. Error in this estimate can be bounded, but not eliminated.

Condition 7 (reading error) *For nonfaulty clocks p and q*

$$|IC_q^i(t_p^{i+1}) - \Theta_p^{i+1}(q)| \leq \Lambda$$

There is bound to the number of faults which can be tolerated[3]

Condition 8 (bounded faults) *At any time t, the number of faulty processes is at most F.*

For the purpose of the algorithm presented here, we will assume that the number of clocks, N, satisfies the inequality $N \geq 3F + 1$.

Synchronization algorithms execute a convergence function $cfn(p, \theta)$ which must satisfy the conditions of *translation invariance*, *precision enhancement*, and *accuracy preservation* irrespective of the physical constraints on the system. Shankar mechanically proves that Lamport and Melliar-Smith's Interactive Convergence function [9] satisfies these three conditions. The next section defines these conditions in the context of the fault-tolerant midpoint function used by Welch and Lynch [11].

4 Fault-Tolerant Midpoint as an Instance of Schneider's Schema

The convergence function for the implementation described here is the fault-tolerant midpoint used by Welch and Lynch in [11]. The function consists of discarding the F largest and F smallest clock readings, and then determining the midpoint of the range of the remaining readings. Its formal definition is

$$cfn_{MID}(p, \theta) = \frac{\theta_{(F+1)} + \theta_{(N-F)}}{2}$$

where $\theta_{(m)}$ returns the mth largest element in θ. This formulation of the convergence function is different from that used in [11]. A proof of equality between the two formulations is not needed since it is shown that this formulation satisfies the properties required by Schneider's paradigm.

[3]This condition will need to be changed to "the number of processes not working ...", where working will be a predicate analogous to the one used in [4, 5]. This is necessary for reasoning about recovery from transient failures.

This section presents informal proofs that $cfn_{MID}(p, \theta)$ satisfies the desired properties. The EHDM proofs are presented in the appendix and assume that there is a deterministic sorting algorithm which arranges the array of clock readings. This assumption will need to be discharged when the implementation is verified.

The properties presented in this section are applicable for any clock synchronization protocol which employs the fault-tolerant midpoint convergence function. All that will be required for a verified implementation is a proof that the function is correctly implemented and proofs that the other conditions have been satisfied.

4.1 Translation Invariance

Translation invariance states that the value obtained by adding x to the result of the convergence function should be the same as adding x to each of the clock readings used in evaluating the convergence function.

Condition 9 (translation invariance) *For any function θ mapping clocks to clock values,*

$$cfn(p, (\lambda n : \theta(n) + x)) = cfn(p, \theta) + x$$

Translation invariance is evident by noticing that for all m:

$$(\lambda l : \theta(l) + x)_{(m)} = \theta_{(m)} + x$$

and

$$\frac{(\theta_{(F+1)} + x) + (\theta_{(N-F)} + x)}{2} = \frac{\theta_{(F+1)} + \theta_{(N-F)}}{2} + x$$

4.2 Precision Enhancement

Precision enhancement is a formalization of the concept that, after executing the convergence function, the values of interest should be closer together.

Condition 10 (precision enhancement) *Given any subset C of the N clocks with $|C| \geq N - F$, and clocks p and q in C, then for any readings γ and θ satisfying the conditions*

1. *for any l in C, $|\gamma(l) - \theta(l)| \leq x$*
2. *for any l, m in C, $|\gamma(l) - \gamma(m)| \leq y$*
3. *for any l, m in C, $|\theta(l) - \theta(m)| \leq y$*

there is a bound $\pi(x, y)$ such that

$$|cfn(p, \gamma) - cfn(q, \theta)| \leq \pi(x, y)$$

Theorem 2 *Precision Enhancement is satisfied for $cfn_{MID}(p, \vartheta)$ if*

$$\pi(x, y) = \frac{y}{2} + x$$

One characteristic of $cfn_{MID}(p, \vartheta)$ is that it is possible for it to use readings from faulty clocks. If this occurs, we know that such readings are bounded by readings from good clocks. The next few lemmas establish this fact. To prove these lemmas it was necessary to develop a pigeon hole principle.

Lemma 1 (Pigeon Hole Principle) *If N is the number of clocks in the system, and C_1 and C_2 are subsets of these N clocks,*

$$|C_1| + |C_2| \geq N + k \supset |C_1 \cap C_2| \geq k$$

This principle greatly simplifies the existence proofs required to establish the next two lemmas. First, we establish that the values used in computing the convergence function are bounded by readings from good clocks.

Lemma 2 *Given any subset C of the N clocks with $|C| \geq N - F$ and any reading θ, there exists a $p, q \in C$ such that,*

$$\theta(p) \geq \theta_{(F+1)} \text{ (and } \theta_{(N-F)} \geq \theta(q))$$

Proof: By definition, $|\{p : \theta(p) \geq \theta_{(F+1)}\}| \geq F+1$ (similarly, $|\{q : \theta_{(N-F)} \geq \theta(q)\}| \geq F + 1$). The conclusion follows immediately from the pigeon hole principle. ∎

Now we introduce a lemma that allows us to relate values from two different readings to the same good clock.

Lemma 3 *Given any subset C of the N clocks with $|C| \geq N - F$ and readings θ and γ, there exists a $p \in C$ such that,*

$$\theta(p) \geq \theta_{(N-F)} \text{ and } \gamma_{(F+1)} \geq \gamma(p).$$

Proof: Recalling that $N \geq 3F + 1$, we can apply the pigeon hole principle twice. First to establish that $|\{p : \theta(p) \geq \theta_{(N-F)}\} \cap C| \geq F + 1$, and then to establish the conclusion. ∎

A immediate consequence of the preceding lemma is that the readings used in computing $cfn_{MID}(p, \theta)$ bound a reading from a good clock.

The next lemma introduces a useful fact for bounding the difference between good clock values from different readings.

Lemma 4 *Given any subset C of the N clocks, and clock readings θ and γ such that for any l in C, the bound $|\theta(l) - \gamma(l)| \leq x$ holds, forall $p, q \in C$,*

$$\theta(p) \geq \theta(q) \wedge \gamma(q) \geq \gamma(p) \supset |\theta(p) - \gamma(q)| \leq x$$

Proof: By cases,

- If $\theta(p) \geq \gamma(q)$, then $|\theta(p) - \gamma(q)| \leq |\theta(p) - \gamma(p)| \leq x$
- If $\theta(p) \leq \gamma(q)$, then $|\theta(p) - \gamma(q)| \leq |\theta(q) - \gamma(q)| \leq x$

∎

This enables us to establish the following lemma.

Lemma 5 *Given any subset C of the N clocks, and clock readings θ and γ such that for any l in C, the bound $|\theta(l) - \gamma(l)| \leq x$ holds, there exist $p, q \in C$ such that,*

$$\theta(p) \geq \theta_{(F+1)},$$
$$\gamma(q) \geq \gamma_{(F+1)}, \text{ and}$$
$$|\theta(p) - \gamma(q)| \leq x.$$

Proof: We know from lemma 2 that there are $p_1, q_1 \in C$ that satisfy the first two conjuncts of the conclusion. There are three cases to consider:

- If $\gamma(p_1) > \gamma(q_1)$, let $p = q = p_1$.
- If $\theta(q_1) > \theta(p_1)$, let $p = q = q_1$.
- Otherwise, we have satisfied the hypotheses for lemma 4, so we let $p = p_1$ and $q = q_1$.

■

We are now able to establish precision enhancement for $cfn_{MID}(p, \vartheta)$ (Theorem 2).
Proof: Without loss of generality, assume $cfn_{MID}(p, \gamma) \geq cfn_{MID}(q, \theta)$.

$$\begin{aligned} & |cfn_{MID}(p,\gamma) - cfn_{MID}(q,\theta)| \\ = & \left|\frac{\gamma_{(F+1)}+\gamma_{(N-F)}}{2} - \frac{\theta_{(F+1)}+\theta_{(N-F)}}{2}\right| \\ = & \frac{|\gamma_{(F+1)}+\gamma_{(N-F)}-(\theta_{(F+1)}+\theta_{(N-F)})|}{2} \end{aligned}$$

Thus we need to show that

$$|\gamma_{(F+1)} + \gamma_{(N-F)} - (\theta_{(F+1)} + \theta_{(N-F)})| \leq y + 2x$$

By choosing good clocks p, q from lemma 5, p_1 from lemma 3, and q_1 from the right conjunct of lemma 2, we establish

$$\begin{aligned} & |\gamma_{(F+1)} + \gamma_{(N-F)} - (\theta_{(F+1)} + \theta_{(N-F)})| \\ \leq & |\gamma(q) + \gamma(p_1) - \theta(p1) - \theta(q_1)| \\ = & |\gamma(q) + (\theta(p) - \theta(p)) + \gamma(p_1) - \theta(p_1) - \theta(q_1)| \\ \leq & |\theta(p) - \theta(q_1)| + |\gamma(q) - \theta(p)| + |\gamma(p_1) - \theta(p_1)| \\ \leq & y + 2x \text{ (by hypotheses and lemma 5)} \end{aligned}$$

■

4.3 Accuracy Preservation

Accuracy preservation formalizes the notion that there should be a bound on the amount of correction applied in any synchronization interval.

Condition 11 (accuracy preservation) *Given any subset C of the N clocks with $|C| \geq N - F$, and clock readings θ such that for any l and m in C, the bound $|\theta(l) - \theta(m)| \leq x$ holds, there is a bound $\alpha(x)$ such that for any q in C*

$$|cfn(p, \theta) - \theta(q)| \leq \alpha(x)$$

Theorem 3 *Accuracy preservation is satisfied for $cfn_{MID}(p, \theta)$ if $\alpha(x) = x$.*

Proof: Begin by selecting p_1 and q_1 using lemma 2. Clearly, $\theta(p_1) \geq cfn_{MID}(p, \theta)$ and $cfn_{MID}(p, \theta) \geq \theta(q_1)$. There are two cases to consider:

- If $\theta(q) \leq cfn_{MID}(p, \theta)$, then $|cfn_{MID}(p, \theta) - \theta(q)| \leq |\theta(p_1) - \theta(q)| \leq x$.
- If $\theta(q) \geq cfn_{MID}(p, \theta)$, then $|cfn_{MID}(p, \theta) - \theta(q)| \leq |\theta(q_1) - \theta(q)| \leq x$.

■

4.4 EHDM Proofs of Convergence Properties

This section presents the important details of the EHDM proofs that $cfn_{MID}(p, \theta)$ satisfies the convergence properties. In general, the proofs closely follow the presentation given above. The EHDM modules used in this effort are listed in the appendix. One underlying assumption is that $N \geq 3F + 1$. This is a well known requirement for systems to achieve Byzantine fault-tolerance without requiring authentication. Another assumption added for this effort states that the array of clock readings can be sorted. Additionally, a few properties one would expect to be true of a sorted array were assumed. These additional properties used in the EHDM proofs are (from module clocksort):

funsort_ax: **Axiom**
$i \leq j \wedge j \leq N \supset \vartheta(\mathsf{funsort}(\vartheta)(i)) \geq \vartheta(\mathsf{funsort}(\vartheta)(j))$

funsort_trans_inv: **Axiom**
$k \leq N \supset (\vartheta(\mathsf{funsort}((\lambda q : \vartheta(q) + X))(k)) = \vartheta(\mathsf{funsort}(\vartheta)(k)))$

cnt_sort_geq: **Axiom**
$k \leq N \supset \mathsf{count}((\lambda p : \vartheta(p) \geq \vartheta(\mathsf{funsort}(\vartheta)(k))), N) \geq k$

cnt_sort_leq: **Axiom**
$k \leq N \supset \mathsf{count}((\lambda p : \vartheta(\mathsf{funsort}(\vartheta)(k)) \geq \vartheta(p)), N) \geq N - k + 1$

These properties will be proven in the context of the design.

A few of the given modules are taken from Shankar's proofs [2]. These include the arithmetic modules (**absmod**, **multiplication**, and **division**), **clockassumptions**, and **countmod**. With the exception of **countmod** these modules were unaltered. A number of lemmas were added to (and proven in) module **countmod**. The most important of these is the aforementioned pigeon hole principle. In addition, lemma **count_complement** was moved from Shankar's module **ica3** to **countmod**. Shankar's complete proof was re-run after the changes to ensure that nothing was inadvertently destroyed. Future efforts will likely require additional modifications to Shankar's modules.

The induction modules, **natinduction** and **noetherian**, were taken from Rushby's transient recovery verification [6]. The standard induction schema was modified to syntactically match that used by Shankar. In addition, a lemma was added for complete induction over the natural numbers. The remaining modules were generated in the course of this verification.

The appendix contains the proof chain analysis for the three properties stated above. The proof for translation invariance is in module **mid**, precision enhancement is in **mid3**, and accuracy preservation is in **mid4**.

5 Proposed Verification

This section describes the proposed verification that the circuit correctly implements the convergence function. First an informal description of the circuit is given, and then the verification plan is discussed. This design assumes that the network of clocks is completely connected.

5.1 Informal Description

As in other synchronization algorithms, this one consists of an infinite sequence of synchronization intervals of duration $\approx R$. For the time being, we will presume the constraints listed above. It is assumed that all good clocks know the index of the current interval (a simple counter is sufficient, provided that all *good* channels start the counter in the same interval). The major concern is when to begin the next interval. For this we require readings of the other clocks in the system, and a suitable convergence function. As stated above, the selected convergence function is the fault-tolerant midpoint.

In order to execute the convergence function to start the $(i+1)$th interval clock, we need an estimate of the other processes clocks when local time is

$(i+1)R$ (according to $IC_p^i(t)$). Our estimate, Θ_p^{i+1}, of other clocks is

$$\Theta_p^{i+1}(q) = (i+1)R + (Q - LC_p^i(t_{pq}))$$

where t_{pq} is the time that p receives the signal from q, and LC is a local counter measuring elapsed time since the beginning of the current interval. All clocks participating in the protocol know to send their signal when $LC_p^i(t) = Q$. The value $(Q - LC_p^i(t_{pq}))$ gives the difference between when the local clock expected the signal and when it observed a signal from q. The reading is taken in such a way, that simply adding the value to the current time gives an estimate of the other processors clocks at that instant (modulo any effects from drift).

If the local processor p reads its clock at time t it will receive the pair $(i, LC_p^i(t))$. This reading gives the duration of time since the beginning of the protocol. The correct interpretation is $VC_p(t) = iR + LC_p^i(t)$. Thus the reading of the virtual clock just before p resets its registers for the ith interval will be $iR + cfn_{MID}(p, (\lambda q.\Theta_p^i(q) - iR))$. Notice that *translation invariance* allows the computation of the convergence function based solely on $(\lambda q.(Q - LC_p^i(t_{pq})))$.

Figure 3 presents an informal block model of the proposed clock synchronization circuit. The circuit consists of the following components:

- N pulse recognizers (only one pulse per clock is recognized in any given interval),
- a pulse counter (triggers events based upon pulse arrivals),
- a local counter LC (measures elapsed time since beginning of current interval),
- an interval counter (contains the index i of the current interval),
- one adder for computing the value $-(Q - LC_p^i(t_{pq}))$,
- one register each for storing $-\theta_{(F+1)}$ and $-\theta_{(N-F)}$,
- an adder for computing the sum of these two registers, and
- a divide-by-2 component.

The pulses are already sorted by arrival time, so it is natural to use a pulse counter to select the time-stamp of the $(F+1)$th and the $(N-F)$th pulses for the computation of the convergence function. As stated previously, all that is required is the difference between the local and remote clocks. Let $\theta = (\lambda q.\Theta_p^{i+1}(q) - (i+1)R)$. When the $F+1$st $(N-F$th$)$ signal is observed, register $-\theta_{(F+1)}$ $(-\theta_{(N-F)})$ is clocked, saving the value $-(Q - LC_p^i(t))$. After $N-F$ signals have been observed, the multiplexer selects the computed

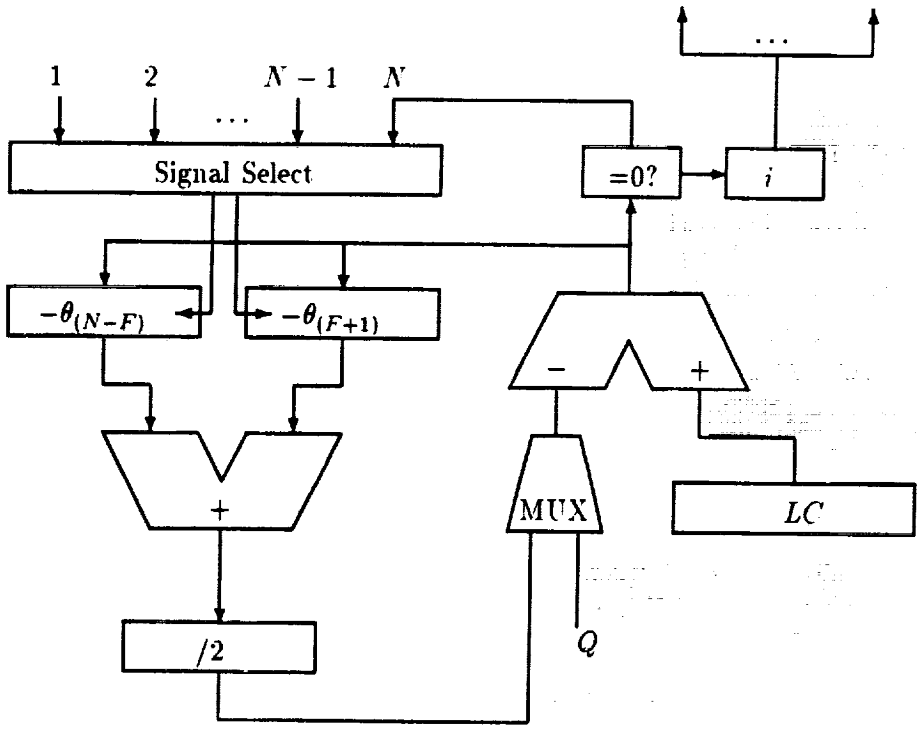

Figure 3: Informal Block Model

convergence function instead of Q. When $LC_p^i(t) - (-cfn_{MID}(p, (\theta))) = R$ it is time to begin the $i+1$st interval. To do this, all that is required is to increment i and reset LC to 0. The pulse recognizers, multiplexer select and registers are also reset at this time.

