Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Συστημάτων

Managing Dependability and Temperature in the Presence of Performance Variability for Embedded Systems

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Ρ. Ζαμπέλης

Επιβλέπων : Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Απρίλιος 2019

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Συστημάτων

Managing Dependability and Temperature in the Presence of Performance Variability for Embedded Systems

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Ρ. Ζαμπέλης

Επιβλέπων : Δημήτριος Σούντρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 17η Απριλίου 2019.

Αθήνα, Απρίλιος 2019

....

Νικόλαος Ρ. Ζαμπέλης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Τα τελευταία χρόνια, η αξιοπιστία των ψηφιακών συστημάτων απειλείται εν μέρει εξαιτίας των εξαιρετικά δυναμικών φόρτων εργασίας, των δεδομένων εισόδου των χρηστών και των ατελειών του περιβάλλοντος και του υλικού που είναι γνωστές ως κατασκευαστική μεταβλητότητα. Η τελευταία προκαλεί ανησυχίες για την αξιόπιστη λειτουργία του συστήματος, καθώς μπορεί να δημιουργήσει σφάλματα στο στρώμα υλικού που μπορεί να προκαλέσουν δυαδικά λάθη και να επηρεάσουν τη συνολική απόδοση του τσιπ. Για να μετριαστεί αυτή η μεταβλητότητα, η βιομηχανία έχει αναπτύξει μια σειρά τεχνικών αξιοπιστίας, διαθεσιμότητας και συντήρησης (RAS). Αυτές οι τεχνικές, μεταξύ άλλων, αντισταθμίζουν τις επιδράσεις της κατασκευαστικής μεταβλητότητας και εξασφαλίζουν αξιόπιστες επιδόσεις ανταλλάσοντας ισχύ, περιοχή πυριτίου ή χρόνο εκτέλεσης.

Σε αυτή τη διατριβή παρουσιάζουμε έναν ελεγκτή PID που θα εκτελεί δυναμικές αλλαγές τάσης και συχνότητας (DVFS) για να αντισταθμίσει τις επιπτώσεις των καθυστερήσεων επαναφοράς που προκαλούνται από τον RAS μηχανισμό και να διαχειριστεί τις χρονικές προθεσμίες. Επιπλέον, η αποτελεσματικότητα του ελεγκτή μας θα δοκιμαστεί όταν μια άλλη ταυτόχρονη εφαρμογή τρέχει στον ίδιο επεξεργαστή, προσομοιάζοντας μια άλλη περίπτωση δυναμισμού φόρτου εργασίας. Συγκρίνουμε τη μεθοδολογία μας με τους εξέχοντες αλγόριθμους DVFS τελευταίας τεχνολογίας και αποδεικνύουμε τις βελτιωμένες επιδόσεις του PID μειώνοντας παράλληλα την κατανάλωση ενέργειας. Τέλος, παρουσιάζουμε μια έκδοση του ελεγκτή PID που στοχεύει στη διαχείριση της θερμοκρασίας του συστήματος, καθώς οι θερμοκρασιακοί περιορισμοί είναι πρωταρχικής σημασίας στα σύγχρονα ενσωματωμένα συστήματα.

Το πρόγραμμά μας αναπτύσσεται στην πλακέτα iMX6Q-SABRE-SD από την NXP, ενώ επικεντρωνόμαστε σε μια ρεαλιστική εφαρμογή σε πραγματικό χρόνο. Αναλυτικά, η εφαρμογή μας προέρχεται από τον τομέα των τηλεπικοινωνιών και αναμένεται να εκτελεστεί σε τυπικές ενσωματωμένες πλατφόρμες. Επομένως, συζητάμε έναν ελεγκτή κλειστού βρόχου που χρησιμοποιεί το DVFS για να περιρρίσει τη διακύμανση της απόδοσης χρησιμοποιώντας μια ρεαλιστική εφαρμογή με χρονικούς περιορισμούς.

Λέξεις κλειδιά

Ενσωματωμένα Συστήματα, PID, Αξιοπιστία, Διαθεσιμότητα, Λειτουργικότητα, μηχανισμοί RAS, Κατασκευαστική Μεταβλητότητα, Μεταβλητότητα της απόδοσης, Δυναμική Κλιμάκωση Τάσης και Συχνότητας, DVFS, ARM, Συστήματα Πολλών Επεξεργαστών

Abstract

In recent years, dependability of digital systems has been threatened partly due to highly dynamic workloads, user input data, environment and hardware imperfections known as process variability. The latter causes concerns for the reliable operation of the system as it can generate faults in the hardware layer that may cause binary errors and affect the overall performance of the chip. To mitigate this variation, the industry has developed a series of Reliability, Availability and Serviceability (RAS) techniques. These techniques, among others, counter the effects of process variability and ensure dependable performance by trading-off either power, silicon area or execution time.

In this thesis we present a PID controller that will perform Dynamic Voltage and Frequency Scaling (DVFS) switches to counter the effects of RAS-induced rollback delays and manage timing deadlines. In addition, the efficiency of our controller will be tested when another concurrent application is running on the same CPU, simulating another case of workload dynamism. We compare our methodology with prominent state-of-the-art DVFS algorithms and prove the PID's enhanced performance while minimizing energy consumption. Finally, we present a version of the PID controller that aims to manage the system's temperature, as meeting temperature constraints is of paramount importance in modern embedded systems.

Our scheme is deployed on the iMX6Q-SABRE-SD board from NXP while we focus on a realistic, real-time application that is streaming in nature. In detail, our application is drawn from the telecommunication domain and is expected to run on typical embedded platforms. Therefore, we discuss a closed-loop controller that utilizes DVFS to account for performance variation using a realistic application with timing deadlines.

Key words

PID, Reliability, RAS mechanisms, Process Variation, performance variability, DVFS, iMX6Q Board

Acknowledgements

First of all, I would like to wholeheartedly thank Professor Dimitrios Soudris for giving me the opportunity to work on this thesis. His valuable advice helped me immensely in this work and his research experience and passionate teaching in my undergraduate classes had a direct impact in developing my interest for embedded systems.

Moreover, this thesis would not be possible without the help and support of Michail Noltsis. His constructive advice and experience in the subject matter urged me to work on this thesis and create significant results that we are proud to present to the public. His insight definitely peaked my interest in working with embedded systems and the scientific community in general.

Finally, I would like to thank my family and friends for their continuous support throughout my studies in NTUA.

Nikolaos R. Zampelis,

Athens, April 17, 2019

Εκτενής Περίληψη

Η βιομηχανία κατασκευής ολοκληρωμένων κυκλωμάτων έχει συμμορφωθεί με την σμίκρυνση της τεχνολογίας παραγωγής που περιγράφει ο νόμος του Moore για τα τελευταία 40 χρόνια και βελτίωσε την απόδοση των ενσωματωμένων συστημάτων δραστικά. Καθώς τα μικρότερα τρανζίστορ και τα καλώδια σύνδεσης τους είναι συσκευασμένα σε ολοένα και μικρότερες περιοχές πυριτίου, η συνολική απόδοση του ολοκληρωμένου αυξάνεται λόγω των περισσότερων αριθμητικών λειτουργιών που πραγματοποιούνται ανά λεπτό και της λιγότερης ενέργειας που απαιτείται για την τροφοδοσία του. Την ίδια στιγμή, το κόστος παραγωγής χιλιάδων πλακιδίων πυριτίου που το καθένα περιέχει δισεκατομμύρια τρανζίστορ σε μια ημέρα έχει γίνει όλο και μικρότερο, πράγμα που οδηγεί σε μια ανερχόμενη βιομηχανία τσιπ που έχει διαμορφώσει τον 21ο αιώνα όπως τον ξέρουμε σήμερα.

Αυτή η σμίκρυνση της τεχνολογίας παραγωγής, ωστόσο, αντιμετωπίζει σήμερα σημαντικές προκλήσεις που πρέπει να αντιμετωπιστούν. Η απαίτηση για όλο και πιο γρήγορα και αξιόπιστα τσιπ, ικανά να τρέχουν όλο και πιο απαιτητικές εφαρμογές για τους καταναλωτές, βρίσκει αντιμέτωπα τα προβλήματα της κατασκευαστικής μεταβλητότητας, της διάχυσης της θερμοκρασίας και της κατανάλωσης ενέργειας[1]. Ο όρος ”κατασκευαστική μεταβλητότητα” περιγράφει το φαινόμενο κατά το οποίο δύο τρανζίστορ ομοίως σχεδιασμένα και κατασκευασμένα μερικά νανόμετρα μακριά μπορεί να εμφανίσουν διαφορετικά ηλεκτρικά και δομικά χαρακτηριστικά εξαιτίας αναπόφευκτων αιτιών[2][3]. Κατασκευαστική μεταβλητότητα υπήρχε ανέκαθεν στην παραγωγική διαδικασία[2] αλλά τα τελευταία χρόνια, από τότε που οι διαστάσεις των τρανζίστορ έφτασαν τον κόμβο των 90 νανομέτρων, προκαλεί όχι μόνο απώλειες στην ποσότητα παραγωγής (αφού πολλά δεν ικανοποιούν το σχεδιαστικά χαρακτηριστικά) αλλά και προκαλεί διακύμανση στην εύρυθμη λειτουργία των ολοκληρωμένων[3].

Επιπλέον, η απόδοση των σύγχρονων ενσωματωμένων συστημάτων επηρεάζεται από τη δυναμική φύση του του φόρτου εργασίας τους. Σήμερα, οι εφαρμογές ενεργούν με ιδιαίτερα δυναμικό τρόπο ως προς τους πόρους συστήματος που απαιτούν και τους χρονικούς στόχους τους. Αυτός ο δυναμισμός είναι το αποτέλεσμα τόσο σφαλμάτων που σχετίζονται με το υλικό [2] όσο και απαιτήσεων του λογισμικού όπως οι εφαρμογές βαρέων υπολογισμών και η μεταβλητότητα στο είδος και τη συχνότητα των δεδομένων εισόδου. Επομένως, καθίσταται όλο και πιο εμφανές ότι αυτός ο δυναμισμός επηρεάζει τις επιδόσεις του συστήματος και πρέπει να ληφθεί υπόψη κατά τον σχεδιασμό των περιορισμών του συστήματος ως προς την ενέργεια και τις χρονικές αποκρίσεις.