5.2 Correctness Criteria

First, the RTL description will be entered in the EHDM specification language, and then EHDM will be used to prove that RTL description correctly implements $cfn_{MID}(p, \theta)$. Each block in the informal model will be decomposed into normal hardware components such as registers, arithmetic logic units, multiplexors, and standard logic components. A functional description will be given for each device, and their composition will be shown to implement the fault-tolerant mid-point convergence function. This part of the verification will assume the properties of read error, bounded drift, and initial synchronization. Any assumptions about the convergence function used in the proofs of translation invariance, precision enhancement, or accuracy preservation need to be discharged at this level.

6 Transient Recovery

The argument for transient recovery capabilities hinges upon the following observation:

> *As long is there is power to the circuit and no faults are present, the circuit will execute the algorithm.*

Using the fact that the algorithm executes continually, and that pulses can be observed during the entire synchronization interval, we can establish that up to F transiently affected channels will automatically reintegrate themselves into the set of good channels.

We will break the discussion down into cases: First, the simple case when $F = 1$, and then the more general case for $F > 1$. Remember that $N \geq 3F + 1$. The reason two cases are considered is that only a simple modification to the hardware is required to guarantee reintegration when $F = 1$; the more general case require more inventive techniques.

6.1 Single Fault Scenario

The only modification required is that the synchronization signals include the sender's value for i (the index for the current synch interval). By virtue of the maintenance algorithm the $N-1$ good clocks are synchronized within a bounded skew $\delta \ll R$. Suppose the recovering clock observes $N-1$ pulses within $\delta + 2\Lambda$: it will chose two of these good values for computing the convergence function and a simple vote of the received interval indices will restore correct time to a lost process.

There is a possibility that the readings from the good clocks will straddle the frame boundary. The recovering clock will be ignored in the computations of the good channel, and it should adjust its own clock such that in its next interval, it will see all of the good clocks. If the window is symmetric (i.e. $Q = R/2$), it is possible that the recovering channel will compute no correction and will remain unsynchronized. However, if the window is asymmetric, a split at the boundaries will cause a recovering process to compute sufficient correction to push it into a region where it will see all the good clocks in the same interval. Thus, Q should be selected so that the window is asymmetric (i.e. $Q \neq R/2$).

6.2 General Case

When $F \geq 2$ the problem becomes more complicated. As above, if the recovering clock observes $N - F$ pulses within $\delta + 2\Lambda$, it will restore its synchrony via the convergence function and a vote of the received interval indices. However, if the good clocks straddle the boundary, the additional faulty clock(s) can prevent any adjustment from being computed on the recovering clock. It is likely that recovery cannot be guaranteed unless a timeout mechanism is added.

6.3 Comparison with Other Approaches

A number of other fault-tolerant clock synchronization protocols allow for restoration of a lost clock. The approach taken here is very similar to that proposed by Welch and Lynch [11]. They propose that when a process awakens, that it observe incoming messages until it can determine which round is underway, and then wait sufficiently long to ensure that it has seen all valid messages in that round. It can then compute the necessary correction to become synchronized. Srikanth and Toueg [12] use a similar approach, modified to the context of their algorithm. Halpern et al. [13] suggest a rather complicated protocol which requires explicit cooperation of other clocks in the system. It is more appropriate when the number of clocks in the system varies greatly over time. All of these approaches have the common theme, namely, that the joining processor knows that it wants to join. This implies the presence of some diagnostic logic or timeout mechanism which triggers the recovery process. The approach suggested here happens automatically. By virtue of the algorithm's execution in dedicated hardware, there is no need to awaken a process to participate in the protocol. The main idea is for the recovering process to converge to a state where it will observe all other clocks in the same interval, and then to restore the correct interval counter.

7 Initial Synchronization

If we can get into a state which satisfies the requirements for *precision enhancement*:

Given any subset C of the N clocks with $|C| \geq N - F$, and clocks p and q in C, then for any readings γ and θ satisfying the conditions

1. *for any l in C, $|\gamma(l) - \theta(l)| \leq x$*
2. *for any l, m in C, $|\gamma(l) - \gamma(m)| \leq y$*
3. *for any l, m in C, $|\theta(l) - \theta(m)| \leq y$*

there is a bound $\pi(x, y)$ such that

$$|cfn(p, \gamma) - cfn(q, \theta)| \leq \pi(x, y)$$

where $y = R/2$ and x is the normal value ($\approx 2\Lambda$), the above circuit will converge to within δ_S in approximately $\log_2(R/2)$ intervals. Byzantine agreement will then be required to establish a consistent interval counter. It will be necessary to ensure that the clocks converge to a state satisfying the above constraints.

7.1 Mechanisms for Initialization

In order to ensure that we reach a state which satisfies the above requirements, it is necessary to identify possible states which violate the above requirements. Such states would happen due to the behavior of clocks prior to the time that enough good clocks are running. In previous cases we knew we had a set C of good clocks with $|C| \geq N - F$. This means that there were a sufficient number of clock readings to resolve $\theta_{(F+1)}$ and $\theta_{(N-F)}$. This may not be the case during initialization. We need to determine a course of action when we do not observe $N - F$ clocks. Two plausible options are to

1. pretend all clocks are observed to be in perfect synchrony, or
2. pretend that unobserved clocks are observed at the end of the interval (i.e. $(LC_p^i(t_{pq}) - Q) = (R - Q)$). Compute the correction based upon this value.

Both options will be explored. The first option is simple to implement because no correction is necessary. When $LC = R$, set both i and LC to 0, and reset the circuit for the next interval. To implement the second option, perform the following action when $LC = R$: if fewer than $N - F$ $(F + 1)$ signals are observed, then enable register $-\theta_{(N-F)}$ $(-\theta_{(F+1)})$. This will cause the unobserved readings to be $(R - Q)$ which is equivalent to observing the pulse at the end of an interval of duration R.

It will be necessary to define a *convergence stair* (ala [15]) for scenarios that don't converge by default.

7.2 Comparison to Other Approaches

Most of the comments concerning the approach to transient recovery are applicable here as well. This approach for achieving initial synchronization differs from most methods in that it first synchronizes the interval clocks, and then it decides upon a value for the current interval. Techniques in [11], [12], and [13] all depend upon the good clocks knowing that they wish to initialize. Agreement is reached among the clocks wishing to join, and then the protocol begins. The approach taken here seems closer to that used in [14], however, details of their approach are not given.

8 Concluding Remarks

Clock synchronization provides the cornerstone of any fault-tolerant computer architecture. To avoid a single point failure it is imperative that each processor maintain a local clock which is periodically resynchronized with other clocks in a fault-tolerant manner. Due to subtleties involved in reasoning about interactions involving misbehaving components, it is necessary to prove that the clock synchronization function operates correctly. Shankar

[2] provides a mechanical proof (using EHDM [3]) that Schneider's generalized protocol [1] achieves Byzantine fault-tolerant clock synchronization, provided that eleven constraints are satisfied. Shankar's work provides the formal specification of the proposed verified design.

The fault-tolerant midpoint convergence function has been proven (in EHDM) to satisfy the properties of translation invariance, precision enhancement, and accuracy preservation. These proofs are reusable in the verification of any synchronization algorithm which uses the same function. An informal design of a circuit to implement this function has been presented. Future efforts will focus on formalizing this design and proving the additional required properties from it. A register transfer level description of the design will be expressed in the specification language of EHDM, and proven to correctly implement the fault-tolerant midpoint function. Other properties to be proven from the design include bounded interval, bounded delay, initial synchronization, non-overlap, and any assumptions made in establishing the properties of the convergence function. Bounded drift is a physical property of the oscillator and cannot be established formally. The value for drift will be taken from the oscillator's stated performance parameters. It is assumed that the number of faults F is less than $N/3$, where N denotes the number of clocks in the system. Read error will be assumed in this development, but there is ongoing work at SRI to prove that remote clocks can be read with bounded error. An approach for bounding initial skew will be verified for the single fault scenario and a more general solution will be explored.

In keeping with the spirit of the Reliable Computing Platform, it is imperative that the clock synchronization subsystem provide for recovery from transient faults. This paper has argued that the proposed design will recover from a single transient fault. This argument will be formalized in EHDM using an approach similar to that used by DiVito, Butler, and Caldwell for the RCP [4]. Extensions to accommodate the more general case will be developed, but would likely involve modifications to the design. An interesting feature of this design is that for the single fault case (i.e. 4, 5, or 6 clocks), the properties of transient recovery and initial synchronization occur automatically. The clock system will recover without explicitly recognizing that something is amiss. The system can be augmented to recognize loss of synchrony due to a transient fault, but need not do so for recovery purposes.

A Proof Summary

Notice that the only modules with failed proofs have the suffix _tcc. These modules are automatically generated by EHDM, and cannot be altered by the user. When a proof fails the user must prove the type check constraint elsewhere. The proof chain analysis (Appendix C) ensures that these obligations have been discharged.

```
Proof summaries for modules on using chain of module mid_top

Module mid4_tcc:            1 successful proof,   1 failure,   0 errors
Module mid3_tcc:            8 successful proofs,  5 failures,  0 errors
Module mid2_tcc:            2 successful proofs,  2 failures,  0 errors
Module mid_tcc:             2 successful proofs,  1 failure,   0 errors
Module tcc_mid:             9 successful proofs,  0 failures,  0 errors
Module division_tcc:        7 successful proofs,  0 failures,  0 errors
Module natinduction_tcc:    1 successful proof,   0 failures,  0 errors
Module countmod_tcc:        3 successful proofs,  3 failures,  0 errors
Module ft_mid_assume:      no proofs
Module clocksort:          no proofs
Module select_defs:         6 successful proofs,  0 failures,  0 errors
Module mid:                 2 successful proofs,  0 failures,  0 errors
Module mid2:                2 successful proofs,  0 failures,  0 errors
Module mid3:                9 successful proofs,  0 failures,  0 errors
Module noetherian:          1 successful proof,   0 failures,  0 errors
Module natinduction:        5 successful proofs,  0 failures,  0 errors
Module countmod:           30 successful proofs,  0 failures,  0 errors
Module clockassumptions:    9 successful proofs,  0 failures,  0 errors
Module absmod:             15 successful proofs,  0 failures,  0 errors
Module division:           11 successful proofs,  0 failures,  0 errors
Module multiplication:     11 successful proofs,  0 failures,  0 errors
Module arith:              no proofs
Module mid4:                9 successful proofs,  0 failures,  0 errors
Module mid_top:             3 successful proofs,  0 failures,  0 errors

Totals:                    146 successful proofs,  12 failures,   0 errors


Total time: 715 seconds.
```

B LaTeX printed EHDM Modules

mid_top: **Module**

Using mid4, countmod_tcc, natinduction_tcc, division_tcc,
  tcc_mid

Theory

Proof

  posint_TCC1_PROOF: **Prove** posint_TCC1 $\{i_1 - 1\}$

  countmod_TCC4_pr: **Prove** count_TCC4 **from**
    countsize,
    countsize $\{i$ — (**if** $i > 0$ **then** $i - 1$ **else** i **end if**)$\}$

  countmod_TCC5_pr: **Prove** count_TCC5 **from**
    countsize,
    countsize $\{i$ — (**if** $i > 0$ **then** $i - 1$ **else** i **end if**)$\}$

End mid_top

countmod_tcc: **Module**

Using countmod

Exporting all with countmod

Theory

i_1: **Var** integer
ppred: **Var** function[naturalnumber — boolean]
i: **Var** naturalnumber
p: **Var** naturalnumber
k: **Var** naturalnumber
n: **Var** naturalnumber
d_1: **Var** nk_type
nk: **Var** nk_type
nk2: **Var** nk_type
j: **Var** naturalnumber
posint_TCC1: **Formula** $(\exists i_1 : i_1 > 0)$

count_TCC1: **Formula** $(i > 0) \supset (i - 1 \geq 0)$

count_TCC2: **Formula** $(\text{ppred}(i - 1)) \wedge (i > 0) \supset (i - 1 \geq 0)$

count_TCC3: **Formula** $(\neg(\text{ppred}(i - 1))) \wedge (i > 0) \supset (i - 1 \geq 0)$

count_TCC4: **Formula**
$(\text{ppred}(i - 1)) \wedge (i > 0)$
$\supset \text{countsize}(\text{ppred}, i) > \text{countsize}(\text{ppred}, i - 1)$

count_TCC5: **Formula**
$(\neg(\text{ppred}(i - 1))) \wedge (i > 0)$
$\supset \text{countsize}(\text{ppred}, i) > \text{countsize}(\text{ppred}, i - 1)$

Proof

posint_TCC1_PROOF: **Prove** posint_TCC1

count_TCC1_PROOF: **Prove** count_TCC1

count_TCC2_PROOF: **Prove** count_TCC2

count_TCC3_PROOF: **Prove** count_TCC3

count_TCC4_PROOF: **Prove** count_TCC4

count_TCC5_PROOF: **Prove** count_TCC5

End countmod_tcc

natinduction_tcc: **Module**

Using natinduction

Exporting all with natinduction

Theory

m: **Var** naturalnumber
n: **Var** naturalnumber
i: **Var** naturalnumber
j: **Var** naturalnumber
d_1: **Var** naturalnumber
ind_m_proof_TCC1: **Formula**
(**if** $n \geq m$ **then** $n - m$ **else** 0 **end if** ≥ 0)

Proof

ind_m_proof_TCC1_PROOF: **Prove** ind_m_proof_TCC1

End natinduction_tcc

division_tcc: **Module**

Using division

Exporting all with division

Theory

x: **Var** number
y: **Var** number
z: **Var** number
mult_div_1_TCC1: **Formula** $(z \neq 0) \supset (z \neq 0)$

mult_div_TCC1: **Formula** $(y \neq 0) \supset (y \neq 0)$

div_cancel_TCC1: **Formula** $(x \neq 0) \supset (x \neq 0)$

ceil_mult_div_TCC1: **Formula** $(y > 0) \supset (y \neq 0)$

div_nonnegative_TCC1: **Formula** $(x \geq 0 \wedge y > 0) \supset (y \neq 0)$

div_ineq_TCC1: **Formula** $(z > 0 \wedge x \leq y) \supset (z \neq 0)$

div_minus_1_TCC1: **Formula** $(y > 0 \wedge x < 0) \supset (y \neq 0)$

Proof

mult_div_1_TCC1_PROOF: **Prove** mult_div_1_TCC1

mult_div_TCC1_PROOF: **Prove** mult_div_TCC1

div_cancel_TCC1_PROOF: **Prove** div_cancel_TCC1

ceil_mult_div_TCC1_PROOF: **Prove** ceil_mult_div_TCC1

div_nonnegative_TCC1_PROOF: **Prove** div_nonnegative_TCC1

div_ineq_TCC1_PROOF: **Prove** div_ineq_TCC1

div_minus_1_TCC1_PROOF: **Prove** div_minus_1_TCC1

End division_tcc

```
tcc_mid: Module

Using mid_tcc, mid2_tcc, mid3_tcc, mid4_tcc

Theory

Proof

  ft_mid_TCC2_PROOF: Prove ft_mid_TCC2 from ft_mid_maxfaults

  good_less_NF_TCC1_PROOF: Prove good_less_NF_TCC1 from
    ft_mid_maxfaults

  good_less_NF_pr_TCC1_PROOF: Prove good_less_NF_pr_TCC1 from
    ft_mid_maxfaults

  good_between_TCC1_PROOF: Prove good_between_TCC1 from
    ft_mid_maxfaults

  ft_mid_prec_sym1_TCC2_PROOF: Prove ft_mid_prec_sym1_TCC2 from
    ft_mid_maxfaults

  ft_mid_prec_sym1_TCC4_PROOF: Prove ft_mid_prec_sym1_TCC4 from
    ft_mid_maxfaults

  mid_gt_imp_sel_gt_TCC2_PROOF: Prove mid_gt_imp_sel_gt_TCC2 from
    ft_mid_maxfaults

  ft_mid_prec_sym1_pr_TCC2_PROOF: Prove ft_mid_prec_sym1_pr_TCC2
    from ft_mid_maxfaults

  ft_mid_greater_TCC1_PROOF: Prove ft_mid_greater_TCC1 from
    ft_mid_maxfaults

End tcc_mid
```

absmod: **Module**

Using multiplication

Exporting all

Theory

$x, y, z, x_1, y_1, z_1, x_2, y_2, z_2$: **Var** number
$|\star 1|$: **Definition** function[number → number] =
  $(\lambda x : ($ **if** $x < 0$ **then** $-x$ **else** x **end if**$))$

abs_main: **Lemma** $|x| < z \supset (x < z \vee -x < z)$

abs_leq_0: **Lemma** $|x - y| \leq z \supset (x - y) \leq z$

abs_diff: **Lemma** $|x - y| < z \supset ((x - y) < z \vee (y - x) < z)$

abs_leq: **Lemma** $|x| \leq z \supset (x \leq z \vee -x \leq z)$

abs_bnd: **Lemma**
  $0 \leq z \wedge 0 \leq x \wedge x \leq z \wedge 0 \leq y \wedge y \leq z \supset |x - y| \leq z$

abs_1_bnd: **Lemma** $|x - y| \leq z \supset x \leq y + z$

abs_2_bnd: **Lemma** $|x - y| \leq z \supset x \geq y - z$

abs_3_bnd: **Lemma** $x \leq y + z \wedge x \geq y - z \supset |x - y| \leq z$

abs_drift: **Lemma**
  $|x - y| \leq z \wedge |x_1 - x| \leq z_1 \supset |x_1 - y| \leq z + z_1$

abs_com: **Lemma** $|x - y| = |y - x|$

abs_drift_2: **Lemma**
  $|x - y| \leq z \wedge |x_1 - x| \leq z_1 \wedge |y_1 - y| \leq z_2$
    $\supset |x_1 - y_1| \leq z + z_1 + z_2$

abs_geq: **Lemma** $x \geq y \wedge y \geq 0 \supset |x| \geq |y|$

abs_ge0: **Lemma** $x \geq 0 \supset |x| = x$

abs_plus: **Lemma** $|x + y| \leq |x| + |y|$

abs_diff_3: **Lemma** $x - y \le z \wedge y - x \le z \supset |x - y| \le z$

Proof

abs_plus_pr: **Prove** abs_plus **from**
  $|\star 1|$ $\{x \leftarrow x + y\}$, $|\star 1|$, $|\star 1|$ $\{x \leftarrow y\}$

abs_diff_3_pr: **Prove** abs_diff_3 **from** $|\star 1|$ $\{x \leftarrow x - y\}$

abs_ge0_proof: **Prove** abs_ge0 **from** $|\star 1|$

abs_geq_proof: **Prove** abs_geq **from** $|\star 1|$, $|\star 1|$ $\{x \leftarrow y\}$

abs_drift_2_proof: **Prove** abs_drift_2 **from**
  abs_drift,
  abs_drift
    $\{x \leftarrow y,$
    $y \leftarrow y_1,$
    $z \leftarrow z_2,$
    $z_1 \leftarrow z + z_1\}$,
  abs_com $\{x \leftarrow y_1\}$

abs_com_proof: **Prove** abs_com **from**
  $|\star 1|$ $\{x \leftarrow (x - y)\}$, $|\star 1|$ $\{x \leftarrow (y - x)\}$

abs_drift_proof: **Prove** abs_drift **from**
  abs_1_bnd,
  abs_1_bnd $\{x \leftarrow x_1,\ y \leftarrow x,\ z \leftarrow z_1\}$,
  abs_2_bnd,
  abs_2_bnd $\{x \leftarrow x_1,\ y \leftarrow x,\ z \leftarrow z_1\}$,
  abs_3_bnd $\{x \leftarrow x_1,\ z \leftarrow z + z_1\}$

abs_3_bnd_proof: **Prove** abs_3_bnd **from** $|\star 1|$ $\{x \leftarrow (x - y)\}$

abs_main_proof: **Prove** abs_main **from** $|\star 1|$

abs_leq_0_proof: **Prove** abs_leq_0 **from** $|\star 1|$ $\{x \leftarrow x - y\}$

abs_diff_proof: **Prove** abs_diff **from** $|\star 1|$ $\{x \leftarrow (x - y)\}$

abs_leq_proof: **Prove** abs_leq **from** $|\star 1|$

abs_bnd_proof: **Prove** abs_bnd **from** $| \star 1 | \; \{x \leftarrow (x - y)\}$

abs_1_bnd_proof: **Prove** abs_1_bnd **from** $| \star 1 | \; \{x \leftarrow (x - y)\}$

abs_2_bnd_proof: **Prove** abs_2_bnd **from** $| \star 1 | \; \{x \leftarrow (x - y)\}$

End absmod

multiplication: **Module**

Exporting all

Theory

$x, y, z, x_1, y_1, z_1, x_2, y_2, z_2$: **Var** number
$\star 1 \star \star 2$: function[number, number — number] = ($\lambda\, x, y : (x \star y)$)

mult_ldistrib: **Lemma** $x \star (y + z) = x \star y + x \star z$

mult_ldistrib_minus: **Lemma** $x \star (y - z) = x \star y - x \star z$

mult_rident: **Lemma** $x \star 1 = x$

mult_lident: **Lemma** $1 \star x = x$

distrib: **Lemma** $(x + y) \star z = x \star z + y \star z$

distrib_minus: **Lemma** $(x - y) \star z = x \star z - y \star z$

mult_non_neg: **Axiom**
$((x \geq 0 \wedge y \geq 0) \vee (x \leq 0 \wedge y \leq 0)) \Leftrightarrow x \star y \geq 0$

mult_pos: **Axiom** $((x > 0 \wedge y > 0) \vee (x < 0 \wedge y < 0)) \Leftrightarrow x \star y > 0$

mult_com: **Lemma** $x \star y = y \star x$

pos_product: **Lemma** $x \geq 0 \wedge y \geq 0 \supset x \star y \geq 0$

mult_leq: **Lemma** $z \geq 0 \wedge x \geq y \supset x \star z \geq y \star z$

mult_leq_2: **Lemma** $z \geq 0 \wedge x \geq y \supset z \star x \geq z \star y$

mult_l0: **Axiom** $0 \star x = 0$

mult_gt: **Lemma** $z > 0 \wedge x > y \supset x \star z > y \star z$

Proof

mult_gt_pr: **Prove** mult_gt **from**
mult_pos $\{x \leftarrow x - y,\ y \leftarrow z\}$, distrib_minus

distrib_minus_pr: **Prove** distrib_minus **from**
mult_ldistrib_minus $\{x — z,\ y — x,\ z — y\}$,
mult_com $\{x — x - y,\ y — z\}$,
mult_com $\{y — z\}$,
mult_com $\{x — y,\ y — z\}$

mult_leq_2_pr: **Prove** mult_leq_2 **from**
mult_ldistrib_minus $\{x — z,\ y — x,\ z — y\}$,
mult_non_neg $\{x — z,\ y — x - y\}$

mult_leq_pr: **Prove** mult_leq **from**
distrib_minus, mult_non_neg $\{x — x - y,\ y — z\}$

mult_com_pr: **Prove** mult_com **from**
$\star 1 \star \star 2$, $\star 1 \star \star 2$ $\{x — y,\ y — x\}$

pos_product_pr: **Prove** pos_product **from** mult_non_neg

mult_rident_proof: **Prove** mult_rident **from** $\star 1 \star \star 2$ $\{y — 1\}$

mult_lident_proof: **Prove** mult_lident **from**
$\star 1 \star \star 2$ $\{x — 1,\ y — x\}$

distrib_proof: **Prove** distrib **from**
$\star 1 \star \star 2$ $\{x — x + y,\ y — z\}$,
$\star 1 \star \star 2$ $\{y — z\}$,
$\star 1 \star \star 2$ $\{x — y,\ y — z\}$

mult_ldistrib_proof: **Prove** mult_ldistrib **from**
$\star 1 \star \star 2$ $\{y — y + z,\ x — x\}$, $\star 1 \star \star 2$, $\star 1 \star \star 2$ $\{y — z\}$

mult_ldistrib_minus_proof: **Prove** mult_ldistrib_minus **from**
$\star 1 \star \star 2$ $\{y — y - z,\ x — x\}$, $\star 1 \star \star 2$, $\star 1 \star \star 2$ $\{y — z\}$

End multiplication

noetherian: **Module** [dom: **Type**, <: function[dom, dom — bool]]

Assuming

measure: **Var** function[dom — nat]
a, b: **Var** dom

well_founded: **Formula**
($\exists$ measure : $a < b \supset$ measure(a) < measure(b))

Theory

p, A, B: **Var** function[dom — bool]
d, d_1, d_2: **Var** dom

general_induction: **Axiom**
$(\forall d_1 : (\forall d_2 : d_2 < d_1 \supset p(d_2)) \supset p(d_1)) \supset (\forall d : p(d))$

d_3, d_4: **Var** dom

mod_induction: **Theorem**
$(\forall d_3, d_4 : d_4 < d_3 \supset A(d_3) \supset A(d_4))$
$\quad \wedge (\forall d_1 : (\forall d_2 : d_2 < d_1 \supset (A(d_1) \wedge B(d_2))) \supset B(d_1))$
$\quad \supset (\forall d : A(d) \supset B(d))$

Proof

mod_proof: **Prove**
mod_induction $\{d_1 — d_1@p1,\ d_3 — d_1@p1,\ d_4 — d_2\}$
from general_induction $\{p — (\lambda d : A(d) \supset B(d))\}$

End noetherian

select_defs: **Module**

Using arith, countmod, clockassumptions, clocksort

Exporting all with clockassumptions

Theory

process: **Type is** nat
Clocktime: **Type is** number
l, m, n, p, q: **Var** process
ϑ: **Var** function[process → Clocktime]
i, j, k: **Var** posint
T, X, Y, Z: **Var** Clocktime
$\star 1_{(\star 2)}$: function[function[process → Clocktime], posint
→ Clocktime] == $(\lambda\, \vartheta, i : \vartheta(\text{funsort}(\vartheta)(i)))$

select_trans_inv: **Lemma**
$k \leq N \supset (\lambda\, q : \vartheta(q) + X)_{(k)} = \vartheta_{(k)} + X$

select_exists1: **Lemma** $i \leq N \supset (\exists\, p : p < N \wedge \vartheta(p) = \vartheta_{(i)})$

select_exists2: **Lemma** $p < N \supset (\exists\, i : i \leq N \wedge \vartheta(p) = \vartheta_{(i)})$

select_ax: **Lemma** $1 \leq i \wedge i < k \wedge k \leq N \supset \vartheta_{(i)} \geq \vartheta_{(k)}$

count_geq_select: **Lemma**
$k \leq N \supset \text{count}((\lambda\, p : \vartheta(p) \geq \vartheta_{(k)}), N) \geq k$

count_leq_select: **Lemma**
$k \leq N \supset \text{count}((\lambda\, p : \vartheta_{(k)} \geq \vartheta(p)), N) \geq N - k + 1$

Proof

select_trans_inv_pr: **Prove** select_trans_inv **from**
funsort_trans_inv

select_exists1_pr: **Prove** select_exists1 $\{p \leftarrow \text{funsort}(\vartheta)(i)\}$
from funsort_fun_1_1 $\{j \leftarrow i\}$

select_exists2_pr: **Prove** select_exists2 $\{i \leftarrow i@p1\}$ **from**
funsort_fun_onto

select_ax_pr: **Prove** select_ax **from**
funsort_ax $\{i — i@c,\ j — k@c\}$

count_leq_select_pr: **Prove** count_leq_select **from** cnt_sort_leq

count_geq_select_pr: **Prove** count_geq_select **from** cnt_sort_geq

End select_defs

ft_mid_assume: **Module**

Using clockassumptions

Exporting all with clockassumptions

Theory

  ft_mid_maxfaults: **Axiom** $N \geq 3 * F + 1$

End ft_mid_assume

arith: **Module**

Using multiplication. division. absmod

Exporting all with multiplication, division. absmod

End arith

clocksort: **Module**

Using clockassumptions

Exporting all with clockassumptions

Theory

l, m, n, p, q: **Var** process
i, j, k: **Var** posint
X, Y: **Var** Clocktime
ϑ: **Var** function[process — Clocktime]
funsort: function[function[process — Clocktime]
— function[posint — process]]

funsort_ax: **Axiom**
$i \leq j \wedge j \leq N \supset \vartheta(\text{funsort}(\vartheta)(i)) \geq \vartheta(\text{funsort}(\vartheta)(j))$

funsort_fun_1_1: **Axiom**
$i \leq N \wedge j \leq N \wedge \text{funsort}(\vartheta)(i) = \text{funsort}(\vartheta)(j)$
$\supset i = j \wedge \text{funsort}(\vartheta)(i) < N$

funsort_fun_onto: **Axiom**
$p < N \supset (\exists i : i \leq N \wedge \text{funsort}(\vartheta)(i) = p)$

funsort_trans_inv: **Axiom**
$k \leq N \supset (\vartheta(\text{funsort}((\lambda q : \vartheta(q) + X))(k)) = \vartheta(\text{funsort}(\vartheta)(k)))$

cnt_sort_geq: **Axiom**
$k \leq N \supset \text{count}((\lambda p : \vartheta(p) \geq \vartheta(\text{funsort}(\vartheta)(k))), N) \geq k$

cnt_sort_leq: **Axiom**
$k \leq N \supset \text{count}((\lambda p : \vartheta(\text{funsort}(\vartheta)(k)) \geq \vartheta(p)), N) \geq N - k + 1$

Proof

End clocksort

clockassumptions: **Module**

Using arith. countmod

Exporting all with countmod. arith

Theory

N: nat

N_0: **Axiom** $N > 0$

process: **Type is nat**
event: **Type is nat**
time: **Type is number**
Clocktime: **Type is number**
$l, m, n, p, q, p_1, p_2, q_1, q_2, p_3, q_3$: **Var** process
i, j, k: **Var** event
x, y, z, r, s, t: **Var** time
X, Y, Z, R, S, T: **Var** Clocktime
γ, θ: **Var** function[process — Clocktime]
$\delta, \mu, \rho, r_{min}, r_{max}, \beta, \Lambda$: number
$PC_{\star 1}(\star 2), VC_{\star 1}(\star 2)$: function[process. time — Clocktime]
$t_{\star 1}^{\star 2}$: function[process, event — time]
$\Theta_{\star 1}^{\star 2}$: function[process, event
— function[process — Clocktime]]
$IC_{\star 1}^{\star 2}(\star 3)$: function[process, event, time — Clocktime]
correct: function[process, time — bool]
em cfn: function[process, function[process — Clocktime]
— Clocktime]
π: function[Clocktime, Clocktime — Clocktime]
α: function[Clocktime — Clocktime]

delta_0: **Axiom** $\delta \geq 0$

mu_0: **Axiom** $\mu \geq 0$

rho_0: **Axiom** $\rho \geq 0$

rho_1: **Axiom** $\rho < 1$

rmin_0: **Axiom** $r_{min} > 0$

rmax_0: **Axiom** $r_{max} > 0$

beta_0: **Axiom** $\beta \geq 0$

lamb_0: **Axiom** $\Lambda \geq 0$

init: **Axiom** correct$(p, 0) \supset PC_p(0) \geq 0 \wedge PC_p(0) \leq \mu$

correct_closed: **Axiom** $s \geq t \wedge$ correct$(p, s) \supset$ correct(p, t)

rate_1: **Axiom**
correct$(p, s) \wedge s \geq t \supset PC_p(s) - PC_p(t) \leq (s - t) \star (1 + \rho)$

rate_2: **Axiom**
correct$(p, s) \wedge s \geq t \supset PC_p(s) - PC_p(t) \geq (s - t) \star (1 - \rho)$

rts0: **Axiom** correct$(p, t) \wedge t \leq t_p^{i+1} \supset t - t_p^i \leq r_{max}$

rts1: **Axiom** correct$(p, t) \wedge t \geq t_p^{i+1} \supset t - t_p^i \geq r_{min}$

rts_0: **Lemma** correct$(p, t_p^{i+1}) \supset t_p^{i+1} - t_p^i \leq r_{max}$

rts_1: **Lemma** correct$(p, t_p^{i+1}) \supset t_p^{i+1} - t_p^i \geq r_{min}$

rts2: **Axiom**
correct$(p, t) \wedge t \geq t_q^i + \beta \wedge$ correct$(q, t) \supset t \geq t_p^i$

rts_2: **Axiom**
correct$(p, t_p^i) \wedge$ correct$(q, t_q^i) \supset t_p^i - t_q^i \leq \beta$

synctime_0: **Axiom** $t_p^0 = 0$

VClock_defn: **Axiom**
correct$(p, t) \wedge t \geq t_p^i \wedge t < t_p^{i+1} \supset VC_p(t) = IC_p^i(t)$