Επιπρόσθετα, ενώ η βιομηχανία δυσκολεύεται να περιορίσει τις διαστάσεις των τρανζίστορ περαιτέρω, το όριο της τάσης που απαιτούν αυτές οι συσκευές έχει σταματήσει να μειώνεται, όπως πρότεινε αρχικά το Μοντέλο Κλιμάκωσης του Dennard, με αποτέλεσμα μια δραστική αύξηση της κατανάλωσης ενέργειας και της θερμοκρασίας του συστήματος που απειλεί την ομαλή απόδοση του ολοκληρωμένου. Και ενώ στους επιτραπέζιους υπολογιστές και τους διακομιστές τα τσιπ ψύχονται δαπανόντας ενέργεια σε ανεμιστήρες και άλλες τεχνικές ενεργής ψύξης, η επιλογή αυτή δεν είναι διαθέσιμη σε μικρές συσκευές όπως τα smartphones, τα tablets και άλλα μικρά Systems-on-a-Chip (SoC) από το πεδίο των ενσωματωμένων συστημάτων. Η βιομηχανία, τελικά, αναγκάστηκε να ανεχτεί το φαινόμενο Dark Silicon: την αναγκαιότητα που υπαγορεύει ότι μόνο ένα τμήμα των τρανζίστορ του τσιπ θα τροφοδοτείται το ίδιο χρονικό διάστημα προκειμένου το σύστημα να περιορίζεται σε αυστηρούς προϋπολογισμούς ισχύος και θερμοκρασίας[4].

Παράλληλα με την αντιμετώπιση των προαναφερόμενων απειλών, η βιομηχανία των ενσωματωμένων συστημάτων πρέπει επίσης να είναι σε θέση να διασφαλίσει την αξιοπιστία του συστήματος. Ένα ενσωματωμένο σύστημα αναμένεται,ως επί τω πλείστον, να ανταποκρίνεται σε πραγματικό χρόνο και να πετυχαίνει τους στόχους του αποφεύγοντας τους κινδύνους που παρουσιάστηκαν παραπάνω. Γενίκευση αυτού είναι η απαίτηση τα ενσωματωμένα συστήματα να τηρούν τις προθεσμίες που καθορίζονται a priori, εξασφαλίζοντας παράλληλα την ορθότητα του συστήματος. Συγκεκριμένα, για να διασφαλιστεί η σωστή λειτουργία του συστήματος, ο κλάδος έχει αναπτύξει πολυάριθμες τεχνικές Αξιοπιστίας, Διαθεσιμότητας και Λειτουργικότητας (Reliability, Availability and Serviceability RAS)[5].

Προς αυτή την κατεύθυνση, σε αυτή τη διατριβή παρουσιάζουμε έναν ελεγκτή PID που στοχεύει στην αντιμετώπιση των αποτελεσμάτων του δυναμικών φόρτων εργασίας και των παρεμβάσεων RAS χρησιμοποιώντας την τεχνική της Δυναμικής Κλιμάκωσης Τάσης και Συχνότητας (Dynamic Voltage and Frequency Scaling DVFS). Επιπλέον, το πρόγραμμά μας λαμβάνει υπόψιν τη διαχείριση της θερμοκρασίας του συστήματος σε κανονικά επίπεδα. Η εφαρμογή βασίζεται στο έργο HARPA της Ευρωπαϊκής Ένωσης και όλος ο βρόχος αισθητήρων-εφαρμογής-ελεγκτή αναπτύσσεται σε πειραματικό υλικό, συγκεκριμένα στην πλακέτα NXP iMX6Q-SABER-SD[6].

Μεταβλητότητα στη διαδικασία παραγωγής

Μία σημαντική πηγή δυναμικής απόκρισης στη λειτουργία του συστήματος, που δημιουργεί ελαττωματικό υλικού είναι η κατασκευαστική μεταβλητότητα. Χωρίζεται σε δύο μεγάλες κατηγορίες: ενιαία ή συστηματική μεταβλητότητα και τυχαία μεταβλητότητα επίσης γνωστή ως παραγωγική μεταβλητότητα [2]. Ενώ και οι δύο κατηγορίες συνήθως προκαλούν παροδικά σφάλματα και επηρεάζουν τα ίδια ηλεκτρικά χαρακτηριστικά του τρανζίστορ, δηλαδή το ρεύμα διαρροής, τη συχνότητα λειτουργίας και την τάση κατωφλίου V_{th}, εκδηλώνονται μέσω διαφορετικών πηγών και συνεπώς χρειάζονται διαφορετικές μεθόδους για να αντιμετωπιστούν. Από τη μια πλευρά, η ενιαία μεταβλητότητα οφείλεται σε μικρές αλλαγές στο περιβάλλον της διαδικασία παραγωγής και επηρεάζει όλα τα τρανζίστορ σε ένα δίσκο με τον ίδιο τρόπο. Οι κύριες πηγές συστηματικής μεταβλητότητας είναι η οπτική διόρθωση εγγύτητας (OPC)[7], η τραχύτητα μήκους ακμής (LER) και η τραχύτητα πλάτους ακμής (LWR)[8][9] και παραλλαγές στο διηλεκτρικό πύλης [10] [2]. Για παράδειγμα μια μικρή απόκλιση στην οπτική του φωτός ενός εργαλείου λιθογραφίας μπορεί να προκαλέσει ένα ολόκληρο πλακίδιο να έχει τρανζίστορ με μικρότερο ή μεγαλύτερο πλάτος καναλιού από ότι σχεδιάστηκε αρχικά. Αυτός ο τύπος μεταβλητότητας υπήρξε πάντοτε στην παραγωγική διαδικασία και η βιομηχανία έχει αναπτύξει με επιτυχία πολλούς τρόπους αντιμετώπισης της αποτελέσματα[11].

Από την άλλη πλευρά, η παραγωγική μεταβλητότητα παρουσιάστηκε μόνο από τότε που η βιομηχανία έφθασε στον κατασκευαστικό κόμβο των 90 nm. Ενώ η μείωση των τρανζίστορ συνεχίστηκε, οι επιπτώσεις της κατασκευαστικής μεταβλητότητας έχουν αποδειχθεί δύσκολες να αντιμετωπιστούν, καθώς τα ελαττώματα υλικού που τις προκαλούν συνδέονται άμεσα με τις μικρές διαστάσεις των τρανζίστορ. Πρέπει να σημειωθεί ότι η κατασκευαστική μεταβλητότητα μπορεί να χωριστεί περαιτέρω σε δύο κατηγορίες: εξ-αρχής μεταβλητότητα, δηλαδή μεταβλητότητα που προκαλείται από εσωτερικές πηγές κατά το στάδιο κατασκευής της διάταξης πυριτίου, και χρονικά εξαρτώμενη μεταβλητότητα που προκαλείται και ενισχύεται από εξωτερικές πηγές όπως η θερμοκρασία λειτουργίας, η υγρασία και ο εργασιακός φόρτος της συσκευής.

Μεταξύ άλλων, μία σημαντική αιτία εξ-αρχής μεταβλητότητας είναι οι τυχαίες διακυμάνσεις εμφυτεύσεων (Random Dopant Fluctuations RDF)[12]. Εδώ και δεκαετίες η βιομηχανία εμφυτεύει τις πύλες των τρανζίστορ MOS με άτομα προσμίξεων έτσι ώστε να ελέγχεται αποτελεσματικά η τάση κατωφλίου V_{th}. Και ενώ σε παλαιότερους τεχνολογικούς κόμβους οι διαστάσεις της πύλης ήταν αρκετά μεγάλες ώστε η συγκέντρωση των προσμίξεων να είναι αμελητέα, πλέον συναντάμε ένα φαινόμενο όπου ο αριθμός των ατόμων που έχουν προσμιχθεί και οι διαστάσεις της πύλης είναι αρκετά μικροί ώστε να μην ισχύει πλέον ο νόμος των μεγάλων αριθμών. Επομένως, η ποσόστωση των ατόμων προσμίξεων μπορεί να προκαλέσει δύο τρανζίστορ, το ένα δίπλα στο άλλο, να έχουν διαφορετικές τάσεις κατωφλίου και επομένως να λειτουργούν διαφορετικά ή να καταναλώνουν περισσότερη ενέργεια.

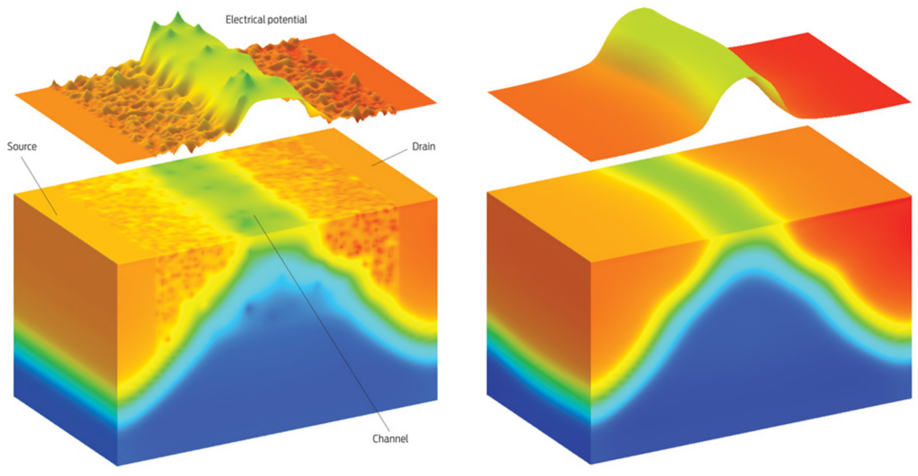

Figure 0.1: RDF η τραχύτητα σε ένα MOS τρανζίστορ μπορούν να εισάγουν μεταβολές στην τάση κατωφλίου του[11]

Επιπλέον, η συνεχής λειτουργία των τρανζίστορ, ειδικά σε συστήματα πραγματικού χρόνου, ενεργοποιεί μηχανισμούς γήρανσης που επηρεάζουν την αξιοπιστία του τσιπ και οδηγούν σε υποβάθμιση της απόδοσης. Αυτές οι επιπτώσεις γίνονται όλο και πιο αισθητές καθώς αυξάνεται η θερμοκρασία λειτουργίας του συστήματος. Κάποια αξιοσημείωτα φαινόμενα που προκαλούν χρονικά εξαρτώμενη μεταβλητότητα και συνδέονται στενά με αυξημένες θερμοκρασίες είναι η Εισροή Θερμών Φορέων (Hot Carrier Injection HCI)[13][14], η Χρονικά-Εξαρτώμενη Καταστροφή Διηλεκτρικού (Time-Dependent Dielectric Breakdown TDDB) και, κυρίως, Κατα-Θερμοκρασία Αστάθεια της Τάσης Κατωφλιού (Bias Temperature Instability BTI)[15][16][17][18]. Ενώ αυτά τα προβλήματα επηρεάζουν κυρίως το οξείδιο της πύλης του τρανζίστορ, και τελικά το V_{th} του, υπάρχουν επίσης φαινόμενα που προκαλούν χρονικά-εξαρτώμενη μεταβλητότητα των διασυνδετικών συρμάτων των τρανζίστορ όπως η Ηλεκτρομεταφορά (Electromigration EM)[19] και Αυτοθέρμανση [13][20].

Δεδομένου ότι η BTI έχει αποδειχθεί ο πιο σημαντικός παράγοντας της χρονικά εξαρτώμενης μεταβλητότητας, η ερευνητική κοινότητα έχει αναπτύξει δύο βασικές θεωρίες για να την εξηγήσει. Η πρώτη και η παλαιότερη θεωρία είναι το μοντέλο Διάχυσης-Αντίδρασης (Reaction-Diffusion RD) [21][22]. Αυτό το μοντέλο υποδηλώνει ότι η BTI μπορεί να εξηγηθεί από τη θραύση των Si-H δεσμών στη διεπιφάνεια Si / οξειδίου. Ενώ τα άτομα υδρογόνου διαχέονται στην πύλη, η μειονότητα των φορέων, επίσης γνωστή ως ”τρύπες”, κινούνται για να πάρουν τη θέση τους στο δίκτυο δημιουργώντας έτσι παγίδες φορτίου που μετακινούν το V_{th} του τρανζίστορ [15]. Ωστόσο, αυτή η προσέγγιση δεν έχει περιγράψει πλήρως το πρόβλημα και σήμερα υποκαθίσταται με μια πιο καινοτόμο θεωρία που επικεντρώνεται σε ατέλειες. Αυτό είναι το ατομικό μοντέλο που εισάγει ένα πιθανοτικό στοιχείο στην προσπάθεια εξήγησης των επιπτώσεων των σχηματιζόμενων ελαττωμάτων. Αντιφάσκει με το μοντέλο RD, καθώς προτείνει ότι οι παγίδες που δημιουργούνται από ελαττώματα μπορούν επίσης να εκκενωθούν και όχι απαραίτητα να φορτιστούν και μόνο οι φορτισμένες μετατοπίζουν το V_{th} [21].

Τέλος πρέπει να αναφέρουμε τους, όχι λιγότερο σημαντικούς, μηχανισμούς που οδηγούν σε μεταβλητότητα της απόδοσης του συστήματος. Αυτά είναι τα σωματίδια κοσμικής και επίγειας ακτινοβολίας, όπως τα σωματίδια άλφα και η κοσμική ακτίνες υψηλής ενέργειας [2][20]. Οι διαστάσεις των τρανζίστορ είναι τώρα αρκετά μικρές ώστε σωματίδια ακτινοβολίας, που χτυπούν τυχαία πάνω τους, μπορούν να αυξήσουν ή να μειώσουν την τάση τους για ένα αρκετά σημαντικό χρονικό διάστημα ώστε το δημιουργούμενο σφάλμα να διαδίδεται σε άλλα τρανζίστορ και τελικά να τέθει σε κίνδυνο η σωστή λειτουργία τουλάχιστον ενός υποσυστήματος του τσιπ. Αυτό το φαινόμενο είναι συνήθως εμφανές στα συστήματα μνήμης και αναφέρεται ως ”bit flip”, επειδή το σωματίδιο ακτινοβολίας μπορεί να αναστρέψει ένα bit μνήμης από 0 σε 1 και το αντίστροφο και να αναγκάσει το σύστημα να διαβάσει λανθασμένα δεδομένα μνήμης. Κανονικά το bit θα γυρίσει πίσω στη σωστή του κατάσταση και το σφάλμα θα αντιστραφεί χωρίς να προκαλέσει περαιτέρω βλάβη. Ωστόσο, εάν η ακτινοβολία προκαλέσει διαταραχή της φόρτισης σε ένα κρίσιμο σύστημα, μπορεί να παρουσιαστεί δυσλειτουργία όλου του προϊόντος[23].

Μέθοδοι διαχείρισης ενέργειας και θερμοκρασίας

Διαπιστώνεται, επομένως, ότι η κατασκευαστική μεταβλητότητα επιδεινώνεται από την αύξηση των θερμοκρασιών του συστήματος. Επιπλέον, η κατανάλωση ενέργειας εξαρτάται επίσης σε μεγάλο βαθμό από τη θερμοκρασία. Στην πραγματικότητα, η στατική ισχύς, ένα σημαντικό στοιχείο της συνολικής κατανάλωσης ενέργειας που ήταν αμελητέα πριν από 10 χρόνια, έχει μια εκθετική σχέση με τη θερμοκρασία του συστήματος [24]. Επομένως, η διαχείριση της θερμοκρασίας του συστήματος και η διατήρησή της σε ασφαλή επίπεδα είναι ζωτικής σημασίας για την βιομηχανία ενσωματωμένων συστημάτων, ειδικά αφού το Μοντέλο Κλιμάκωσης του Dennard σταμάτησε να εφαρμόζεται στα τρανζίστορ. Και ενώ οι επιτραπέζιοι υπολογιστές και οι φορητοί υπολογιστές χρησιμοποιούν τους ανεμιστήρες και άλλες τεχνολογίες ψύξης για την ψύξη των επεξεργαστών, η επιλογή αυτή δεν είναι διαθέσιμη για συσκευές χειρός. Τελικά, χρειαζόμαστε να αναζητήσουμε αλλού την αποτελεσματική διαχείριση των θερμοκρασιών.

Μεταξύ άλλων τεχνικών που χρησιμοποιούνται σε μεγάλο βαθμό, η Δυναμική Κλιμάκωση Τάσης και Συχνότητας (DVFS) είναι μια ευρέως γνωστή τεχνική για τη διαχείριση της καταναλισκόμενης ισχύος με τη μεταγωγή της τάσης και της συχνότητας λειτουργίας του σύστηματος. Η δυναμική ισχύς έχει τετραγωνική σχέση με την εφαρμοζόμενη τάση πολλαπλασιασμένη με τη συχνότητα λειτουργίας, επομένως επηρεάζεται άμεσα και ελέγχεται σε μεγάλο βαθμό από το DVFS. Επιπλέον, η στατική ισχύς, που καταναλώνεται από τα ρεύματα διαρροής της πύλης του τρανζίστορ, έχει μια γραμμική σχέση με την εφαρμοζόμενη τάση αλλά μια εκθετική σχέση με τη θερμοκρασία του συστήματος που επηρεάζει δραστικά την στατική ισχύ [24]. Ο ευαίσθητος έλεγχος που επιτυγχάνεται με το DVFS είναι ζωτικής σημασίας για τα ενσωματωμένα συστήματα που πρέπει να εξοικονομούν ταυτόχρονα όσο το δυνατόν περισσότερη ενέργεια και να εξασφαλίζουν αξιόπιστες επιδόσεις. Ένα ενδιαφέρον έργο που βασίζεται στα οφέλη του DVFS παρουσιάζεται στο [25]. Η AMD και η Intel έχουν επίσης εισαγάγει μεθοδολογίες για τον έλεγχο της ισχύος με τις ”Cool’n’Quiet” και ”SpeedStep” τεχνολογίες αντίστοιχα [26][27]. Την ίδια στιγμή, ο πυρήνας των Linux περιέχει τον ελεγκτή CPUFreq που αποτελεί καλό παραδείγμα για ενέργειες DVFS, ειδικά στις ”On demand” και ”Conservative” λειτουργίες του [28]. Για την ακρίβεια, θα συγκρίνουμε τον PID ελεγκτή μας με μια παραλλαγή του «συντηρητικού» ελεγκτή τόσο ως προς την απόδοση όσο και ως προς την κατανάλωση ενέργειας.

Όσον αφορά την αποτελεσματική διαχείριση της θερμοκρασίας, η έρευνα έχει επικεντρωθεί σε προγράμματα καταμερισμού εργασιών και αναδιανομή φόρτου εργασίας σε πολλαπλούς πυρήνες προκειμένου να ικανοποιηθούν οι θερμικές προδιαγραφές σχεδίασης [29][30][31]. Το DVFS παίζει επίσης σημαντικό ρόλο, καθώς η ρύθμιση της τάσης του συστήματος σε χαμηλότερα επίπεδα μπορεί να έχει σημαντικές επιδράσεις στη θερμοκρασία του συστήματος [3][32]. Ένας μηχανισμός πρόβλεψης θερμοκρασίας αναφέρεται στη βιβλιογραφία [33], καθώς και ένας ελεγκτή PI για τη θερμική διαχείριση [34] και ένας αλγόριθμος θεμροκρασιακής διαχείρισης που χρησιμοποιεί τους μηχανισμούς CPUFreq του Linux σε ARM επεξεργαστή [35]. Θα συγκρίνουμε τις προσπάθειές μας με αυτόν τον συγκεκριμένο αλγορίθμο στο Κεφάλαιο 4. Όσον αφορά σχετικές εργασίες που έγιναν από τη βιομηχανία, τα Intel Pentium 4 και οι επεξεργαστές της IBM PowerPC έχουν ήδη ενσωματώσει λειτουργίες διαχείρισης θερμοκρασίας [36][37].