Adj: function[process, event — Clocktime] =
($\lambda\, p, i$:
(**if** $i > 0$ **then** $cfn(p, \Theta_p^i) - PC_p(t_p^i)$ **else** 0 **end if**))

IClock_defn: **Axiom** correct$(p, t) \supset IC_p^i(t) = PC_p(t) +$ Adj(p, i)

Readerror: **Axiom**
correct(p, t_p^{i+1}) $\wedge$ correct(q, t_p^{i+1})
$\supset |\Theta_p^{i+1} q) - IC_q^i(t_p^{i+1})| \leq \Lambda$

translation_invariance: **Axiom**
$X \geq 0 \supset cfn(p, (\lambda p_1$ — Clocktime : $\gamma(p_1) + X)) = cfn(p, \gamma) + X$

ppred: **Var** function[process — bool]
F: process
okay_Readpred: function[function[process — Clocktime],
Clocktime, function[process — bool]
— bool] =
($\lambda \gamma, Y$, ppred :
($\forall l, m$: ppred(l) $\wedge$ ppred(m) $\supset |\gamma(l) - \gamma(m)| \leq Y$))
okay_pairs: function[function[process — Clocktime],
function[process — Clocktime], Clocktime,
function[process — bool] — bool] =
($\lambda \gamma, \theta, X$, ppred :
($\forall p_3$: ppred(p_3) $\supset |\gamma(p_3) - \theta(p_3)| \leq X$))

N_maxfaults: **Axiom** $F \leq N$

precision_enhancement_ax: **Axiom**
count(ppred, N) $\geq N - F$
$\wedge$ okay_Readpred(γ, Y, ppred)
$\wedge$ okay_Readpred(θ, Y, ppred)
$\wedge$ okay_pairs(γ, θ, X, ppred) $\wedge$ ppred(p) $\wedge$ ppred(q)
$\supset |cfn(p, \gamma) - cfn(q, \theta)| \leq \pi(X, Y)$

correct_count: **Axiom** count((λp : correct(p, t)), N) $\geq N - F$

```
okay_Reading: function[function[process → Clocktime],
                                    Clocktime, time → bool] =
  ( λ γ, Y, t :
      ( ∀ p_1, q_1 :
         correct(p_1, t) ∧ correct(q_1, t) ⊃ |γ(p_1) − γ(q_1)| ≤ Y))
okay_Readvars: function[function[process → Clocktime],
                                    function[process → Clocktime],
                                    Clocktime, Clocktime → bool] =
  ( λ γ, θ, X, t :
      ( ∀ p_3 : correct(p_3, t) ⊃ |γ(p_3) − θ(p_3)| ≤ X))

okay_Readpred_Reading: Lemma
  okay_Reading(γ, Y, t)
    ⊃ okay_Readpred(γ, Y, ( λ p : correct(p, t)))

okay_pairs_Readvars: Lemma
  okay_Readvars(γ, θ, X, t)
    ⊃ okay_pairs(γ, θ, X, ( λ p : correct(p, t)))

precision_enhancement: Lemma
  okay_Reading(γ, Y, t_p^{i+1})
        ∧ okay_Reading(θ, Y, t_p^{i+1})
          ∧ okay_Readvars(γ, θ, X, t_p^{i+1})
            ∧ correct(p, t_p^{i+1}) ∧ correct(q, t_p^{i+1})
    ⊃ |cfn(p, γ) − cfn(q, θ)| ≤ π(X, Y)

okay_Reading_defn_lr: Lemma
  okay_Reading(γ, Y, t)
    ⊃ ( ∀ p_1, q_1 :
          correct(p_1, t) ∧ correct(q_1, t) ⊃ |γ(p_1) − γ(q_1)| ≤ Y)

okay_Reading_defn_rl: Lemma
  ( ∀ p_1, q_1 :
          correct(p_1, t) ∧ correct(q_1, t) ⊃ |γ(p_1) − γ(q_1)| ≤ Y)
    ⊃ okay_Reading(γ, Y, t)

okay_Readvars_defn_lr: Lemma
  okay_Readvars(γ, θ, X, t)
    ⊃ ( ∀ p_3 : correct(p_3, t) ⊃ |γ(p_3) − θ(p_3)| ≤ X)
```

okay_Readvars_defn_rl: **Lemma**
($\forall p_3$: correct$(p_3, t) \supset |\gamma(p_3) - \theta(p_3)| \leq X$)
$\supset$ okay_Readvars(γ, θ, X, t)

accuracy_preservation_ax: **Axiom**
okay_Readpred$(\gamma, X,$ ppred$)$
$\wedge$ count(ppred, N) $\geq N - F \wedge$ ppred$(p) \wedge$ ppred(q)
$\supset |cfn(p, \gamma) - \gamma(q)| \leq \alpha(X)$

Proof

okay_Reading_defn_rl_pr: **Prove**
okay_Reading_defn_rl $\{p_1 \leftarrow p_1$@P1S, $q_1 \leftarrow q_1$@P1S$\}$ **from**
okay_Reading

okay_Reading_defn_lr_pr: **Prove** okay_Reading_defn_lr **from**
okay_Reading $\{p_1 \leftarrow p_1$@CS, $q_1 \leftarrow q_1$@CS$\}$

okay_Readvars_defn_rl_pr: **Prove**
okay_Readvars_defn_rl $\{p_3 \leftarrow p_3$@P1S$\}$ **from** okay_Readvars

okay_Readvars_defn_lr_pr: **Prove** okay_Readvars_defn_lr **from**
okay_Readvars $\{p_3 \leftarrow p_3$@CS$\}$

precision_enhancement_pr: **Prove** precision_enhancement **from**
precision_enhancement_ax
{ppred $\leftarrow$ (λq : correct(q, t_p^{i+1}))},
okay_Readpred_Reading $\{t \leftarrow t_p^{i+1}\}$,
okay_Readpred_Reading $\{t \leftarrow t_p^{i+1}, \gamma \leftarrow \theta\}$,
okay_pairs_Readvars $\{t \leftarrow t_p^{i+1}\}$,
correct_count $\{t \leftarrow t_p^{i+1}\}$

okay_Readpred_Reading_pr: **Prove** okay_Readpred_Reading **from**
okay_Readpred {ppred $\leftarrow$ (λp : correct(p, t))},
okay_Reading $\{p_1 \leftarrow l@P1S, q_1 \leftarrow m@P1S\}$

okay_pairs_Readvars_pr: **Prove** okay_pairs_Readvars **from**
okay_pairs {ppred $\leftarrow$ (λp : correct(p, t))},
okay_Readvars $\{p_3 \leftarrow p_3$@P1S$\}$

rts_0_proof: **Prove** rts_0 **from** rts0 $\{t \leftarrow t_p^{i+1}\}$

rts_1_proof: **Prove** rts_1 **from** rts1 $\{t \leftarrow t_p^{i+1}\}$

End clockassumptions

countmod: **Module**

Exporting all

Theory

```
  $i_1$: Var int
  posint: Type from nat with ( $\lambda\, i_1 : i_1 > 0$)
  $l, m, n, p, q, p_1, p_2, q_1, q_2, p_3, q_3$: Var nat
  $i, j, k$: Var nat
  $x, y, z, r, s, t$: Var number
  $X, Y, Z$: Var number
  ppred, ppred1, ppred2: Var function[nat — bool]
  $\vartheta, \theta, \gamma$: Var function[nat — number]
  countsize: function[function[nat — bool], nat — nat] =
     ( $\lambda$ ppred, $i$ : $i$)
  count: Recursive function[function[nat — bool], nat — nat] =
     ( $\lambda$ ppred, $i$ :
          ( if $i > 0$
               then ( if ppred($i - 1$)
                        then 1 + (count(ppred, $i - 1$))
                        else count(ppred, $i - 1$)
                        end if)
               else 0
               end if))
    by countsize
  count_complement: Lemma
    count(( $\lambda\, q$ : $\neg$ppred($q$)), $n$) = $n$ − count(ppred, $n$)

  count_exists: Lemma
    count(ppred, $n$) > 0 $\supset$ ( $\exists\, p : p < n \wedge$ ppred($p$))

  count_true: Lemma count(( $\lambda\, p$ : true), $n$) = $n$

  count_false: Lemma count(( $\lambda\, p$ : false), $n$) = 0

  count_bounded_imp: Lemma
    count(( $\lambda\, p : p < n \supset$ ppred($p$)), $n$) = count(ppred, $n$)

  count_bounded_and: Lemma
    count(( $\lambda\, p : p < n \wedge$ ppred($p$)), $n$) = count(ppred, $n$)
```

pigeon_hole: **Lemma**
  count(ppred1, n) + count(ppred2, n) $\geq n + k$
    $\supset$ count(($\lambda\, p$: ppred1(p) $\wedge$ ppred2(p)), n) $\geq k$

pred1, pred2: **Var** function[nat — bool]

pred_extensionality: **Axiom**
  ($\forall\, p$: pred1(p) = pred2(p)) $\supset$ pred1 = pred2

nk_type: **Type** = **Record** n : nat,
                                    k : nat
                   end record

nk, nk1, nk2: **Var** nk_type
nk_less: function[nk_type, nk_type — bool] ==
  (λ nk1, nk2 : nk1.n + nk1.k < nk2.n + nk2.k)

Proof

Using natinduction, noetherian

count_bounded_imp0: **Lemma**
  $k \geq 0 \supset$ count(($\lambda\, p : p < k \supset$ ppred(p)), 0) = count(ppred, 0)

count_bounded_imp_ind: **Lemma**
  ($k \geq n \supset$ count(($\lambda\, p : p < k \supset$ ppred(p)), n)
        = count(ppred, n))
    $\supset$ ($k \geq n + 1$
        $\supset$ count(($\lambda\, p : p < k \supset$ ppred(p)), $n + 1$)
          = count(ppred, $n + 1$))

count_bounded_imp_k: **Lemma**
  ($k \geq n \supset$ count(($\lambda\, p : p < k \supset$ ppred(p)), n)
        = count(ppred, n))

count_bounded_imp0_pr: **Prove** count_bounded_imp0 **from**
  count {i — 0},
  count {ppred — ($\lambda\, p : p < k \supset$ ppred(p)), i — 0}

count_bounded_imp_ind_pr: **Prove** count_bounded_imp_ind **from**
  count {i — $n + 1$},
  count {ppred — ($\lambda\, p : p < k \supset$ ppred(p)), i — $n + 1$}

```
count_bounded_imp_k_pr: Prove count_bounded_imp_k from
  induction
    {prop
       — ( λ n :
            k ≥ n
              ⊃ count(( λ p : p < k ⊃ ppred(p)), n)
                = count(ppred, n)),
     i — n},
  count_bounded_imp0,
  count_bounded_imp_ind {n — j@p1}

count_bounded_imp_pr: Prove count_bounded_imp from
  count_bounded_imp_k {k — n}

count_bounded_and0: Lemma
  k ≥ 0 ⊃ count(( λ p : p < k ∧ ppred(p)), 0) = count(ppred, 0)

count_bounded_and_ind: Lemma
  (k ≥ n ⊃ count(( λ p : p < k ∧ ppred(p)), n) = count(ppred, n))
     ⊃ (k ≥ n + 1
            ⊃ count(( λ p : p < k ∧ ppred(p)), n + 1)
              = count(ppred, n + 1))

count_bounded_and_k: Lemma
  (k ≥ n ⊃ count(( λ p : p < k ∧ ppred(p)), n) = count(ppred, n))

count_bounded_and0_pr: Prove count_bounded_and0 from
  count {i — 0},
  count {ppred — ( λ p : p < k ∧ ppred(p)), i — 0}

count_bounded_and_ind_pr: Prove count_bounded_and_ind from
  count {i — n + 1},
  count {ppred — ( λ p : p < k ∧ ppred(p)), i — n + 1}
```

```
count_bounded_and_k_pr: Prove count_bounded_and_k from
  induction
    {prop
      — ( λ n :
          k ≥ n
            ⊃ count(( λ p : p < k ∧ ppred(p)), n)
              = count(ppred, n)),
     i — n},
   count_bounded_and0,
   count_bounded_and_ind {n — j@p1}

count_bounded_and_pr: Prove count_bounded_and from
   count_bounded_and_k {k — n}

count_false_pr: Prove count_false from
   count_true,
   count_complement {ppred — ( λ p : true)},
   pred_extensionality
     {pred1 — ( λ p : ¬true),
      pred2 — ( λ p : false)}

cc0: Lemma count(( λ q : ¬ppred(q)), 0) = 0 − count(ppred, 0)

cc_ind: Lemma
   (count(( λ q : ¬ppred(q)), n) = n − count(ppred, n))
      ⊃ (count(( λ q : ¬ppred(q)), n + 1)
            = n + 1 − count(ppred, n + 1))

cc0_pr: Prove cc0 from
   count {ppred — ( λ q : ¬ppred(q)), i — 0},
   count {i — 0}

cc_ind_pr: Prove cc_ind from
   count {ppred — ( λ q : ¬ppred(q)), i — n + 1},
   count {i — n + 1}
```

```
count_complement_pr: Prove count_complement from
  induction
    {prop
      ← ( λ n :
           count(( λ q : ¬ppred(q)), n) = n − count(ppred, n)),
     i ← n},
  cc0,
  cc_ind {n ← j@p1}

instance: Module is noetherian[nk_type, nk_less]
nk_measure: function[nk_type → nat] ==
  ( λ nk1 : nk1.n + nk1.k)

nk_well_founded: Prove well_founded {measure ← nk_measure}

nk_ph_pred: function[function[nat → bool],
                       function[nat → bool], nk_type → bool] =
  ( λ ppred1, ppred2, nk :
      count(ppred1, nk.n) + count(ppred2, nk.n) ≥ nk.n + nk.k
        ⊃ count(( λ p : ppred1(p) ∧ ppred2(p)), nk.n) ≥ nk.k)
nk_noeth_pred: function[function[nat → bool],
                          function[nat → bool], nk_type
                          → bool] =
  ( λ ppred1, ppred2, nk1 :
      ( ∀ nk2 :
         nk_less(nk2, nk1) ⊃ nk_ph_pred(ppred1, ppred2, nk2)))

ph_case1: Lemma
  count(( λ p : ppred1(p) ∧ ppred2(p)), pred(n)) ≥ k
    ⊃ count(( λ p : ppred1(p) ∧ ppred2(p)), n) ≥ k

ph_case1_pr: Prove ph_case1 from
  count {ppred ← ( λ p : ppred1(p) ∧ ppred2(p)), i ← n}

ph_case2: Lemma
  count(ppred1, pred(n)) + count(ppred2, pred(n)) < pred(n) + k
      ∧ count(ppred1, n) + count(ppred2, n) ≥ n + k
        ∧ count(( λ p : ppred1(p) ∧ ppred2(p)), pred(n)) ≥ pred(k)
    ⊃ count(( λ p : ppred1(p) ∧ ppred2(p)), n) ≥ k
```

ph_case2a: **Lemma**
count(ppred1, pred(n)) + count(ppred2, pred(n)) $<$ pred(n) + k
$\wedge$ count(ppred1, n) + count(ppred2, n) $\geq n + k$
$\supset$ ppred1(pred(n)) $\wedge$ ppred2(pred(n))

ph_case2b: **Lemma**
$n > 0 \wedge k > 0$
$\wedge$ count(ppred1, pred(n)) + count(ppred2, pred(n))
$<$ pred(n) + k
$\wedge$ count(ppred1, n) + count(ppred2, n) $\geq n + k$
$\supset$ count(ppred1, pred(n)) + count(ppred2, pred(n))
$\geq$ pred(n) + pred(k)

ph_case2a_pr: **Prove** ph_case2a **from**
count {ppred — ppred1, i — n},
count {ppred — ppred2, i — n}

ph_case2b_pr: **Prove** ph_case2b **from**
count {ppred — ppred1, i — n},
count {ppred — ppred2, i — n}

ph_case2_pr: **Prove** ph_case2 **from**
count {ppred — ($\lambda\, p$: ppred1(p) $\wedge$ ppred2(p)), i — n},
ph_case2a

ph_case0: **Lemma**
($n = 0 \vee k = 0$)
$\supset$ (count(ppred1, n) + count(ppred2, n) $\geq n + k$
$\supset$ count(($\lambda\, p$: ppred1(p) $\wedge$ ppred2(p)), n) $\geq k$)

ph_case0n: **Lemma**
(count(ppred1, 0) + count(ppred2, 0) $\geq k$
$\supset$ count(($\lambda\, p$: ppred1(p) $\wedge$ ppred2(p)), 0) $\geq k$)