PID ελεγκτής για ελαχιστοποίηση των επιπτώσεων της μεταβλητότητας

Στην παρούσα εργασία θα παρουσιάσουμε μια εφαρμογή του ελεγκτή PID που θα μετριάσει τις επιπτώσεις της κατασκευαστικής μεταβλητότητας όπως αυτή προσομοιάζεται με σφάλματα εισαγόμενα απο το χρήστη. Κάθε φορά που ο χρήστης εισάγει ένα σφάλμα στο σύστημα, ο ελεγκτής καταλαβαίνει ότι, λόγω κατασκευαστικής μεταβλητότητας, έχει γίνει κάποιος λάθος υπολογισμός. Επομένως, φροντίζει να επιστρέφει στην τελευταία ασφαλή κατάσταση όπου οι υπολογισμοί ήταν σωστοί. Αυτή η επαναφορά είναι ένας RAS μηχανισμός γνωστός ως rollback. Ακόμη, ο ελεγκτής πρέπει να φροντίσει να τηρούνται οι χρονικές προθεσμίες της εξεταζόμενης εφαρμογής ακόμη και αν υπάρχουν απρόβλεπτες, δυναμικές εργασίες στο σύστημα. Ο ελεγκτής PID δουλεύει πάνω από μια εφαρμογή ανίχνευσης φάσματος που αναπτύχθηκε από την εταιρεία Thales [38], μία ρεαλιστική εφαρμογή για ενσωματωμένα συστήματα από τον τομέα των επικοινωνιών της οποίας οι προθεσμίες είναι κρίσιμες για τη σωστή λειτουργία του συστήματος. Συγκεκριμένα, η εφαρμογή εκτελεί χαρακτηρισμό σήματος των καναλιών GSM. Η ιδέα του ελεγκτή είναι να αυξήσει τη συχνότητα λειτουργίας της πλακέτας όποτε υπάρχει καθυστέρηση από τις προθεσμίες της εφαρμογής. Ως εκ τούτου, η ανάγκη για σωστή παρακολούθηση των χρονικών καθυστερήσεων που εισάγονται όταν συμβαίνει κάποιο σφάλμα και εκτελείται ένα rollback, είναι απαραίτητη για την προσπάθειά μας.

Για το σκοπό αυτό, εισάγουμε μια μεθοδολογία για να εντοπίσουμε σωστά πότε παρουσιάστηκε ένα σφάλμα, με ένα μηχανισμό επαναφοράς στο τελευταίο σωστό σημείο λειτουργίας. Πρώτον, η εν λόγω εφαρμογή τρέχει στην ονομαστική συχνότητα λειτουργίας της πλακέτας υπό συνθήκες χωρίς σφάλματα και συλλέγεται μια υπογραφή που περιλαμβάνει δεδομένα σχετικά με τα παραγόμενα αποτελέσματα και συγκεκριμένα χρονικά διαστήματα. Είναι στη φύση της εφαρμογής ότι σε ορισμένα σημεία της πορείας της, οι υπολογισμοί μπορούν να επαναληφθούν εάν χρειαστεί. Ειδικότερα, η εφαρμογή μπορεί να επαναλάβει τον αλγόριθμό ανίχνευσης για διαφορετικά κανάλια GSM. Αυτό επιτρέπει παρεμβάσεις επαναφοράς ή rollback interventions. Τα δεδομένα της εκτέλεσης χωρίς σφάλματα συλλέγονται σε αρχείο που περιέχει το χρόνο και την υπογραφή της εφαρμογής στο συγκεκριμένο χρόνο που ονομάζονται ”χρυσά ίχνη” ή ”golden traces”. Στη συνέχεια, υπό κανονική λειτουργία του συστήματος - περιμένοντας δηλαδή σφάλματα -, οι υπογραφές της εφαρμογής θα συλλεγχθούν και πάλι και θα συγκρίθούν με το χρυσό ίχνος στα συγκεκριμένα χρονικά διαστήματα. Τα δεδομένα αποθηκεύονται σε ένα αρχείο που ονομάζεται ”Signatures.txt”. Το μέγεθος του αρχείου είναι μικρότερο από 2 kilobytes.

Τώρα που τα σωστά αποτελέσματα της εφαρμογής είναι γνωστά στον μηχανισμό RAS, μια κανονική εκτέλεση κάτω από, ελεγχόμενες απο PID, αλλαγές DVFS μπορεί να ξεκινήσει. Σφάλμα υπάρχει όταν τα αποτελέσματα που υπολογίζονται στα συγκεκριμένα χρονικά διαστήματα δεν παράγουν την ίδια υπογραφή όπως στην εκτέλεση χωρίς σφάλματα. Θεωρούμε ότι ο μόνος τρόπος που μπορεί να γίνει αυτό είναι ένα σφάλμα υλικού. Για το λόγο αυτό, για να σταματήσουμε την λανθασμένη λειτουργία του συστήματος και να διασφαλίσουμε τη λειτουργική ορθότητα, κάνουμε μια επαναφορά / rollback στην τελευταία σωστά υπολογισμένη υπογραφή και επανεκκινούμε την εφαρμογή από το συγκεκριμένο σημείο ελέγχου. Η διαδικασία αυτή, ωστόσο, εισάγει καθυστέρηση στο σύστημα οπότε πλέον υπάρχει κίνδυνος να χάσουμε την τελική προθεσμία της εφαρμογής.

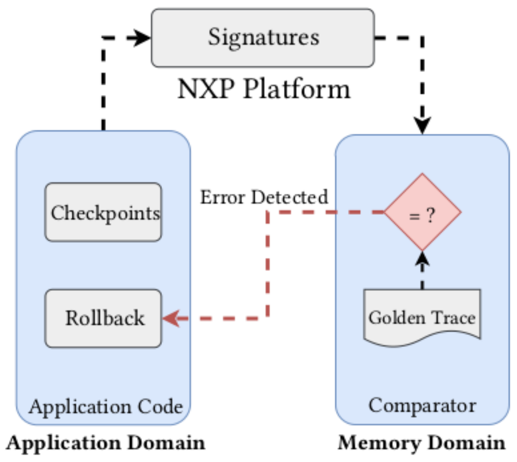

Figure 0.2: Η περιγραφή του μηχανισμού RAS. Το σύστημα ελέγχει τις σωστές υπογραφές με αυτές τα ”χρυσά ίχνη” αλλιώς κάνει επαναφορά / rollback

Σε αυτό το σημείο παρουσιάζουμε τη μεταβλητή που χρησιμοποιούμε για να παρουσιάσουμε χρονικές αποκλίσεις.

Definition. *Slack* ($s[n]$) είναι η χρονική διαφορά που υπάρχει όταν το σύστημα παράγει την ίδια υπογραφή καθώς δουλεύει στην ονομαστική συχνότητα f_{nom} και την τρέχουσα συχνότητα $f[n]$

$$s[n] = t_{ref} - t[n] \tag{0.1}$$

όπου το t_{ref} αναφέρεται στο χρόνο αναφοράς και το $t[n]$ στο χρόνο λειτουργίας στην τρέχουσα συχνότητα. Επομένως, μια χρονική καθυστέρηση παρουσιάζεται με αρνητικό slack και σημαίνει ότι η πλακέτα πρέπει να αυξήσει τη συχνότητα λειτουργίας της για να προλάβει την προθεσμία ενώ θετικό slack σημαίνει ότι είμαστε μπροστά σε χρόνο και μπορούμε να μειώσουμε τη συχνότητα λειτουργίας για να εξοικονομήσουμε ενέργεια. Είναι προφανές ότι η βέλτιστη χρήση του ελεγκτή μας θα κρατήσει το slack στο 0.

Είναι προφανές από τώρα ότι η ανάγκη εναλλαγής μεταξύ πολλαπλών τιμών συχνότητας και τάσης είναι πολύ σημαντική για την εργασία μας. Ωστόσο, η εργοστασιακή διαθεσιμότητα DVFS για την πλακέτα μας, την iMX6Q από την NXP, είναι περιορισμένη. Προσφέρει μόνο την ονομαστική συχνότητα 792 MHz, μια αργή συχνότητα 396 MHz και μια γρήγορη συχνότητα στα 996 MHz. Αυτή η διαμόρφωση δεν βοηθάει τα πειράματά μας. Επιπλέον, η αρχιτεκτονική της πλακέτας επιτρέπει μέγιστο αριθμό 50 σημείων συχνότητας που κυμαίνονται από 396 MHz έως 996 MHz. Αυτά τα σημεία δεν είναι διαθέσιμα με τα προεπιλεγμένα προγράμματα της πλακέτας. Αυτός είναι ο λόγος που χρησιμοποιούμε ένα καινούργιο πρόγραμμα που αναπτύχθηκε από την Thales που επιτρέπει όσα περισσότερα σημεία θέλουμε (έως και 50) και χειρίζεται τις DVFS αλλαγές. Μια μελέτη για το πόσα σημεία θα πρέπει να επιλέξουμε για τη βέλτιστη απόδοση του PID μας μπορεί να βρεθεί αργότερα σε αυτό το Κεφάλαιο. Τα σημεία που επιλέγουμε φαίνονται στους Πίνακες 3.1 και 3.2.

Θα παρουσιάσουμε τώρα τον PID ελεγκτή που αποφασίζει τις αλλαγές DVFS σύμφωνα με την τιμή του slack. Το όνομα PID προέρχεται από τα αρχικά της φράσης ”Proportional-Integral-Derivative” που περιγράφει την ακόλουθη εξίσωση που χρησιμοποιεί ο PID:

$$m[n] = \underbrace{k_p s[n]}_{\text{proportional}} + \underbrace{(s[n] - s[n-1])k_d}_{\text{differential}} + \underbrace{(m[n-1] + s[n]k_i)}_{\text{integral}} \tag{0.2}$$

όπου το s[n] ορίζεται στην Εξίσωση 0.2. Το proportional τμήμα, που σχετίζεται άμεσα με το κέρδος k_p παράγει ένα αποτέλεσμα που είναι ανάλογο με τη μέτρηση του slack. Το integral μέρος, που περιγράφεται με το κέρδος k_i, περιέχει προηγούμενες τιμές της τιμής που υπολογίζεται από τον PID, επομένως αποτελεί τη ”μνήμη” του ελεγκτή. Το integral τμήμα, με βάση το κέρδος k_d, λαμβάνει υπόψη την κλίση του σφάλματος και ενεργεί ως το προβλεπτικό μέρος της εξίσωσης. Η υπολογιζόμενη τιμή, που ορίζεται εδώ ως m[n], είναι ο πολλαπλασιαστής συχνότητας που θα κβαντιστεί και θα στρογγυλοποιηθεί προς την πλησιέστερη τιμή που αντιπροσωπεύει ένα σημείο DVFS στο σύστημά μας.

Ο ελεγκτής PID είναι το βιομηχανικό πρότυπο στον κλάδο της μηχανικής συστημάτων και στους χώρους εργασίας όπου οι ελεγκτές πρέπει να εξασφαλίζουν την αξιοπιστία και την ασφάλεια του χρήστη, όπως τα εργοστάσια και τα διυλιστήρια. Προσθέτοντας σε αυτό, αναγνωρίζεται η εύχρηστη φύση της εξίσωσης. Μπορεί να κατανοηθεί και να αναπτυχθεί από μηχανικούς εύκολα και είναι μέχρι σήμερα ένας πολύ αξιόπιστος ελεγκτής, παρά τους πολλούς πιο εξελιγμένους ελεγκτές που σχεδιάζονται και κατασκευάζονται. Στη συνέχεια, ακολουθούν μερικά βασικά χαρακτηριστικά της λειτουργίας ενός PID ελεγκτή, τροποποιημένα ώστε να ταιριάζουν στους σκοπούς των πειραμάτων μας.

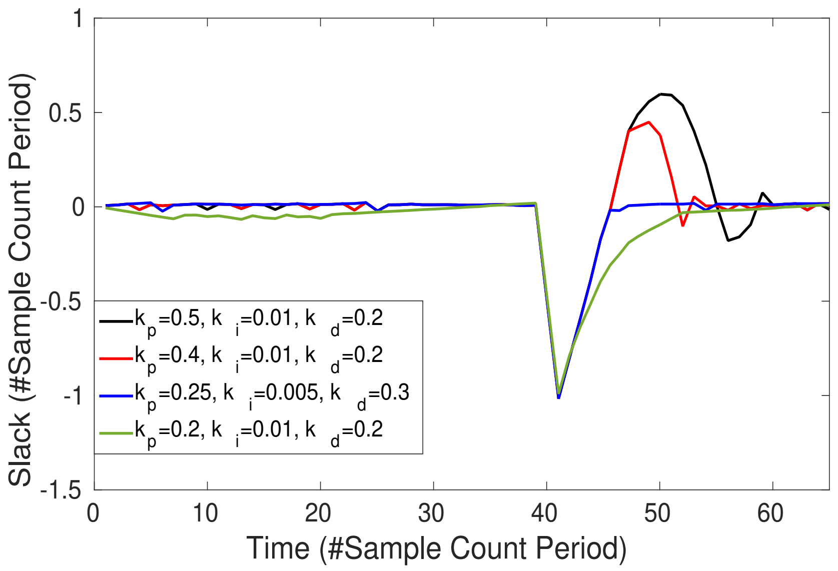

Figure 0.3: Ρυθμίσεις των κερδών του PID για να ελαχιστοποιήθει η υπέρβαση και ο χρόνος διευθέτησης

Το ποσοστό του ποσού του σφάλματος που ξεπερνά την τιμή στόχου όταν το PID προσπαθεί να συγκλίνει στην τιμή στόχου ονομάζεται υπέρβαση ή *overshoot*. Στην περίπτωσή μας, η υπέρβαση περιγράφει πόσο γρηγορότερα λειτουργούμε σε σύγκριση με το τρέξιμο χωρίς σφάλματα, επομένως χάνουμε ενέργεια. Είναι ζωτικής σημασίας να διατηρηθεί η υπέρβαση όσο το δυνατόν μικρότερη και κατά προτίμηση να εξαλειφθεί.

Ο χρόνος που απαιτείται για το slack να συγκλίνει στην τιμή στόχου, εντός του ορίου του 10% από τη στιγμή που το σφάλμα εμφανίστηκε για πρώτη φορά ονομάζεται χρόνος διευθέτησης ή settling time. Η διατήρηση ενός μικρού χρόνου διευθέτησης θα σημαίνει ότι το σύστημα αντιδρά γρήγορα και μετριάζει τα λάθη γρήγορα, επομένως είναι καλή πρακτική για το σύστημά μας. Ωστόσο, έχει γενικά διαπιστωθεί ότι η ελαχιστοποίηση της υπέρβασης και του χρόνου διευθέτησης ταυτόχρονα μπορεί να είναι αρκετά δύσκολη και είναι συχνά θέμα επιλογών. Η μείωση και των δύο μπορεί να γίνει με την τελειοποίηση των κερδών του PID. Η εργασία μας παρουσιάζεται στο Σχήμα 0.3, όπου μπορούμε να δούμε τους διάφορους συνδυασμούς των κερδών και να επιλέξουμε το ένα με τη μικρότερη υπέρβαση και χρόνο διευθέτησης.

Πειραματικά αποτελέσματα

Το πρώτο σετ πειραμάτων το πώς το DVFS μετριάζει τις επιδράσεις της επαναφοράς / rollback που προκαλείται από τον μηχανισμό RAS και εισάγει καθυστερήσεις. Λαμβάνοντας υπόψη τον μηχανισμό που παρουσιάστηκε προηγουμένως, όταν εισάγεται σφάλμα, παρατηρείται μια καθυστέρηση μίας Περιόδου Παρακολούθησης Δείγματος στο slack και ο PID αυξάνει τη συχνότητα λειτουργίας του συστήματος προκειμένου να επιταχυνθεί η εφαρμογή και να απαλειφεί η καθυστέρηση.

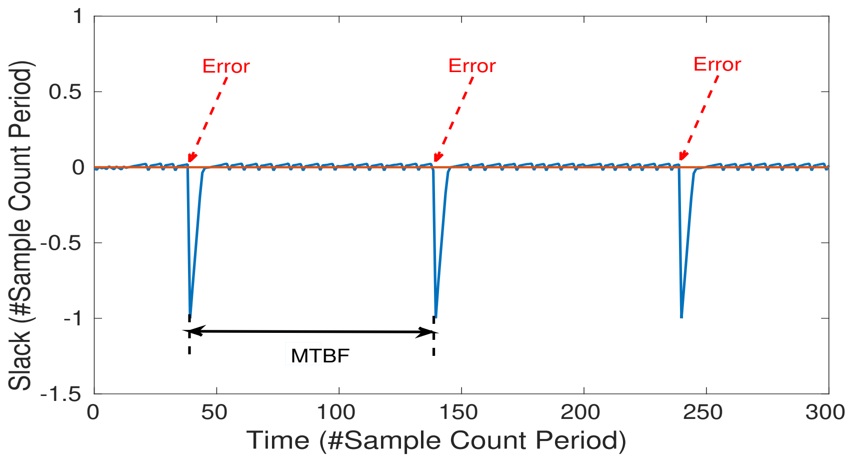

Figure 0.4: Slack συναρτήσει του χρόνου για 3 επαναφορές / rollback

Στο Σχήμα 0.4 έχουμε δείξει τα σημεία στα οποία εισήγαμε τα σφάλματα και προκαλέσαμε την καθυστέρηση. Παρατηρούμε μέσω του Σχήματος 0.5 ότι ο PID μεταβαίνει αμέσως στη μέγιστη συχνότητα για όσο χρονικό διάστημα είναι απαραίτητο για το slack να επιστρέψει στο 0 όσο το δυνατόν γρηγορότερα. Πρέπει να σημειώσουμε ότι είναι επιλογή του χρήστη να επιλέξει τα κέρδη του PID για να πετύχει ελάχιστο χρόνο διευθέτησης και μέγιστη απόδοση και να ανέχεται πιθανώς μεγαλύτερη κατανάλωση ενέργειας. Σε άλλη ρύθμιση των κερδών, θα μπορούσαμε να ανεχθούμε μεγαλύτερους χρόνους διευθέτησης και, πιθανή επιδείνωση της κατανάλωσης ενέργειας κατά τη διάρκεια της μεταβατικής περιόδου του ελεγκτή. Εν πάση περιπτώσει, πρόκειται για ανταλλαγές κερδών και οι μεγάλοι χρόνοι διευθέτησης δεν εγγυώνται απαραίτητα μικρή ενέργεια. Στο Σχήμα 0.6 παρουσιάζουμε τα ακριβή ζεύγη συχνότητας και τάσης που ο PID επέλεξε να χρησιμοποιήσει. Στο Σχήμα 0.7 παρουσιάζουμε τη συνολική ενέργεια που καταναλώνεται από το σύστημα για το χρονικό πλαίσιο και το