ph_case0n_pr: **Prove** ph_case0n **from**
count {ppred — ppred1, i — 0},
count {ppred — ppred2, i — 0},
count {ppred — ($\lambda\, p$: ppred1(p) $\wedge$ ppred2(p)), i — 0}

ph_case0k: **Lemma** count(($\lambda\, p$: ppred1(p) $\wedge$ ppred2(p)), n) ≥ 0

```
ph_case0k_pr: Prove ph_case0k from
   nat_invariant
      {nat_var ← count(( λ p : ppred1(p) ∧ ppred2(p)), n)}

ph_case0_pr: Prove ph_case0 from ph_case0n, ph_case0k

nk_ph_expand: Lemma
   ( ∀ n, k :
        (count(ppred1, pred(n)) + count(ppred2, pred(n))
                   ≥ pred(n) + pred(k)
                ⊃ count(( λ p : ppred1(p) ∧ ppred2(p)), pred(n))
                   ≥ pred(k))
            ∧ (count(ppred1, pred(n)) + count(ppred2, pred(n))
                   ≥ pred(n) + k
                ⊃ count(( λ p : ppred1(p) ∧ ppred2(p)), pred(n))
                   ≥ k)
          ⊃ (count(ppred1, n) + count(ppred2, n) ≥ n + k
               ⊃ count(( λ p : ppred1(p) ∧ ppred2(p)), n) ≥ k))

nk_ph_expand_pr: Prove nk_ph_expand from
   ph_case0, ph_case1, ph_case2, ph_case2a, ph_case2b

nk_ph_noeth_hyp: Lemma
   ( ∀ nk1 :
        nk_noeth_pred(ppred1, ppred2, nk1)
          ⊃ nk_ph_pred(ppred1, ppred2, nk1))

nk_ph_noeth_hyp_pr: Prove nk_ph_noeth_hyp from
   nk_ph_pred {nk ← nk1},
   nk_noeth_pred {nk2 ← nk1 with [(n) := pred(nk1.n)]},
   nk_noeth_pred
      {nk2 ← nk1 with [(n) := pred(nk1.n), (k) := pred(nk1.k)]},
   nk_ph_pred {nk ← nk1 with [(n) := pred(nk1.n)]},
   nk_ph_pred
      {nk ← nk1 with [(n) := pred(nk1.n), (k) := pred(nk1.k)]},
   nk_ph_expand {n ← nk1.n, k ← nk1.k},
   ph_case0 {n ← nk1.n, k ← nk1.k},
   nat_invariant {nat_var ← nk1.n},
   nat_invariant {nat_var ← nk1.k}
```

```
nk_ph_lem: Lemma nk_ph_pred(ppred1, ppred2, nk)

nk_ph_lem_pr: Prove nk_ph_lem from
  general_induction
    {p — ( λ nk : nk_ph_pred(ppred1, ppred2, nk)),
     d₂ — nk2@p3,
     d — nk@c},
  nk_ph_noeth_hyp {nk1 — d₁@p1},
  nk_noeth_pred {nk1 — d₁@p1}

pigeon_hole_pr: Prove pigeon_hole from
  nk_ph_lem {nk — nk with [(n) := n@c, (k) := k@c]},
  nk_ph_pred {nk — nk@p1}

exists_less: function[function[nat — bool], nat — bool] =
  ( λ ppred, n : ( ∃ p : p < n ∧ ppred(p)))

count_exists_base: Lemma
  count(ppred, 0) > 0 ⊃ exists_less(ppred, 0)

count_exists_base_pr: Prove count_exists_base from
  count {i — 0}, exists_less {n — 0}

count_exists_ind: Lemma
  (count(ppred, n) > 0 ⊃ exists_less(ppred, n))
     ⊃ (count(ppred, n + 1) > 0 ⊃ exists_less(ppred, n + 1))

count_exists_ind_pr: Prove count_exists_ind from
  count {i — n + 1},
  exists_less,
  exists_less
    {n — n + 1,
     p — ( if ppred(n) then n else p@p2 end if)}
```

```
count_exists_pr: Prove count_exists {$p$ — $p$@$p1$} from
  induction
    {prop
      — ( $\lambda\, n$ : count(ppred, $n$) > 0 $\supset$ exists_less(ppred, $n$)),
     $i$ — $n$@$c$},
  count_exists_base,
  count_exists_ind {$n$ — $j$@$p1$},
  exists_less {$n$ — $i$@$p1$}

count_base: Sublemma count(ppred, 0) = 0

count_base_pr: Prove count_base from count {$i$ — 0}

count_true_ind: Sublemma
  (count(( $\lambda\, p$ : true), $n$) = $n$)
    $\supset$ count(( $\lambda\, p$ : true), $n + 1$) = $n + 1$

count_true_ind_pr: Prove count_true_ind from
  count {ppred — ( $\lambda\, p$ : true), $i$ — $n + 1$}

count_true_pr: Prove count_true from
  induction
    {prop — ( $\lambda\, n$ : count(( $\lambda\, p$ : true), $n$) = $n$),
     $i$ — $n$@$c$},
  count_base {ppred — ( $\lambda\, p$ : true)},
  count_true_ind {$n$ — $j$@$p1$}

End countmod
```

natinduction: **Module**

Theory

i, j, m, m_1, n: **Var** nat
p, prop: **Var** function[nat — bool]

induction: **Theorem**
 $(\text{prop}(0) \wedge (\forall j : \text{prop}(j) \supset \text{prop}(j+1))) \supset \text{prop}(i)$

complete_induction: **Theorem**
 $(\forall i : (\forall j : j < i \supset p(j)) \supset p(i)) \supset (\forall n : p(n))$

induction_m: **Theorem**
 $p(m) \wedge (\forall i : i \geq m \wedge p(i) \supset p(i+1))$
  $\supset (\forall n : n \geq m \supset p(n))$

limited_induction: **Theorem**
 $(m \leq m_1 \supset p(m)) \wedge (\forall i : i \geq m \wedge i < m_1 \wedge p(i) \supset p(i+1))$
  $\supset (\forall n : n \geq m \wedge n \leq m_1 \supset p(n))$

Proof

Using noetherian

less: function[nat, nat — bool] == $(\lambda\, m, n : m < n)$

instance: **Module is** noetherian[nat, less]
x: **Var** nat
identity: function[nat — nat] == $(\lambda\, n : n)$

discharge: **Prove** well_founded {measure — identity}

complete_ind_pr: **Prove** complete_induction $\{i — d_1@p1\}$ **from**
 general_induction $\{d — n,\ d_2 — j\}$

ind_proof: **Prove** induction $\{j — \text{pred}(d_1@p1)\}$ **from**
 general_induction $\{p — \text{prop},\ d — i,\ d_2 — j\}$

```
ind_m_proof: Prove induction_m {i — j@p1 + m} from
   induction
      {prop — ( λ x : p@c(x + m)),
       i —  if n ≥ m then n − m else 0 end if}

limited_proof: Prove limited_induction {i — i@p1} from
   induction_m {p — ( λ x : x ≤ m₁ ⊃ p@c(x))}

End natinduction
```

division: **Module**

Using multiplication, absmod

Exporting all

Theory

$x, y, z, x_1, y_1, z_1, x_2, y_2, z_2$: **Var** number
$\lceil \star 1 \rceil$: function[number → int]

ceil_defn: **Axiom** $\lceil x \rceil \geq x \wedge \lceil x \rceil - 1 < x$

mult_div_1: **Axiom** $z \neq 0 \supset x \star y / z = x \star (y/z)$

mult_div_2: **Axiom** $z \neq 0 \supset x \star y / z = (x/z) \star y$

mult_div_3: **Axiom** $z \neq 0 \supset (z/z) = 1$

mult_div: **Lemma** $y \neq 0 \supset (x/y) \star y = x$

div_cancel: **Lemma** $x \neq 0 \supset x \star y / x = y$

div_distrib: **Lemma** $z \neq 0 \supset ((x + y)/z) = (x/z) + (y/z)$

ceil_mult_div: **Lemma** $y > 0 \supset \lceil x/y \rceil \star y \geq x$

ceil_plus_mult_div: **Lemma** $y > 0 \supset \lceil x/y \rceil + 1 \star y > x$

div_nonnegative: **Lemma** $x \geq 0 \wedge y > 0 \supset (x/y) \geq 0$

div_minus_distrib: **Lemma**
$z \neq 0 \supset (x - y)/z = (x/z) - (y/z)$

div_ineq: **Lemma** $z > 0 \wedge x \leq y \supset (x/z) \leq (y/z)$

abs_div: **Lemma** $y > 0 \supset |x/y| = |x|/y$

mult_minus: **Lemma** $y \neq 0 \supset -(x/y) = (-x/y)$

div_minus_1: **Lemma** $y > 0 \wedge x < 0 \supset (x/y) < 0$

Proof

```
div_nonnegative_pr: Prove div_nonnegative from
   mult_non_neg {x — ( if y ≠ 0 then (x/y) else 0 end if)},
   mult_div

div_distrib_pr: Prove div_distrib from
   mult_div_1 {x — x + y, y — 1, z — z},
   mult_rident {x — x + y},
   mult_div_1 {x — x, y — 1, z — z},
   mult_rident,
   mult_div_1 {x — y, y — 1, z — z},
   mult_rident {x — y},
   distrib {z — ( if z ≠ 0 then (1/z) else 0 end if)}

div_cancel_pr: Prove div_cancel from
   mult_div_2 {z — x},
   mult_div_3 {z — x},
   mult_lident {x — y}

mult_div_pr: Prove mult_div from
   mult_div_2 {z — y},
   mult_div_1 {z — y},
   mult_div_3 {z — y},
   mult_rident

abs_div_pr: Prove abs_div from
   |⋆1| {x — ( if y ≠ 0 then (x/y) else 0 end if)},
   |⋆1| ,
   div_nonnegative,
   div_minus_1,
   mult_minus

mult_minus_pr: Prove mult_minus from
   mult_div_1 {x — −1, y — x, z — y},
   ⋆1 ⋆⋆2 {x — −1, y — x},
   ⋆1 ⋆⋆2
      {x — −1,
      y — ( if y ≠ 0 then (x/y) else 1 end if)}
```

```
div_minus_1_pr: Prove div_minus_1 from
   mult_div,
   pos_product
      {x — ( if y ≠ 0 then (x/y) else 0 end if),
       y — y}

div_minus_distrib_pr: Prove div_minus_distrib from
   div_distrib {y — −y}, mult_minus {x — y, y — z}

div_ineq_pr: Prove div_ineq from
   mult_div {y — z},
   mult_div {x — y, y — z},
   mult_gt
      {x — ( if z ≠ 0 then (x/z) else 0 end if),
       y — ( if z ≠ 0 then (y/z) else 0 end if)}

ceil_plus_mult_div_proof: Prove ceil_plus_mult_div from
   ceil_mult_div,
   distrib
      {x — ⌈( if y ≠ 0 then (x/y) else 0 end if)⌉,
       y — 1,
       z — y},
   mult_lident {x — y}

ceil_mult_div_proof: Prove ceil_mult_div from
   mult_div,
   mult_leq
      {x — ⌈( if y ≠ 0 then (x/y) else 0 end if)⌉,
       y — ( if y ≠ 0 then (x/y) else 0 end if),
       z — y},
   ceil_defn {x — ( if y ≠ 0 then (x/y) else 0 end if)}

End division
```

mid_tcc: **Module**

Using mid

Exporting all with mid

Theory

ft_mid_TCC1: **Formula** $(F + 1 > 0)$

ft_mid_TCC2: **Formula** $(N - F \geq 0) \wedge (N - F > 0)$

ft_mid_TCC3: **Formula** $(2 \neq 0)$

Proof

ft_mid_TCC1_PROOF: **Prove** ft_mid_TCC1

ft_mid_TCC2_PROOF: **Prove** ft_mid_TCC2

ft_mid_TCC3_PROOF: **Prove** ft_mid_TCC3

End mid_tcc

mid2_tcc: **Module**

Using mid2

Exporting all with mid2

Theory

ppred: **Var** function[naturalnumber — boolean]
p: **Var** naturalnumber
good_greater_F1_TCC1: **Formula**
$(\text{ppred}(p)) \wedge (\text{count}(\text{ppred}, N) \geq N - F) \supset (F + 1 > 0)$

good_less_NF_TCC1: **Formula**
$(\text{ppred}(p)) \wedge (\text{count}(\text{ppred}, N) \geq N - F)$
$\supset (N - F \geq 0) \wedge (N - F > 0)$

good_greater_F1_pr_TCC1: **Formula** $(F + 1 > 0)$

good_less_NF_pr_TCC1: **Formula** $(N - F \geq 0) \wedge (N - F > 0)$

Proof

good_greater_F1_TCC1_PROOF: **Prove** good_greater_F1_TCC1

good_less_NF_TCC1_PROOF: **Prove** good_less_NF_TCC1

good_greater_F1_pr_TCC1_PROOF: **Prove** good_greater_F1_pr_TCC1

good_less_NF_pr_TCC1_PROOF: **Prove** good_less_NF_pr_TCC1

End mid2_tcc

mid3_tcc: **Module**

Using mid3

Exporting all with mid3

Theory

X: **Var** number
Z: **Var** number
ppred: **Var** function[naturalnumber — boolean]
k: **Var** countmod.posint
ppred2: **Var** function[naturalnumber — boolean]
ppred1: **Var** function[naturalnumber — boolean]
θ: **Var** function[naturalnumber — number]
γ: **Var** function[naturalnumber — number]
q: **Var** naturalnumber
p_3: **Var** naturalnumber
p_1: **Var** naturalnumber
p: **Var** naturalnumber
q_1: **Var** naturalnumber
ft_mid_Pi_TCC1: **Formula** $(2 \neq 0)$

good_geq_F_add1_TCC1: **Formula**
  $(\text{ppred}(p)) \wedge (\text{count}(\text{ppred}, N) \geq N - F) \supset (F + 1 > 0)$

okay_pair_geq_F_add1_TCC1: **Formula**
  $(\text{ppred}(p_1))$
      $\wedge (\text{count}(\text{ppred}, N) \geq N - F \wedge \text{okay\_pairs}(\theta, \gamma, X, \text{ppred}))$
    $\supset (F + 1 > 0)$

okay_pair_geq_F_add1_TCC2: **Formula**
  $(\text{ppred}(q_1))$
      $\wedge (\theta(p_1) \geq \theta_{(F+1)})$
        $\wedge (\text{ppred}(p_1))$
          $\wedge (\text{count}(\text{ppred}, N) \geq N - F$
              $\wedge \text{okay\_pairs}(\theta, \gamma, X, \text{ppred}))$
    $\supset (F + 1 > 0)$

good_between_TCC1: **Formula**

$(\gamma_{(F+1)} \geq \gamma(p)) \wedge (\text{ppred}(p)) \wedge (\text{count}(\text{ppred}, N) \geq N - F)$
$\quad \supset (N - F \geq 0) \wedge (N - F > 0)$

ft_mid_prec_sym1_TCC1: **Formula**

$(\text{okay\_Readpred}(\gamma, Z, \text{ppred}))$
$\quad \wedge (\text{okay\_Readpred}(\theta, Z, \text{ppred}))$
$\quad\quad \wedge (\text{okay\_pairs}(\theta, \gamma, X, \text{ppred}))$
$\quad\quad\quad \wedge (\text{count}(\text{ppred}, N) \geq N - F)$
$\quad \supset (F + 1 > 0)$

ft_mid_prec_sym1_TCC2: **Formula**

$(\text{okay\_Readpred}(\gamma, Z, \text{ppred}))$
$\quad \wedge (\text{okay\_Readpred}(\theta, Z, \text{ppred}))$
$\quad\quad \wedge (\text{okay\_pairs}(\theta, \gamma, X, \text{ppred}))$
$\quad\quad\quad \wedge (\text{count}(\text{ppred}, N) \geq N - F)$
$\quad \supset (N - F \geq 0) \wedge (N - F > 0)$

ft_mid_prec_sym1_TCC3: **Formula**

$(\text{count}(\text{ppred}, N) \geq N - F$
$\quad\quad \wedge \text{okay\_pairs}(\theta, \gamma, X, \text{ppred})$
$\quad\quad\quad \wedge \text{okay\_Readpred}(\theta, Z, \text{ppred})$
$\quad\quad\quad\quad \wedge \text{okay\_Readpred}(\gamma, Z, \text{ppred})$
$\quad\quad\quad\quad\quad \wedge ((\theta_{(F+1)} + \theta_{(N-F)})$
$\quad\quad\quad\quad\quad\quad \geq (\gamma_{(F+1)} + \gamma_{(N-F)})))$
$\quad \supset (F + 1 > 0)$