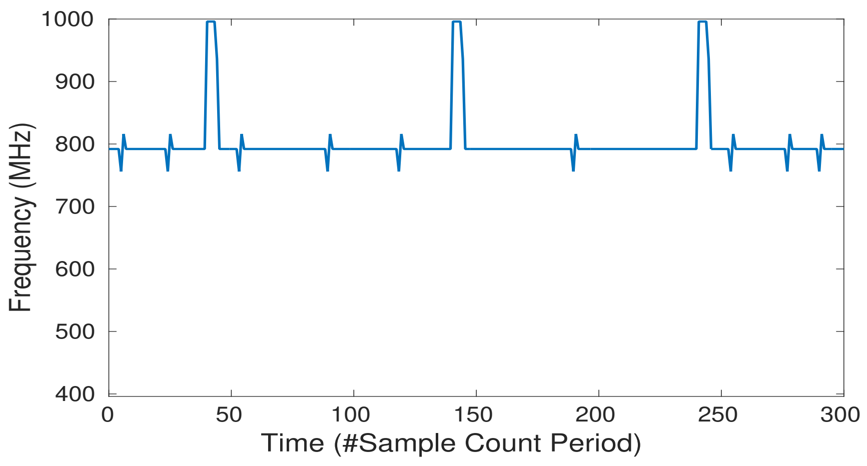

Figure 0.5: Συχνότητα λειτουργίας συναρτήσει του χρόνου για 3 επαναφορές / rollback

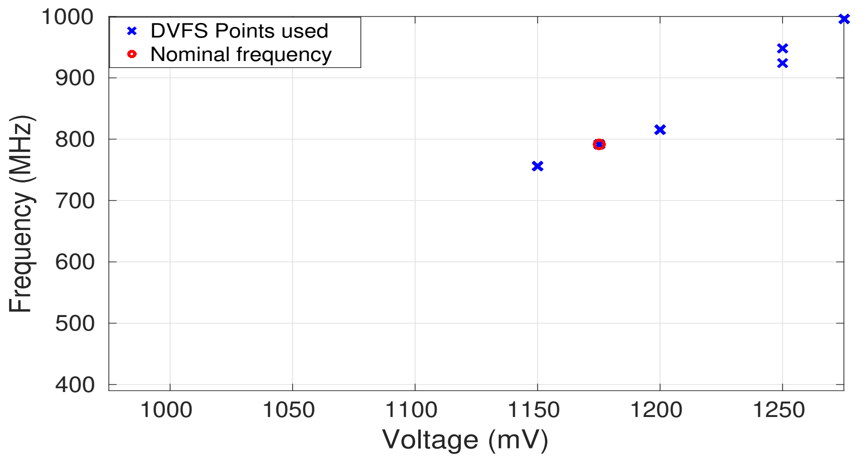

Figure 0.6: Σημεία DVFS που χρησιμοποιεί ο PID

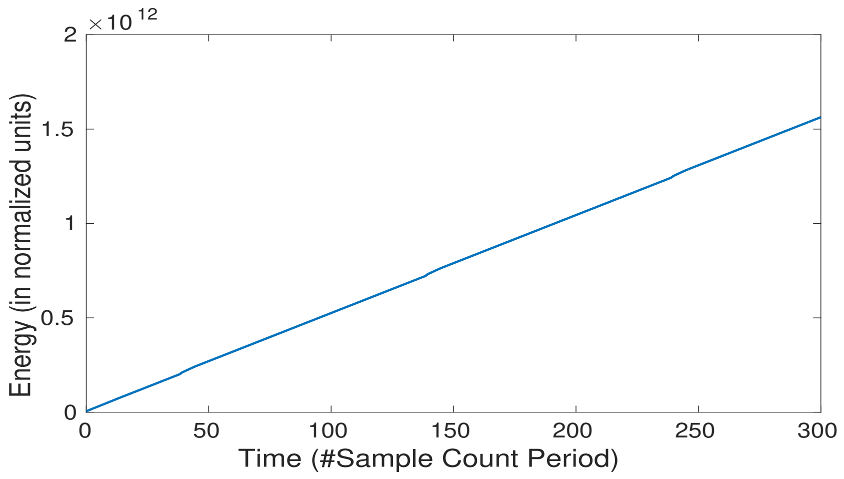

Figure 0.7: Συνολική ενέργεια που καταναλώνεται για 3 επαναφορές / rollback

ποσοστό σφάλματος που απεικονίζεται στο Σχήμα 0.4.

Το επόμενο σετ πειραμάτων που διεξήγαμε στοχεύει να διερευνήσει πώς το σύστημα ανταποκρίνεται σε επιπλέον φόρτο εργασίας που εισάγει μεταβλητότητα απόδοσης, υπό την έννοια ότι οι αυστηροί χρονικοί περιορισμοί κινδυνεύουν να μην ικανοποιηθούν από το σύστημα. Επομένως, ο ελεγκτής PID πρέπει να ρυθμίσει τη συχνότητα λειτουργίας σε τέτοια επίπεδα ώστε να ικανοποιούνται και οι δύο φόρτοι εργασίας και να πληροούνται οι προθεσμίες διατηρώντας το slack κοντά στο 0. Μια εύκολη λύση θα ήταν να ρυθμίσουμε τη συχνότητα στην υψηλότερη τιμή και να κάνουμε το slack θετικό αλλά αυτή η λύση δαπανά περιττή ενέργεια. Αντιθέτως, θέλουμε να μην χάνουμε υπερβολική ενέργεια οπότε ο PID πρέπει να βρει την ελάχιστη συχνότητα λειτουργίας για να ικανοποιήσει και τους δύο φόρτους εργασίας. Σε αυτό το είδος πειραμάτων υποθέτουμε συνθήκες χωρίς σφάλματα, ωστόσο, σε περίπτωση που προκύψουν σφάλματα, η λειτουργία του PID δεν επηρεάζεται και τελικά το σύστημα θα είναι αξιόπιστο.

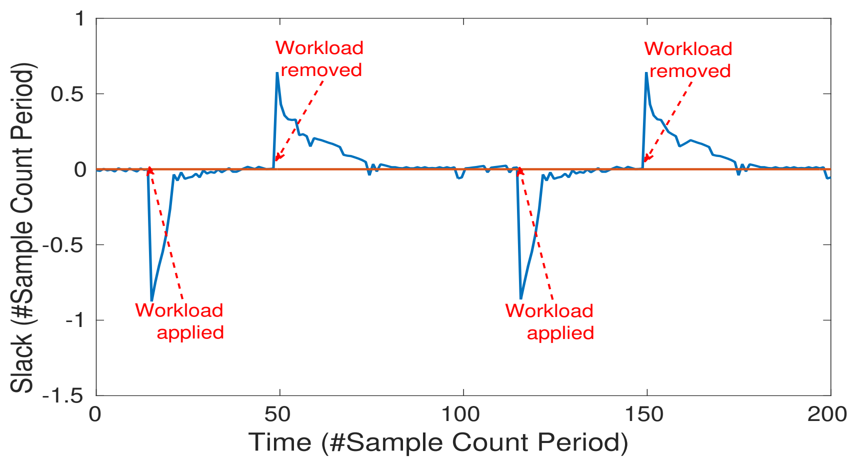

Figure 0.8: Slack συναρτήσει του χρόνου για επιπλέον φόρτο εργασίας

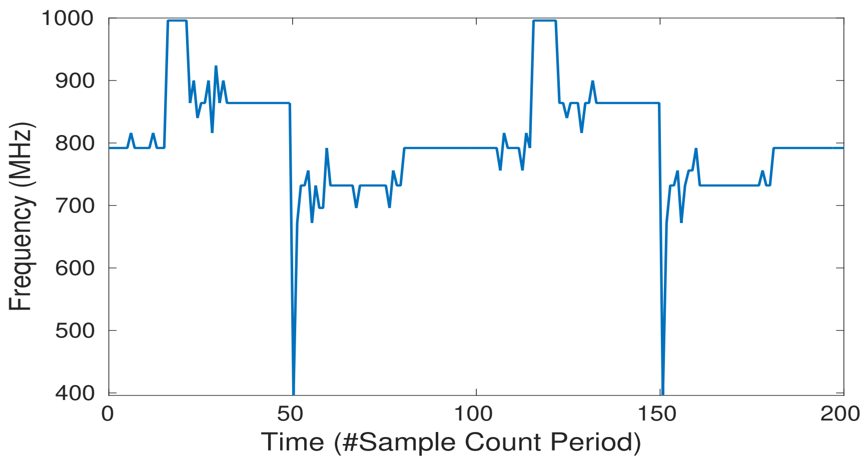

Figure 0.9: Συχνότητα συναρτήσει του χρόνου για επιπλέον φόρτο εργασίας

Όπως φαίνεται στις Εικόνες 0.8 και 0.9, όταν εφαρμόζεται το επιπλέον φορτίο εργασίας δημιουργείται αρνητικό slack που αναγκάζει τον PID να αυξήσει τη συχνότητα σε υψηλότερα επίπεδα. Τελικά, το slack καταλήγει κοντά στο μηδέν σε συχνότητα 894 MHz, καθώς αυτή είναι η συχνότητα που βρήκε ο PID ώστε να μπορεί να χειριστεί και τους δύο φόρτους εργασίας. Όταν ο επιπλέον φόρτος

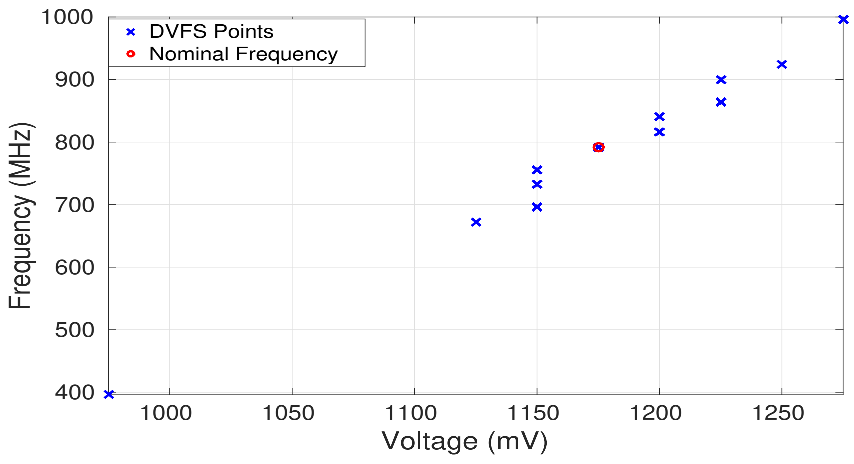

Figure 0.10: Σημεία DVFS που χρησιμοποιεί ο PID για επιπλέον φόρτο εργασίας

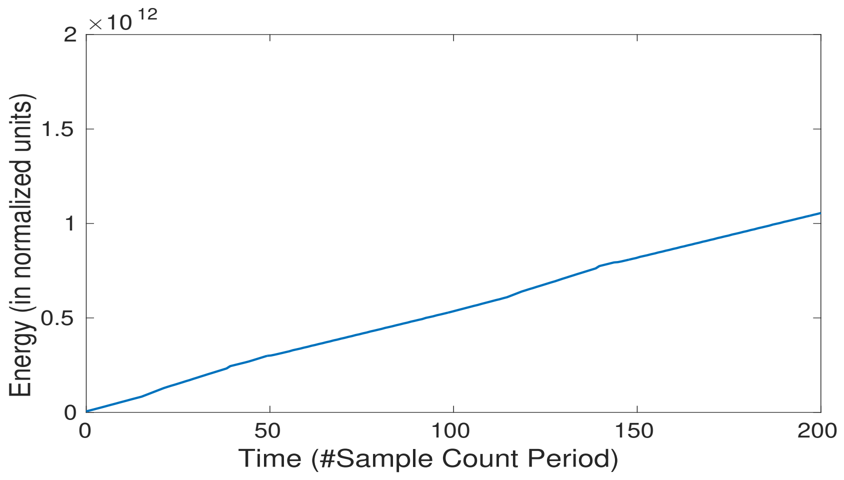

Figure 0.11: Συνολική ενέργεια που καταναλώνεται για επιπλέον φόρτο εργασίας

εργασίας απομακρύνεται, το slack αμέσως αυξάνεται σε θετικές ιμές, επειδή το σύστημα λειτουργεί σε αχρείαστα υψηλά επίπεδα συχνότητας σπαταλώντας ενέργεια. Με τη σειρά του, ο PID θα επιβραδύνει αργά μέχρι την ονομαστική συχνότητα επιστρέφοντας ταυτόχρονα το slack στο 0, εξοικονομώντας έτσι ενέργεια. Οι αποφάσεις του PID παρουσιάζονται στο Σχήμα 0.10 και η συνολική ενέργεια που καταναλώνεται για τη διάρκεια 2 διαδοχικών πειραμάτων παρουσιάζεται στο Σχήμα 0.11.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Integrated Circuit manufacturing industry has complied to technology scaling in accordance with Moore's Law for the last 40 years, which has improved the performance of embedded systems drastically. As smaller transistors and their connecting wires are packed in increasingly smaller areas of the silicon, the overall performance of the chip rises due to more operations conducted per minute and less energy needed to power the chip. At the same time, the manufacturing cost of producing thousands of wafers each containing billions of transistors in a single day's work has been getting smaller and smaller, leading to a booming chip industry that has shaped the 21st century as we know it today.

This miniaturization, however, is now facing serious challenges that need to be overcome. The demand for faster and reliable chips, able to run increasingly demanding consumer applications has hit a "brick wall" in the problems of process variation, temperature dissipation and energy consumption[1]. The term process variation describes the phenomenon in which two transistors with the same design and fabricated a few nanometers apart may have different electrical and structural characteristics due to a series of unavoidable causes[2][3]. Process variation has always been present in the manufacturing process[2] but recently, since the dimensions of the transistors reached the 90-nanometer node, it is not only causing loss of yield for the industry (as a number of transistors do not meet the specifications) but also causes a variability in the performance of chips[3].

Furthermore, the performance of modern embedded systems is impacted by the dynamic nature of their workloads. Nowadays, applications are highly dynamic in the resources they demand and timing constraints they abide by. This dynamism is the result of both hardware-related faults[2] and software-related demands in the form of computation-heavy applications and input data of varying scale and intensity. It is, therefore, becoming more and more apparent that system performance is affected by this dynamism and it needs to be taken into account when designing energy and timing budgets.

In addition, while the industry is having a hard time to further scale the transistors down, the threshold voltage of these devices has stopped decreasing, as Dennard's Scaling model originally suggested, resulting in a drastic rise in consumed energy and system temperature which threatens the optimal performance of the chip. While on desktops and servers the chips are cooled down by spending even more energy on fans and other active cooling techniques, that option is not available with hand-held devices such as smartphones, tablets and other small Systems-on-a-Chip (SoC) from the embedded systems domain. The industry, in fact, has been forced to tolerate the *Dark Silicon* phenomenon; the necessity which dictates that only a portion of the transistors of the chip will be powered on at the same time in order to comply with strict power and temperature budgets[4].

While addressing the aforementioned threats, the embedded systems industry also needs to be able to ensure the dependability of the system. An embedded system is usually expected to fulfill real-time tasks and accomplish its goals while avoiding the dangers presented above. A gross generalization of this demand is that embedded systems are expected to meet deadlines set a priori while securing binary correctness of the system. Specifically, to ensure correct system operation, the industry has developed numerous Reliability, Availability and Serviceability (RAS) techniques[5].

Towards this direction, in this thesis we present a PID controller that aims to counter the effects of dynamic workloads and RAS interventions by using Dynamic Voltage and Frequency Scaling (DVFS) techniques. Moreover, our scheme is updated to manage chip temperature within normal levels. The application is drawn from the HARPA project of the European Union and the complete feedback-based controller is deployed on pure hardware, namely the NXP iMX6Q-SABRE-SD board[6].

In Chapter 2, we discuss theoretical aspects of process variability and present several RAS schemes already developed by the industry. We also introduce the technique of Dynamic Voltage and Frequency Scaling. In Chapter 3, we present our application and explain critical concepts of our technique. In Chapter 4, the experimental setup is introduced and the results for process and workload variability as well as thermal management are presented. In the Conclusion, we summarize key findings of our work and propose interesting aspects for future work.

Chapter 2

Related Work

2.1 From hardware faults to failure manifestation

2.1.1 Sources of process variation

A major source of dynamic system operation, causing hardware defects is process variation. It can be split into two major categories: global or systematic variation and random variation also known as process variability[2]. While both categories can potentially cause transient errors and effect the same electrical characteristics of the transistor, namely the leakage current, operating frequency and threshold voltage V_{th}, they manifest through different sources and therefore need different methods to be countered. On one hand global variation is caused by small changes in the environment of the manufacturing process and affects all transistors on a wafer in the same way. The main sources of systematic variation are optical proximity correction (OPC)[7], line edge roughness (LER) and line width roughness(LWR)[8][9] and variations in the gate dielectric[10][2]. For example a slight deviation in the optics of the light of a lithography tool can cause an entire wafer to have transistors of smaller or bigger channel width than originally designed. This type of variation has always been present in the manufacturing process and the industry has successfully developed numerous ways to counter its effects[11].

On the other hand, process variability has presented itself only since the industry reached the 90 nm node. While the downscaling of transistors continued, the effects of process variability have proven difficult to counter as the hardware defects that cause them are directly associated with the small dimensions of the transistors. It should be noted that process variability can be further split into two categories: time-zero variability, i.e. variability that is caused by internal sources at the manufacturing stage of the silicon device, and time-dependent variability that is caused by external sources such as operating temperature, humidity and workload stress of the device.

Among others, one major cause for time-zero process variability is Random Dopant Fluctuations (RDF)[12]. For years the industry has implanted the gates of MOS transistors with dopant atoms in order to efficiently control their threshold voltage V_{th}. And while at older technological nodes the dimensions of the gate were big enough for the concentration of the dopant to be negligible, we are now coming across a phenomenon where the number of atoms doped and the dimensions of the gate are small enough that the law of large numbers no longer applies. Therefore the discreteness of the atoms can cause two transistors, one next to the other, to have different threshold voltages and therefore operate differently or consume more energy.

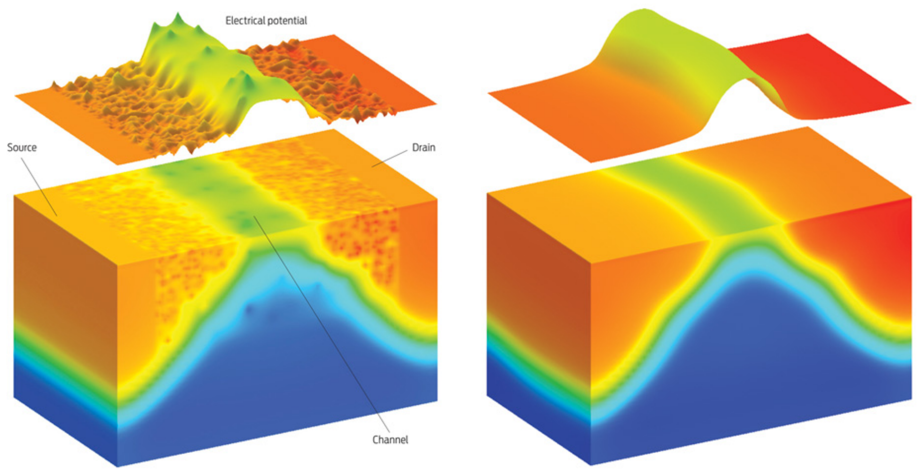

Figure 2.1: RDF and roughness of the MOS transistor can cause variations in its threshold voltage[11]

Moreover, the continuous operation of transistors, especially in real-time systems, is provoking aging mechanisms that affect the reliability of the chip and lead to performance degradation. These effects are only getting more dangerous as the operating temperature of the system increases. Some notable phenomena that cause time-dependent variability and are tightly associated with elevated temperatures are Hot Carrier Injection (HCI)[13][14], Time-Dependent Dielectric Breakdown (TDDB) and, most importantly, Bias Temperature Instability(BTI)[15][16][17][18]. While these problems affect mainly the gate oxide of the transistor, and ultimately its V_{th}, there are also phenomena that provoke time-dependent variability in the interconnecting wires of the transistors such as Electromigration(EM)[19] and Self Heating[13][20].