ft_mid_prec_sym1_TCC4: **Formula**

$(\text{count}(\text{ppred}, N) \geq N - F$
$\quad\quad \wedge \text{okay\_pairs}(\theta, \gamma, X, \text{ppred})$
$\quad\quad\quad \wedge \text{okay\_Readpred}(\theta, Z, \text{ppred})$
$\quad\quad\quad\quad \wedge \text{okay\_Readpred}(\gamma, Z, \text{ppred})$
$\quad\quad\quad\quad\quad \wedge ((\theta_{(F+1)} + \theta_{(N-F)})$
$\quad\quad\quad\quad\quad\quad \geq (\gamma_{(F+1)} + \gamma_{(N-F)})))$
$\quad \supset (N - F \geq 0) \wedge (N - F > 0)$

mid_gt_imp_sel_gt_TCC1: **Formula**

$((cfn_{MID}(p, \theta) \geq cfn_{MID}(q, \gamma))) \supset (F + 1 > 0)$

mid_gt_imp_sel_gt_TCC2: **Formula**
$((cfn_{MID}(p,\theta) \geq cfn_{MID}(q,\gamma)))$
$\supset (N - F \geq 0) \wedge (N - F > 0)$

ft_mid_prec_sym1_pr_TCC1: **Formula** $(F + 1 > 0)$

ft_mid_prec_sym1_pr_TCC2: **Formula** $(N - F \geq 0) \wedge (N - F > 0)$

Proof

ft_mid_Pi_TCC1_PROOF: **Prove** ft_mid_Pi_TCC1

good_geq_F_add1_TCC1_PROOF: **Prove** good_geq_F_add1_TCC1

okay_pair_geq_F_add1_TCC1_PROOF: **Prove**
okay_pair_geq_F_add1_TCC1

okay_pair_geq_F_add1_TCC2_PROOF: **Prove**
okay_pair_geq_F_add1_TCC2

good_between_TCC1_PROOF: **Prove** good_between_TCC1

ft_mid_prec_sym1_TCC1_PROOF: **Prove** ft_mid_prec_sym1_TCC1

ft_mid_prec_sym1_TCC2_PROOF: **Prove** ft_mid_prec_sym1_TCC2

ft_mid_prec_sym1_TCC3_PROOF: **Prove** ft_mid_prec_sym1_TCC3

ft_mid_prec_sym1_TCC4_PROOF: **Prove** ft_mid_prec_sym1_TCC4

mid_gt_imp_sel_gt_TCC1_PROOF: **Prove** mid_gt_imp_sel_gt_TCC1

mid_gt_imp_sel_gt_TCC2_PROOF: **Prove** mid_gt_imp_sel_gt_TCC2

ft_mid_prec_sym1_pr_TCC1_PROOF: **Prove** ft_mid_prec_sym1_pr_TCC1

ft_mid_prec_sym1_pr_TCC2_PROOF: **Prove** ft_mid_prec_sym1_pr_TCC2

End mid3_tcc

mid4_tcc: **Module**

Using mid4

Exporting all with mid4

Theory

q: **Var** naturalnumber
p: **Var** naturalnumber
y: **Var** number
x: **Var** number
p_1: **Var** naturalnumber
ft_mid_less_TCC1: **Formula** $(F + 1 > 0)$

ft_mid_greater_TCC1: **Formula** $(N - F \geq 0) \wedge (N - F > 0)$

Proof

ft_mid_less_TCC1_PROOF: **Prove** ft_mid_less_TCC1

ft_mid_greater_TCC1_PROOF: **Prove** ft_mid_greater_TCC1

End mid4_tcc

mid: **Module**

Using arith, clockassumptions: select_defs, ft_mid_assume

Exporting all with select_defs

Theory

process: **Type is** nat
Clocktime: **Type is** number
l, m, n, p, q: **Var** process
ϑ: **Var** function[process — Clocktime]
i, j, k: **Var** posint
T, X, Y, Z: **Var** Clocktime
cfn_{MID}: function[process, function[process — Clocktime]
— Clocktime] =
$(\lambda\, p, \vartheta : (\vartheta_{(F+1)} + \vartheta_{(N-F)})/2)$

ft_mid_trans_inv: **Lemma**
$cfn_{MID}(p, (\lambda\, q : \vartheta(q) + X)) = cfn_{MID}(p, \vartheta) + X$

Proof

add_assoc_hack: **Lemma** $X + Y + Z + Y = (X + Z) + 2 \star Y$

add_assoc_hack_pr: **Prove** add_assoc_hack **from**
$\star 1 \star \star 2\ \{x — 2,\ y — Y\}$

```
ft_mid_trans_inv_pr: Prove ft_mid_trans_inv from
   $cfn_{MID}$,
   $cfn_{MID}$ $\{\vartheta — (\lambda q : \vartheta(q) + X)\}$,
   select_trans_inv $\{k — F + 1\}$,
   select_trans_inv $\{k — N - F\}$,
   add_assoc_hack
      $\{X — \vartheta_{(F+1)}$,
       $Z — \vartheta_{(N-F)}$,
       $Y — X\}$,
   div_distrib
      $\{x — (\vartheta_{(F+1)} + \vartheta_{(N-F)})$,
       $y — 2 \star X$,
       $z — 2\}$,
   div_cancel $\{x — 2,\ y — X\}$,
   ft_mid_maxfaults
```

End mid

mid2: **Module**

Using arith, clockassumptions, mid

Exporting all with mid

Theory

Clocktime: **Type is** number
m, n, p, q, p_1, q_1: **Var** process
i, j, k, l: **Var** posint
x, y, z, r, s, t: **Var** time
D, X, Y, Z, R, S, T: **Var** Clocktime
$\vartheta, \theta, \gamma$: **Var** function[process — Clocktime]
ppred, ppred1, ppred2: **Var** function[process — bool]

good_greater_F1: **Lemma**
count(ppred, N) $\geq N - F \supset (\exists p :$ ppred(p) $\wedge\ \vartheta(p) \geq \vartheta_{(F+1)})$

good_less_NF: **Lemma**
count(ppred, N) $\geq N - F \supset (\exists p :$ ppred(p) $\wedge\ \vartheta(p) \leq \vartheta_{(N-F)})$

Proof

good_greater_F1_pr: **Prove** good_greater_F1 $\{p — p@p3\}$ **from**
count_geq_select $\{k — F + 1\}$,
ft_mid_maxfaults,
count_exists
{ppred — (λp_1 : ppred1@p4(p_1) $\wedge$ ppred2@p4(p_1)),
$n — N$},
pigeon_hole
{ppred1 ← ppred,
ppred2 — ($\lambda p_1 : \vartheta(p_1) \geq \vartheta_{(F+1)}$),
$n — N$,
$k — 1$}

```
good_less_NF_pr: Prove good_less_NF {$p$ — $p@p3$} from
  count_leq_select {$k$ — $N - F$},
  ft_mid_maxfaults,
  count_exists
    {ppred — ( $\lambda p_1$ : ppred1@p4($p_1$) ∧ ppred2@p4($p_1$)),
     $n$ — $N$},
  pigeon_hole
    {ppred1 — ppred,
     ppred2 — ( $\lambda p_1 : \vartheta_{(N-F)} \geq \vartheta(p_1)$),
     $n$ — $N$,
     $k$ — 1}
```

End mid2

mid3: **Module**

Using arith, clockassumptions, mid2

Exporting all with mid2

Theory

Clocktime: **Type is** number
m, n, p, q, p_1, q_1: **Var** process
i, j, k, l: **Var** posint
x, y, z, r, s, t: **Var** time
D, X, Y, Z, R, S, T: **Var** Clocktime
$\vartheta, \theta, \gamma$: **Var** function[process → Clocktime]
ppred, ppred1, ppred2: **Var** function[process → bool]
ft_mid_Pi: function[Clocktime, Clocktime → Clocktime] ==
  $(\lambda X, Z : Z/2 + X)$

exchange_order: **Lemma**
  $\text{ppred}(p)$
    $\wedge\ \text{ppred}(q)$
      $\wedge\ \theta(q) \le \theta(p)$
        $\wedge\ \gamma(p) \le \gamma(q) \wedge \text{okay\_pairs}(\theta, \gamma, X, \text{ppred})$
  $\supset |\theta(p) - \gamma(q)| \le X$

good_geq_F_add1: **Lemma**
  $\text{count}(\text{ppred}, N) \ge N - F \supset (\exists\, p : \text{ppred}(p) \wedge \vartheta(p) \ge \vartheta_{(F+1)})$

okay_pair_geq_F_add1: **Lemma**
  $\text{count}(\text{ppred}, N) \ge N - F \wedge \text{okay\_pairs}(\theta, \gamma, X, \text{ppred})$
  $\supset (\exists\, p_1, q_1 :$
      $\text{ppred}(p_1)$
        $\wedge\ \theta(p_1) \ge \theta_{(F+1)}$
          $\wedge\ \text{ppred}(q_1)$
            $\wedge\ \gamma(q_1) \ge \gamma_{(F+1)} \wedge |\theta(p_1) - \gamma(q_1)| \le X)$

good_between: **Lemma**
  $\text{count}(\text{ppred}, N) \ge N - F$
  $\supset (\exists\, p : \text{ppred}(p) \wedge \gamma_{(F+1)} \ge \gamma(p) \wedge \theta(p) \ge \theta_{(N-F)})$

ft_mid_precision_enhancement: **Lemma**

$$\begin{aligned}&\mathsf{ppred}(p)\\ &\quad\wedge \mathsf{ppred}(q)\\ &\quad\quad\wedge \mathsf{count}(\mathsf{ppred}, N) \geq N - F\\ &\quad\quad\quad\wedge \mathsf{okay\_pairs}(\theta, \gamma, X, \mathsf{ppred})\\ &\quad\quad\quad\quad\wedge \mathsf{okay\_Readpred}(\theta, Z, \mathsf{ppred})\\ &\quad\quad\quad\quad\quad\wedge \mathsf{okay\_Readpred}(\gamma, Z, \mathsf{ppred})\\ &\supset |cfn_{MID}(p, \theta) - cfn_{MID}(q, \gamma)| \leq \mathsf{ft\_mid\_Pi}(X, Z)\end{aligned}$$

ft_mid_prec_enh_sym: **Lemma**

$$\begin{aligned}&\mathsf{ppred}(p)\\ &\quad\wedge \mathsf{ppred}(q)\\ &\quad\quad\wedge \mathsf{count}(\mathsf{ppred}, N) \geq N - F\\ &\quad\quad\quad\wedge \mathsf{okay\_pairs}(\theta, \gamma, X, \mathsf{ppred})\\ &\quad\quad\quad\quad\wedge \mathsf{okay\_Readpred}(\theta, Z, \mathsf{ppred})\\ &\quad\quad\quad\quad\quad\wedge \mathsf{okay\_Readpred}(\gamma, Z, \mathsf{ppred})\\ &\quad\quad\quad\quad\quad\quad\wedge (cfn_{MID}(p, \theta) \geq cfn_{MID}(q, \gamma))\\ &\supset |cfn_{MID}(p, \theta) - cfn_{MID}(q, \gamma)| \leq \mathsf{ft\_mid\_Pi}(X, Z)\end{aligned}$$

ft_mid_prec_sym1: **Lemma**

$$\begin{aligned}&\mathsf{count}(\mathsf{ppred}, N) \geq N - F\\ &\quad\wedge \mathsf{okay\_pairs}(\theta, \gamma, X, \mathsf{ppred})\\ &\quad\quad\wedge \mathsf{okay\_Readpred}(\theta, Z, \mathsf{ppred})\\ &\quad\quad\quad\wedge \mathsf{okay\_Readpred}(\gamma, Z, \mathsf{ppred})\\ &\quad\quad\quad\quad\wedge ((\theta_{(F+1)} + \theta_{(N-F)})\\ &\quad\quad\quad\quad\quad\geq (\gamma_{(F+1)} + \gamma_{(N-F)}))\\ &\supset |(\theta_{(F+1)} + \theta_{(N-F)}) - (\gamma_{(F+1)} + \gamma_{(N-F)})|\\ &\quad\leq Z + 2 \star X\end{aligned}$$

mid_gt_imp_sel_gt: **Lemma**

$$\begin{aligned}&(cfn_{MID}(p, \theta) \geq cfn_{MID}(q, \gamma))\\ &\quad\supset ((\theta_{(F+1)} + \theta_{(N-F)}) \geq (\gamma_{(F+1)} + \gamma_{(N-F)}))\end{aligned}$$

okay_pairs_sym: **Lemma**

$$\mathsf{okay\_pairs}(\theta, \gamma, X, \mathsf{ppred}) \supset \mathsf{okay\_pairs}(\gamma, \theta, X, \mathsf{ppred})$$

Proof

ft_mid_prec_sym1_pr: **Prove** ft_mid_prec_sym1 **from**

good_between,
okay_pair_geq_F_add1,
good_less_NF $\{\vartheta — \gamma\}$,
abs_geq
$\{x — (\gamma(q_1@p2) - \gamma(p@p3)) + (\theta(p@p1) - \gamma(p@p1)) + (\theta(p_1@p2) - \gamma(q_1@p2)),$
$y — (\theta_{(F+1)} + \theta_{(N-F)}) - (\gamma_{(F+1)} + \gamma_{(N-F)})\},$
abs_plus
$\{x — (\gamma(q_1@p2) - \gamma(p@p3)) + (\theta(p@p1) - \gamma(p@p1)),$
$y — (\theta(p_1@p2) - \gamma(q_1@p2))\},$
abs_plus
$\{x — (\gamma(q_1@p2) - \gamma(p@p3)),$
$y — (\theta(p@p1) - \gamma(p@p1))\},$
okay_pairs $\{\gamma — \theta,\ \theta — \gamma,\ p_3 — p@p1\},$
okay_Readpred
$\{\gamma — \gamma,$
$Y — Z,$
$l — q_1@p2,$
$m — p@p3\},$
distrib $\{x — 1,\ y — 1,\ z — X\},$
mult_lident $\{x — X\}$

mid_gt_imp_sel_gt_pr: **Prove** mid_gt_imp_sel_gt **from**

cfn_{MID} $\{\vartheta — \theta\},$
cfn_{MID} $\{\vartheta — \gamma,\ p — q\},$
mult_leq
$\{x — cfn_{MID}(p,\theta),$
$y — cfn_{MID}(q,\gamma),$
$z — 2\},$
mult_div $\{x — (\theta_{(F+1)} + \theta_{(N-F)}),\ y — 2\},$
mult_div $\{x — (\gamma_{(F+1)} + \gamma_{(N-F)}),\ y — 2\}$

ft_mid_prec_enh_sym_pr: **Prove** ft_mid_prec_enh_sym **from**
  cfn_{MID} $\{\vartheta — \theta\}$,
  cfn_{MID} $\{\vartheta — \gamma,\ p — q\}$,
  div_minus_distrib
    $\{x — (\theta_{(F+1)} + \theta_{(N-F)})$,
    $y — (\gamma_{(F+1)} + \gamma_{(N-F)})$,
    $z — 2\}$,
  abs_div
    $\{x — (\theta_{(F+1)} + \theta_{(N-F)}) - (\gamma_{(F+1)} + \gamma_{(N-F)})$,
    $y — 2\}$,
  ft_mid_prec_sym1,
  mid_gt_imp_sel_gt,
  div_ineq
    $\{x — |(\theta_{(F+1)} + \theta_{(N-F)}) - (\gamma_{(F+1)} + \gamma_{(N-F)})|$,
    $y — Z + 2 \star X$,
    $z — 2\}$,
  div_distrib $\{x — Z,\ y — 2 \star X,\ z — 2\}$,
  div_cancel $\{x — 2,\ y — X\}$

okay_pairs_sym_pr: **Prove** okay_pairs_sym **from**
  okay_pairs $\{\gamma — \theta,\ \theta — \gamma,\ p_3 — p_3$@p2$\}$,
  okay_pairs $\{\gamma — \gamma,\ \theta — \theta\}$,
  abs_com $\{x — \theta(p_3$@p2$),\ y — \gamma(p_3$@p2$)\}$

ft_mid_precision_enhancement_pr: **Prove**
  ft_mid_precision_enhancement **from**
  ft_mid_prec_enh_sym,
  ft_mid_prec_enh_sym
    $\{p — q$@$p1$,
    $q — p$@$p1$,
    $\theta — \gamma$@p1,
    $\gamma — \theta$@p1$\}$,
  okay_pairs_sym,
  abs_com $\{x — cfn_{MID}(p,\theta),\ y — cfn_{MID}(q,\gamma)\}$

```
okay_pair_geq_F_add1_pr: Prove
  okay_pair_geq_F_add1
    {p1 ← if (θ(p@p2) ≥ θ(p@p1))
          then p@p2
          elsif (γ(p@p1) ≥ γ(p@p2)) then p@p1 else p@p3
          end if,
     q1
       ← if (θ(p@p2) ≥ θ(p@p1))
          then p@p2
          elsif (γ(p@p1) ≥ γ(p@p2)) then p@p1 else q@p3
          end if} from
  good_geq_F_add1 {ϑ ← θ},
  good_geq_F_add1 {ϑ ← γ},
  exchange_order {p ← p@p1, q ← p@p2},
  okay_pairs {γ ← θ, θ ← γ, p3 ← p@p1},
  okay_pairs {γ ← θ, θ ← γ, p3 ← p@p2}

good_geq_F_add1_pr: Prove good_geq_F_add1 {p ← p@p1} from
  count_exists
    {ppred ← ( λ p : ((ppred1@p2)p) ∧ ((ppred2@p2)p)),
     n ← N},
  pigeon_hole
    {n ← N,
     k ← 1,
     ppred1 ← ppred,
     ppred2 ← ( λ p : ϑ(p) ≥ ϑ((k@p3)))},
  count_geq_select {k ← F + 1},
  ft_mid_maxfaults
```

good_between_pr: **Prove** good_between $\{p — p@p1\}$ **from**
  count_exists
    {ppred — $(\lambda p : ((\text{ppred1}@p2)p) \wedge ((\text{ppred2}@p2)p))$,
     $n — N\}$,
  pigeon_hole
    $\{n — N$,
     $k — 1$,
     ppred1 — $(\lambda p : ((\text{ppred1}@p3)p) \wedge ((\text{ppred2}@p3)p))$,
     ppred2 — $(\lambda p : \theta(p) \geq \theta_{((k@p4))})\}$,
  pigeon_hole
    $\{n — N$,
     $k — k@p5$,
     ppred1 — ppred,
     ppred2 — $(\lambda p : \gamma_{((k@p5))} \geq \gamma(p))\}$,
  count_geq_select $\{\vartheta — \theta,\ k — N - F\}$,
  count_leq_select $\{\vartheta — \gamma,\ k — F + 1\}$,
  ft_mid_maxfaults

exchange_order_pr: **Prove** exchange_order **from**
  okay_pairs $\{\gamma — \theta,\ \theta — \gamma,\ p_3 — p\}$,
  okay_pairs $\{\gamma — \theta,\ \theta — \gamma,\ p_3 — q\}$,
  abs_geq $\{x — (\theta(p) - \gamma(p)),\ y — \theta(p) - \gamma(q)\}$,
  abs_geq $\{x — (\gamma(q) - \theta(q)),\ y — \gamma(q) - \theta(p)\}$,
  abs_com $\{x — \theta(q),\ y — \gamma(q)\}$,
  abs_com $\{x — \theta(p),\ y — \gamma(q)\}$

End mid3

mid4: **Module**

Using arith, clockassumptions, mid3

Exporting all with clockassumptions, mid3

Theory

```
process: Type is nat
Clocktime: Type is number
$m, n, p, q, p_1, q_1$: Var process
$i, j, k$: Var posint
$x, y, z, r, s, t$: Var time
$D, X, Y, Z, R, S, T$: Var Clocktime
$\vartheta, \theta, \gamma$: Var function[process → Clocktime]
ppred, ppred1, ppred2: Var function[process → bool]

ft_mid_accuracy_preservation: Lemma
  ppred($p$)
       $\wedge$ ppred($q$)
         $\wedge$ count(ppred, $N$) $\geq N - F \wedge$ okay_Readpred($\vartheta$, $X$, ppred)
     $\supset |cfn_{MID}(p, \vartheta) - \vartheta(q)| \leq X$

ft_mid_less: Lemma $cfn_{MID}(p, \vartheta) \leq \vartheta_{(F+1)}$

ft_mid_greater: Lemma $cfn_{MID}(p, \vartheta) \geq \vartheta_{(N-F)}$

abs_q_less: Lemma
  count(ppred, $N$) $\geq N - F$
     $\supset$ ( $\exists p_1$ : ppred($p_1$) $\wedge$ $\vartheta(p_1) \leq cfn_{MID}(p, \vartheta)$)

abs_q_greater: Lemma
  count(ppred, $N$) $\geq N - F$
     $\supset$ ( $\exists p_1$ : ppred($p_1$) $\wedge$ $\vartheta(p_1) \geq cfn_{MID}(p, \vartheta)$)

ft_mid_bnd_by_good: Lemma
  count(ppred, $N$) $\geq N - F$
     $\supset$ ( $\exists p_1$ :
          ppred($p_1$) $\wedge$ $|cfn_{MID}(p, \vartheta) - \vartheta(q)| \leq |\vartheta(p_1) - \vartheta(q)|$)

maxfaults_lem: Lemma $F + 1 \leq N - F$
```

ft_select: **Lemma** $\vartheta_{(F+1)} \geq \vartheta_{(N-F)}$

Proof

ft_select_pr: **Prove** ft_select **from**
  select_ax $\{i - F+1,\ k - N-F\}$, maxfaults_lem

maxfaults_lem_pr: **Prove** maxfaults_lem **from** ft_mid_maxfaults

ft_mid_bnd_by_good_pr: **Prove**
  ft_mid_bnd_by_good
    $\{p_1 - ($ **if** $cfn_{MID}(p, \vartheta) \geq \vartheta(q)$
        then p_1@p1
        else p_1@p2
        end if$)\}$ **from**
  abs_q_greater,
  abs_q_less,
  abs_com $\{x - \vartheta(q),\ y - \vartheta(p_1@c)\}$,
  abs_com $\{x - \vartheta(q),\ y - cfn_{MID}(p, \vartheta)\}$,
  abs_geq $\{x - x@p3 - y@p3,\ y - x@p4 - y@p4\}$,
  abs_geq $\{x - \vartheta(p_1@c) - \vartheta(q),\ y - cfn_{MID}(p, \vartheta) - \vartheta(q)\}$

abs_q_less_pr: **Prove** abs_q_less $\{p_1 - p@p1\}$ **from**
  good_less_NF, ft_mid_greater

abs_q_greater_pr: **Prove** abs_q_greater $\{p_1 - p@p1\}$ **from**
  good_greater_F1, ft_mid_less

mult_hack: **Lemma** $X + X = 2 \star X$

mult_hack_pr: **Prove** mult_hack **from** $\star 1 \star \star 2$ $\{x - 2,\ y - X\}$

ft_mid_less_pr: **Prove** ft_mid_less **from**
  cfn_{MID},
  ft_select,
  div_ineq
    $\{x - (\vartheta_{(F+1)} + \vartheta_{(N-F)}),$
    $y - (\vartheta_{(F+1)} + \vartheta_{(F+1)}),$
    $z - 2\}$,
  div_cancel $\{x - 2,\ y - \vartheta_{(F+1)}\}$,
  mult_hack $\{X - \vartheta_{(F+1)}\}$

ft_mid_greater_pr: **Prove** ft_mid_greater **from**
  cfn_{MID},
  ft_select,
  div_ineq
    $\{x — (\vartheta_{(N-F)} + \vartheta_{(N-F)}),$
    $y — (\vartheta_{(F+1)} + \vartheta_{(N-F)}),$
    $z — 2\},$
  div_cancel $\{x — 2,\ y — \vartheta_{(N-F)}\},$
  mult_hack $\{X — \vartheta_{(N-F)}\}$

ft_mid_acc_pres_pr: **Prove** ft_mid_accuracy_preservation **from**
  ft_mid_bnd_by_good,
  okay_Readpred
    $\{\gamma — \vartheta,$
    $Y — X,$
    $l — p_1$@p1,
    $m — q@c\}$

End mid4

C Proof Chain Status

C.1 Translation Invariance

```
 Terse proof chain for proof ft_mid_trans_inv_pr in module mid


Use of the formula
  mid.ft_mid
requires the following TCCs to be proven
  mid_tcc.ft_mid_TCC1
  mid_tcc.ft_mid_TCC2
  mid_tcc.ft_mid_TCC3

Use of the formula
  division.div_distrib
requires the following TCCs to be proven
  division_tcc.mult_div_1_TCC1
  division_tcc.mult_div_TCC1
  division_tcc.div_cancel_TCC1
  division_tcc.ceil_mult_div_TCC1
  division_tcc.div_nonnegative_TCC1
  division_tcc.div_ineq_TCC1
  division_tcc.div_minus_1_TCC1

    ================== SUMMARY ==================

 The proof chain is complete

 The axioms and assumptions at the base are:
   clocksort.funsort_trans_inv
   division.mult_div_1
   division.mult_div_2
   division.mult_div_3
   ft_mid_assume.ft_mid_maxfaults
 Total: 5

 The definitions and type-constraints are:
   mid.ft_mid
   multiplication.mult
```

```
Total: 2

The formulae used are:
  division.div_cancel
  division.div_distrib
  division_tcc.ceil_mult_div_TCC1
  division_tcc.div_cancel_TCC1
  division_tcc.div_ineq_TCC1
  division_tcc.div_minus_1_TCC1
  division_tcc.div_nonnegative_TCC1
  division_tcc.mult_div_1_TCC1
  division_tcc.mult_div_TCC1
  mid.add_assoc_hack
  mid_tcc.ft_mid_TCC1
  mid_tcc.ft_mid_TCC2
  mid_tcc.ft_mid_TCC3
  multiplication.distrib
  multiplication.mult_lident
  multiplication.mult_rident
  select_defs.select_trans_inv
Total: 17

The completed proofs are:
  division.div_cancel_pr
  division.div_distrib_pr
  division_tcc.ceil_mult_div_TCC1_PROOF
  division_tcc.div_cancel_TCC1_PROOF
  division_tcc.div_ineq_TCC1_PROOF
  division_tcc.div_minus_1_TCC1_PROOF
  division_tcc.div_nonnegative_TCC1_PROOF
  division_tcc.mult_div_1_TCC1_PROOF
  division_tcc.mult_div_TCC1_PROOF
  mid.add_assoc_hack_pr
  mid.ft_mid_trans_inv_pr
  mid_tcc.ft_mid_TCC1_PROOF
  mid_tcc.ft_mid_TCC3_PROOF
  multiplication.distrib_proof
  multiplication.mult_lident_proof
  multiplication.mult_rident_proof
  select_defs.select_trans_inv_pr
  tcc_mid.ft_mid_TCC2_PROOF
Total: 18
```

C.2 Precision Enhancement

```
 Terse proof chain for proof ft_mid_precision_enhancement_pr in module mid3

Use of the formula
  mid3.ft_mid_prec_enh_sym
requires the following TCCs to be proven
  mid3_tcc.ft_mid_Pi_TCC1
  mid3_tcc.good_geq_F_add1_TCC1
  mid3_tcc.okay_pair_geq_F_add1_TCC1
  mid3_tcc.okay_pair_geq_F_add1_TCC2
  mid3_tcc.good_between_TCC1
  mid3_tcc.ft_mid_prec_sym1_TCC1
  mid3_tcc.ft_mid_prec_sym1_TCC2
  mid3_tcc.ft_mid_prec_sym1_TCC3
  mid3_tcc.ft_mid_prec_sym1_TCC4
  mid3_tcc.mid_gt_imp_sel_gt_TCC1
  mid3_tcc.mid_gt_imp_sel_gt_TCC2
  mid3_tcc.ft_mid_prec_sym1_pr_TCC1
  mid3_tcc.ft_mid_prec_sym1_pr_TCC2

Use of the formula
  mid.ft_mid
requires the following TCCs to be proven
  mid_tcc.ft_mid_TCC1
  mid_tcc.ft_mid_TCC2
  mid_tcc.ft_mid_TCC3

Use of the formula
  division.div_minus_distrib
requires the following TCCs to be proven
  division_tcc.mult_div_1_TCC1
  division_tcc.mult_div_TCC1
```

```
   division_tcc.div_cancel_TCC1
   division_tcc.ceil_mult_div_TCC1
   division_tcc.div_nonnegative_TCC1
   division_tcc.div_ineq_TCC1
   division_tcc.div_minus_1_TCC1

 Use of the formula
   countmod.count_exists
 requires the following TCCs to be proven
   countmod_tcc.posint_TCC1
   countmod_tcc.count_TCC1
   countmod_tcc.count_TCC2
   countmod_tcc.count_TCC3
   countmod_tcc.count_TCC4
   countmod_tcc.count_TCC5

Formula countmod_tcc.count_TCC4 is a termination TCC for countmod.count
Proof of
  countmod_tcc.count_TCC4
must not use
  countmod.count

Formula countmod_tcc.count_TCC5 is a termination TCC for countmod.count
Proof of
  countmod_tcc.count_TCC5
must not use
  countmod.count

Use of the formula
  natinduction.induction
requires the following TCCs to be proven
  natinduction_tcc.ind_m_proof_TCC1

Use of the formula
  noetherian[naturalnumber, natinduction.less].general_induction
requires the following assumptions to be discharged
  noetherian[naturalnumber, natinduction.less].well_founded

Use of the formula
```

```
  noetherian[countmod.nk_type, countmod.nk_less].general_induction
requires the following assumptions to be discharged
  noetherian[countmod.nk_type, countmod.nk_less].well_founded

Use of the formula
  mid2.good_less_NF
requires the following TCCs to be proven
  mid2_tcc.good_greater_F1_TCC1
  mid2_tcc.good_less_NF_TCC1
  mid2_tcc.good_greater_F1_pr_TCC1
  mid2_tcc.good_less_NF_pr_TCC1

   ================== SUMMARY ==================

The proof chain is complete

The axioms and assumptions at the base are:
  clocksort.cnt_sort_geq
  clocksort.cnt_sort_leq
  division.mult_div_1
  division.mult_div_2
  division.mult_div_3
  ft_mid_assume.ft_mid_maxfaults
  multiplication.mult_non_neg
  multiplication.mult_pos
  noetherian[EXPR, EXPR].general_induction
Total: 9

The definitions and type-constraints are:
  absmod.abs
  clockassumptions.okay_Readpred
  clockassumptions.okay_pairs
  countmod.count
  countmod.countsize
  countmod.exists_less
  countmod.nk_noeth_pred
  countmod.nk_ph_pred
  mid.ft_mid
  multiplication.mult
```

```
 \naturalnumbers.nat_invariant
Total: 11

The formulae used are:
  absmod.abs_com
  absmod.abs_geq
  absmod.abs_plus
  countmod.count_exists
  countmod.count_exists_base
  countmod.count_exists_ind
  countmod.nk_ph_expand
  countmod.nk_ph_lem
  countmod.nk_ph_noeth_hyp
  countmod.ph_case0
  countmod.ph_case0k
  countmod.ph_case0n
  countmod.ph_case1
  countmod.ph_case2
  countmod.ph_case2a
  countmod.ph_case2b
  countmod.pigeon_hole
  countmod_tcc.count_TCC1
  countmod_tcc.count_TCC2
  countmod_tcc.count_TCC3
  countmod_tcc.count_TCC4
  countmod_tcc.count_TCC5
  countmod_tcc.posint_TCC1
  division.abs_div
  division.div_cancel
  division.div_distrib
  division.div_ineq
  division.div_minus_1
  division.div_minus_distrib
  division.div_nonnegative
  division.mult_div
  division.mult_minus
  division_tcc.ceil_mult_div_TCC1
  division_tcc.div_cancel_TCC1
  division_tcc.div_ineq_TCC1
division_tcc.div_minus_1_TCC1
division_tcc.div_nonnegative_TCC1
division_tcc.mult_div_1_TCC1
division_tcc.mult_div_TCC1
mid2.good_less_NF
mid2_tcc.good_greater_F1_TCC1
mid2_tcc.good_greater_F1_pr_TCC1
mid2_tcc.good_less_NF_TCC1
mid2_tcc.good_less_NF_pr_TCC1
mid3.exchange_order
mid3.ft_mid_prec_enh_sym
mid3.ft_mid_prec_sym1
mid3.good_between
mid3.good_geq_F_add1
mid3.mid_gt_imp_sel_gt
mid3.okay_pair_geq_F_add1
mid3.okay_pairs_sym
mid3_tcc.ft_mid_Pi_TCC1
mid3_tcc.ft_mid_prec_sym1_TCC1
mid3_tcc.ft_mid_prec_sym1_TCC2
mid3_tcc.ft_mid_prec_sym1_TCC3
mid3_tcc.ft_mid_prec_sym1_TCC4
mid3_tcc.ft_mid_prec_sym1_pr_TCC1
mid3_tcc.ft_mid_prec_sym1_pr_TCC2
mid3_tcc.good_between_TCC1
mid3_tcc.good_geq_F_add1_TCC1
mid3_tcc.mid_gt_imp_sel_gt_TCC1
mid3_tcc.mid_gt_imp_sel_gt_TCC2
mid3_tcc.okay_pair_geq_F_add1_TCC1
mid3_tcc.okay_pair_geq_F_add1_TCC2
mid_tcc.ft_mid_TCC1
mid_tcc.ft_mid_TCC2
mid_tcc.ft_mid_TCC3
multiplication.distrib
multiplication.distrib_minus
multiplication.mult_com
multiplication.mult_gt
multiplication.mult_ldistrib_minus
multiplication.mult_leq
```

```
  multiplication.mult_lident
  multiplication.mult_rident
  multiplication.pos_product
  natinduction.induction
  natinduction_tcc.ind_m_proof_TCC1
  noetherian[countmod.nk_type, countmod.nk_less].well_founded
  noetherian[naturalnumber, natinduction.less].well_founded
  select_defs.count_geq_select
  select_defs.count_leq_select
Total: 83

The completed proofs are:
  absmod.abs_com_proof
  absmod.abs_geq_proof
  absmod.abs_plus_pr
  countmod.count_exists_base_pr
  countmod.count_exists_ind_pr
  countmod.count_exists_pr
  countmod.nk_ph_expand_pr
  countmod.nk_ph_lem_pr
  countmod.nk_ph_noeth_hyp_pr
  countmod.nk_well_founded
  countmod.ph_case0_pr
  countmod.ph_case0k_pr
  countmod.ph_case0n_pr
  countmod.ph_case1_pr
  countmod.ph_case2_pr
  countmod.ph_case2a_pr
  countmod.ph_case2b_pr
  countmod.pigeon_hole_pr
  countmod_tcc.count_TCC1_PROOF
  countmod_tcc.count_TCC2_PROOF
  countmod_tcc.count_TCC3_PROOF
  division.abs_div_pr
  division.div_cancel_pr
  division.div_distrib_pr
  division.div_ineq_pr
  division.div_minus_1_pr
  division.div_minus_distrib_pr
division.div_nonnegative_pr
division.mult_div_pr
division.mult_minus_pr
division_tcc.ceil_mult_div_TCC1_PROOF
division_tcc.div_cancel_TCC1_PROOF
division_tcc.div_ineq_TCC1_PROOF
division_tcc.div_minus_1_TCC1_PROOF
division_tcc.div_nonnegative_TCC1_PROOF
division_tcc.mult_div_1_TCC1_PROOF
division_tcc.mult_div_TCC1_PROOF
mid2.good_less_NF_pr
mid2_tcc.good_greater_F1_TCC1_PROOF
mid2_tcc.good_greater_F1_pr_TCC1_PROOF
mid3.exchange_order_pr
mid3.ft_mid_prec_enh_sym_pr
mid3.ft_mid_prec_sym1_pr
mid3.ft_mid_precision_enhancement_pr
mid3.good_between_pr
mid3.good_geq_F_add1_pr
mid3.mid_gt_imp_sel_gt_pr
mid3.okay_pair_geq_F_add1_pr
mid3.okay_pairs_sym_pr
mid3_tcc.ft_mid_Pi_TCC1_PROOF
mid3_tcc.ft_mid_prec_sym1_TCC1_PROOF
mid3_tcc.ft_mid_prec_sym1_TCC3_PROOF
mid3_tcc.ft_mid_prec_sym1_pr_TCC1_PROOF
mid3_tcc.good_geq_F_add1_TCC1_PROOF
mid3_tcc.mid_gt_imp_sel_gt_TCC1_PROOF
mid3_tcc.okay_pair_geq_F_add1_TCC1_PROOF
mid3_tcc.okay_pair_geq_F_add1_TCC2_PROOF
mid_tcc.ft_mid_TCC1_PROOF
mid_tcc.ft_mid_TCC3_PROOF
mid_top.countmod_TCC4_pr
mid_top.countmod_TCC5_pr
mid_top.posint_TCC1_PROOF
multiplication.distrib_minus_pr
multiplication.distrib_proof
multiplication.mult_com_pr
multiplication.mult_gt_pr
```

```
  multiplication.mult_ldistrib_minus_proof
  multiplication.mult_leq_pr
  multiplication.mult_lident_proof
  multiplication.mult_rident_proof
  multiplication.pos_product_pr
  natinduction.discharge
  natinduction.ind_proof
  natinduction_tcc.ind_m_proof_TCC1_PROOF
  select_defs.count_geq_select_pr
  select_defs.count_leq_select_pr
  tcc_mid.ft_mid_TCC2_PROOF
  tcc_mid.ft_mid_prec_sym1_TCC2_PROOF
  tcc_mid.ft_mid_prec_sym1_TCC4_PROOF
  tcc_mid.ft_mid_prec_sym1_pr_TCC2_PROOF
  tcc_mid.good_between_TCC1_PROOF
  tcc_mid.good_less_NF_TCC1_PROOF
  tcc_mid.good_less_NF_pr_TCC1_PROOF
  tcc_mid.mid_gt_imp_sel_gt_TCC2_PROOF
Total: 84
```