As BTI has proven itself the most important factor of time-dependent variability the research community has developed two main theories to explain it. The first and oldest theory is the reaction-diffusion model (RD)[21][22]. This model suggests that in pFETs BTI can be explained by the breaking of Si-H bonds at the Si/oxide interface. While the hydrogen atoms are diffused at the gate-stack, minority carriers, also known as ”holes”, move to take their place in the grid thus creating charge traps that shift the V_{th} of the transistor[15]. This approach however has not captured the problem completely and nowadays is substituted with a more innovative, defect-centric theory. That is the atomistic model that introduces a probabilistic component in the effort to explain the effects of the formed defects. It contradicts the RD model as it suggests that traps created by defects can also be discharged and not necessarily charged and only the charged ones shift the V_{th}[21].

Last to be mentioned, but certainly not least in importance, mechanisms leading to performance variability are cosmic and terrestrial radiation particles, such as alpha particles and high energy cosmic rays[2][20]. The dimensions of transistors are now small enough that random radiation particles bumping on them can raise or lower their voltage for a significant enough amount of time for the error created to propagate to other transistors and ultimately endanger the correct operation of at least a subsystem of the chip. This phenomenon is usually apparent in memory chips and is referred to as ”bit flip”, because the radiation particle can flip a memory bit from 0 to 1 and vice versa and cause the system to read wrong memory data. Normally the bit will flip back to its correct state and the error will be reversed on its own without causing further damage. However if a radiation event causes charge disturbance on a critical system, product malfunction can occur[23].

2.1.2 Modeling of process variation - RAS events

In order for the industry to successfully mitigate the effects of the above-mentioned hardware faults, multiple models have been studied and developed by the reliability community to quantify the effects of these faults on key electrical characteristics of transistors and characterize the problems created by these effects.

Firstly, the variation in the threshold voltage of transistors in a single wafer, that occurs because of time-zero variability, is modeled after a Gaussian distribution around a mean value V_{th} with a variance σ. Staying in agreement with this model, systematic variation can be modeled with a Gaussian distribution with a shifted value of V_{th} and same variance σ[2]. The mean V_{th} represents the optimal value for the transistor's threshold variation. The variance σ can be calculated given a sufficient amount of samples and is equal to:

$$\sigma_{V_{th}} = \frac{A_{VT}}{\sqrt{2WL}} \tag{2.1}$$

where A_{VT} is Pelgrom's mismatch parameter[2][40], W is the nominal width of the transistor's channel and L is the nominal length of the transistor's channel. As we can see, by scaling down (decreasing the dimensions of) the transistors, the variance will keep increasing, resulting in some transistors having a threshold voltage either too high for our energy budget or too low to ensure correct performance. Therefore, the industry has compromised by producing more transistors than it needs and introducing another metric called yield, which describes the percentage of transistors with threshold voltages within accepted limits to the total number of transistors manufactured according to specific electrical characteristics.

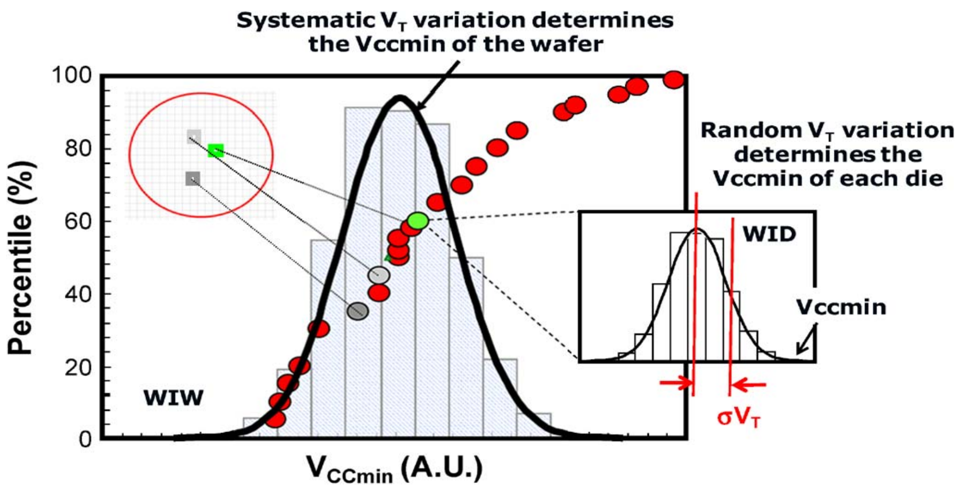

Figure 2.2: MOS V_{th} is affected by both global and random variation modeled with a Gaussian distribution[2]

Another way to model hardware faults is by taking into account their respective lifetimes, i.e. for how long do they appear before corrective action is taken. Permanent faults, such as oxide wear-out, remain in the chip indefinitely and can sometimes be irreversible while transient faults caused by interaction with radiation particles disappear quite soon. This raises the subject of fault detection and error manifestation.

Faults manifest themselves through hardware *errors*. Much like faults, errors can be permanent (commonly referred to as hard errors) and transient or *soft errors*, whose effects are temporary. However not all faults lead to errors. Faults that occur on parts of a chip that are not mission critical, like a branch predictor, can go unnoticed by the end-user, as, in our example, the system's architecture is designed to recover from a failed prediction. Moreover, errors are further classified based on the type of system failure they cause. The two most concerning types of errors are errors that cause Silent Data Corruption (SDC) and Detected Unrecoverable Errors (DUE). For example, assuming an embedded platform running a Fast Fourier Transform algorithm, an SDC error occurs when the algorithm produces wrong results because of corrupted memory data. Contrariwise, a DU Error occurs when the platform shuts down unexpectedly, potentially losing all computed data in the process. Both categories are frequently described as system *failures* by the reliability community[20].

A system failure can be defined as "a malfunction that causes the system to not meet its power, correctness or performance guarantees"[20]. It has been established[39] that specific hardware faults and their errors are tied with specific forms of failure. The industry has developed numerous metrics to describe and control system failure. The most common are:

1. **Failure In Time rate (FIT_{rate})**: Represents how often a device fails in its lifetime normalized to a billion device hours[41].
2. **Mean Time To Failure (MTTF)**: Represents the mean time of operation before a failure[20].
3. **Mean Time Between Failures (MTBF)**: Represents the mean time of operation between two consecutive system failures[20][42].

A characteristic curve constantly used by the industry is the "bathtub" curve, which describes how a system's FIT_{rate} correlates to a system's operating time. Initially, due to the presence of hard errors caused in the manufacturing process, a system will experience increased failure rates but later in its life, commonly during consumer usage, the errors experienced should be the result of only random variation sources and any implemented RAS mechanism should mitigate their effects. As years of correct operation pass, it is inevitable that aging mechanisms, such as BTI and TDDB, will be exacerbated by the constant usage and failure rates will once again climb to a point where the system will be unusable.

2.2 RAS Mechanisms

In order to efficiently counter the effects faulty hardware imposes on embedded systems, the industry has developed a variety of Reliability, Availability and Serviceability (RAS) techniques. A common characteristic between all of them is that they are built around the necessity to tolerate trade-offs. In simple terms, in order to implement a RAS mechanism necessary for the reliable operation of the chip

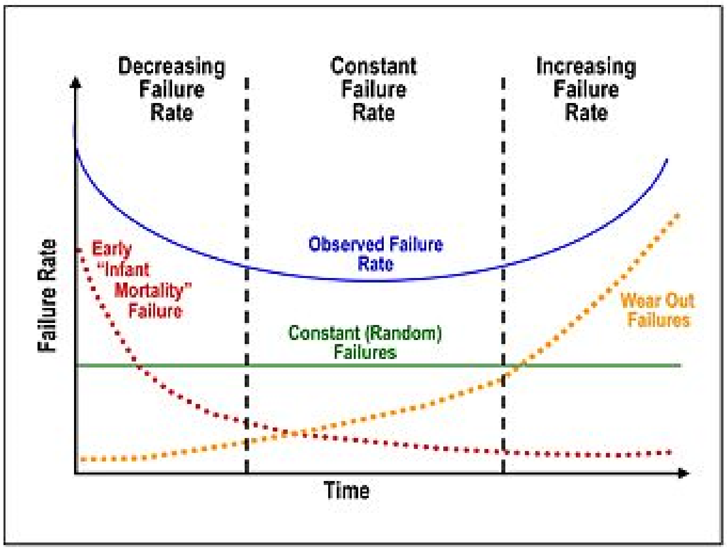

Figure 2.3: FIT_{rate} as is composed by infant mortality, wear-out and random variation failure rates in relation to the system's operating time[39].

for as long as possible, the manufacturer trades either extra silicon area (redundancy mechanisms), or power usage, or timing constraints. The most commonly used RAS schemes can be classified in those targeting *functional reliability* i.e. correct operation on the binary level and those targeting *parametric reliability* i.e. the mitigation of fluctuations of specific performance parameters from their expected values.

Concerning functional reliability, a categorization of the most commonly used techniques was recently done[43]. Firstly the technique of Error Correcting Codes (ECC)[20][44] has been largely employed to recover from soft errors distorting memory data. A second way to target functional reliability is by utilizing modular redundancy techniques[45]. These techniques propose the introduction of numerous similar (in the sense of their purpose on the system) modules, or systems of circuits, so that in the case one is faulty, the others will "take its place" and correct operation will be ensured. Moreover, in the case of defective