C.3 Accuracy Preservation

```
 Terse proof chain for proof ft_mid_acc_pres_pr in module mid4

Use of the formula
  mid4.ft_mid_bnd_by_good
requires the following TCCs to be proven
  mid4_tcc.ft_mid_less_TCC1
  mid4_tcc.ft_mid_greater_TCC1

Use of the formula
  mid2.good_greater_F1
requires the following TCCs to be proven
  mid2_tcc.good_greater_F1_TCC1
  mid2_tcc.good_less_NF_TCC1
  mid2_tcc.good_greater_F1_pr_TCC1
  mid2_tcc.good_less_NF_pr_TCC1
```

```
Use of the formula
  countmod.count_exists
requires the following TCCs to be proven
  countmod_tcc.posint_TCC1
  countmod_tcc.count_TCC1
  countmod_tcc.count_TCC2
  countmod_tcc.count_TCC3
  countmod_tcc.count_TCC4
  countmod_tcc.count_TCC5

Formula countmod_tcc.count_TCC4 is a termination TCC for countmod.count
Proof of
  countmod_tcc.count_TCC4
must not use
  countmod.count

Formula countmod_tcc.count_TCC5 is a termination TCC for countmod.count
Proof of
  countmod_tcc.count_TCC5
must not use
  countmod.count

Use of the formula
  natinduction.induction
requires the following TCCs to be proven
  natinduction_tcc.ind_m_proof_TCC1

Use of the formula
  noetherian[naturalnumber, natinduction.less].general_induction
requires the following assumptions to be discharged
  noetherian[naturalnumber, natinduction.less].well_founded

Use of the formula
  noetherian[countmod.nk_type, countmod.nk_less].general_induction
requires the following assumptions to be discharged
  noetherian[countmod.nk_type, countmod.nk_less].well_founded

Use of the formula
```

```
  mid.ft_mid
requires the following TCCs to be proven
  mid_tcc.ft_mid_TCC1
  mid_tcc.ft_mid_TCC2
  mid_tcc.ft_mid_TCC3

Use of the formula
  division.div_ineq
requires the following TCCs to be proven
  division_tcc.mult_div_1_TCC1
  division_tcc.mult_div_TCC1
  division_tcc.div_cancel_TCC1
  division_tcc.ceil_mult_div_TCC1
  division_tcc.div_nonnegative_TCC1
  division_tcc.div_ineq_TCC1
  division_tcc.div_minus_1_TCC1

   ================== SUMMARY ==================

The proof chain is complete

The axioms and assumptions at the base are:
  clocksort.cnt_sort_geq
  clocksort.cnt_sort_leq
  clocksort.funsort_ax
  division.mult_div_1
  division.mult_div_2
  division.mult_div_3
  ft_mid_assume.ft_mid_maxfaults
  multiplication.mult_pos
  noetherian[EXPR, EXPR].general_induction
Total: 9

The definitions and type-constraints are:
  absmod.abs
  clockassumptions.okay_Readpred
  countmod.count
  countmod.countsize
  countmod.exists_less
  countmod.nk_noeth_pred
  countmod.nk_ph_pred
  mid.ft_mid
  multiplication.mult
  naturalnumbers.nat_invariant
Total: 10

The formulae used are:
  absmod.abs_com
  absmod.abs_geq
  countmod.count_exists
  countmod.count_exists_base
  countmod.count_exists_ind
  countmod.nk_ph_expand
  countmod.nk_ph_lem
  countmod.nk_ph_noeth_hyp
  countmod.ph_case0
  countmod.ph_case0k
  countmod.ph_case0n
  countmod.ph_case1
  countmod.ph_case2
  countmod.ph_case2a
  countmod.ph_case2b
  countmod.pigeon_hole
  countmod_tcc.count_TCC1
  countmod_tcc.count_TCC2
  countmod_tcc.count_TCC3
  countmod_tcc.count_TCC4
  countmod_tcc.count_TCC5
  countmod_tcc.posint_TCC1
  division.div_cancel
  division.div_ineq
  division.mult_div
  division_tcc.ceil_mult_div_TCC1
  division_tcc.div_cancel_TCC1
  division_tcc.div_ineq_TCC1
  division_tcc.div_minus_1_TCC1
  division_tcc.div_nonnegative_TCC1
  division_tcc.mult_div_1_TCC1
  division_tcc.mult_div_TCC1
  mid2.good_greater_F1
  mid2.good_less_NF
  mid2_tcc.good_greater_F1_TCC1
  mid2_tcc.good_greater_F1_pr_TCC1
  mid2_tcc.good_less_NF_TCC1
  mid2_tcc.good_less_NF_pr_TCC1
  mid4.abs_q_greater
  mid4.abs_q_less
  mid4.ft_mid_bnd_by_good
  mid4.ft_mid_greater
  mid4.ft_mid_less
  mid4.ft_select
  mid4.maxfaults_lem
  mid4.mult_hack
  mid4_tcc.ft_mid_greater_TCC1
  mid4_tcc.ft_mid_less_TCC1
  mid_tcc.ft_mid_TCC1
  mid_tcc.ft_mid_TCC2
  mid_tcc.ft_mid_TCC3
  multiplication.distrib_minus
  multiplication.mult_com
  multiplication.mult_gt
  multiplication.mult_ldistrib_minus
  multiplication.mult_lident
  multiplication.mult_rident
  natinduction.induction
  natinduction_tcc.ind_m_proof_TCC1
  noetherian[countmod.nk_type, countmod.nk_less].well_founded
  noetherian[naturalnumber, natinduction.less].well_founded
  select_defs.count_geq_select
  select_defs.count_leq_select
  select_defs.select_ax
Total: 64

The completed proofs are:
  absmod.abs_com_proof
  absmod.abs_geq_proof
  countmod.count_exists_base_pr
countmod.count_exists_ind_pr
countmod.count_exists_pr
countmod.nk_ph_expand_pr
countmod.nk_ph_lem_pr
countmod.nk_ph_noeth_hyp_pr
countmod.nk_well_founded
countmod.ph_case0_pr
countmod.ph_case0k_pr
countmod.ph_case0n_pr
countmod.ph_case1_pr
countmod.ph_case2_pr
countmod.ph_case2a_pr
countmod.ph_case2b_pr
countmod.pigeon_hole_pr
countmod_tcc.count_TCC1_PROOF
countmod_tcc.count_TCC2_PROOF
countmod_tcc.count_TCC3_PROOF
division.div_cancel_pr
division.div_ineq_pr
division.mult_div_pr
division_tcc.ceil_mult_div_TCC1_PROOF
division_tcc.div_cancel_TCC1_PROOF
division_tcc.div_ineq_TCC1_PROOF
division_tcc.div_minus_1_TCC1_PROOF
division_tcc.div_nonnegative_TCC1_PROOF
division_tcc.mult_div_1_TCC1_PROOF
division_tcc.mult_div_TCC1_PROOF
mid2.good_greater_F1_pr
mid2.good_less_NF_pr
mid2_tcc.good_greater_F1_TCC1_PROOF
mid2_tcc.good_greater_F1_pr_TCC1_PROOF
mid4.abs_q_greater_pr
mid4.abs_q_less_pr
mid4.ft_mid_acc_pres_pr
mid4.ft_mid_bnd_by_good_pr
mid4.ft_mid_greater_pr
mid4.ft_mid_less_pr
mid4.ft_select_pr
mid4.maxfaults_lem_pr
```

```
  mid4.mult_hack_pr
  mid4_tcc.ft_mid_less_TCC1_PROOF
  mid_tcc.ft_mid_TCC1_PROOF
  mid_tcc.ft_mid_TCC3_PROOF
  mid_top.countmod_TCC4_pr
  mid_top.countmod_TCC5_pr
  mid_top.posint_TCC1_PROOF
  multiplication.distrib_minus_pr
  multiplication.mult_com_pr
  multiplication.mult_gt_pr
  multiplication.mult_ldistrib_minus_proof
  multiplication.mult_lident_proof
  multiplication.mult_rident_proof
  natinduction.discharge
  natinduction.ind_proof
  natinduction_tcc.ind_m_proof_TCC1_PROOF
  select_defs.count_geq_select_pr
  select_defs.count_leq_select_pr
  select_defs.select_ax_pr
  tcc_mid.ft_mid_TCC2_PROOF
  tcc_mid.ft_mid_greater_TCC1_PROOF
  tcc_mid.good_less_NF_TCC1_PROOF
  tcc_mid.good_less_NF_pr_TCC1_PROOF
Total: 65
```

References

[1] Schneider, Fred B.: *Understanding Protocols for Byzantine Clock Synchronization.* Department of Computer Science, Cornell University, Technical Report 87-859, Ithaca, NY, Aug. 1987.

[2] Shankar, Natarajan: *Mechanical Verification of a Schematic Byzantine Clock Synchronization Algorithm.* NASA, Contractor Report 4386, July 1991.

[3] Rushby, John; von Henke, Friedrich; and Owre, Sam: *An Introduction to Formal Specification and Verification Using* EHDM. Computer Science Laboratory, SRI International, Technical Report SRI-CSL-91-2, Menlo Park, CA, Feb. 1991.

[4] Di Vito, Ben L.; Butler, Ricky W.; and Caldwell, James L.: *Formal Design and Verification of a Reliable Computing Platform For Real-Time Control: Phase 1 Results.* NASA, Technical Memorandum 102716, Langley Research Center, Hampton, VA, Oct. 1990.

[5] Butler, Ricky W.; and Di Vito, Ben L.: *Formal Design and Verification of a Reliable Computing Platform For Real-Time Control: Phase 2 Results.* NASA, Technical Memorandum 104196, Langley Research Center, Hampton, VA, Jan. 1992.

[6] Rushby, John: *Formal Specification and Verification of a Fault-Masking and Transient-Recovery Model for Digital Flight-Control Systems.* NASA, Contractor Report 4384, July 1991.

[7] FAA: *System Design and Analysis.* U.S. Department of Transportation, Advisory Circular AC 25.1309-1A, June 1988.

[8] U.S. Department of Defense. *Reliability Prediction of Electronic Equipment,* Jan. 1982. MIL-HDBK-217D.

[9] Lamport, Leslie; and Melliar-Smith, P.M.: Synchronizing Clocks in the Presence of Faults. *Journal of the ACM,* vol. 21, Jan. 1985, pp. 52–78.

[10] Rushby, John; and von Henke, Friedrich: *Formal Verification of a Fault Tolerant Clock Synchronization Algorithm.* NASA, Contractor Report 4239, June 1989.

[11] Welch, J. Lundelius; and Lynch, N.: A New Fault-Tolerant Algorithm for Clock Synchonization. *Information and Computation*, vol. 77, no. 1, Apr. 1988, pp. 1–36.

[12] Srikanth, T.K.; and Toueg, S.: Optimal Clock Synchronization. *Journal of the ACM*, vol. 34, no. 3, July 1987, pp. 626–645.

[13] Halpern, J.; Simons, B.; Strong, R.; and Dolev, D.: Fault-Tolerant Clock Synchonization. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*. ACM, Aug. 1984, pp. 89–102.

[14] Kieckhafer, R.M.; Walter, C.J.; Finn, A.M.; and Thambidurai, P.: The MAFT Architecture for Distributed Fault Tolerance. *IEEE Transactions on Computers*, vol. 37, no. 4, Apr. 1988, pp. 398–405.

[15] Gouda, M.G.; and Multari, N.J.: Stabilizing Communication Protocols. *IEEE Transactions on Computers*, vol. 40, no. 4, Apr. 1991, pp. 448–458.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1992	3. REPORT TYPE AND DATES COVERED Technical Memorandum

4. TITLE AND SUBTITLE A Verified Design of a Fault-Tolerant Clock Synchronization Circuit: Preliminary Investigations	5. FUNDING NUMBERS WU 505-64-10-05
6. AUTHOR(S) Paul S. Miner	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23665-5225	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA TM-107568
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified-Unlimited Subject Category 59	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Schneider [1] demonstrates that many fault-tolerant clock synchronization algorithms can be represented as refinements of a single proven correct paradigm. Shankar [2] provides a mechanical proof (using Ehdm [3]) that Schneider's schema achieves Byzantine fault-tolerant clock synchronization provided that 11 constraints are satisfied. Some of the constraints are assumptions about physical properties of the system and cannot be established formally. Proofs are given (in Ehdm) that the fault-tolerant midpoint convergence function satisfies three of these constraints. This paper presents a hardware design, implementing the fault-tolerant midpoint function, which will be shown to satisfy the remaining constraints. The synchronization circuit will recover completely from transient faults provided the maximum fault assumption is not violated. The initialization protocol for the circuit also provides a recovery mechanism from total system failure caused by correlated transient faults.

14. SUBJECT TERMS Clock Synchronization, Fault Tolerance, Formal Methods, Transient Faults	15. NUMBER OF PAGES 100
	16. PRICE CODE A05

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT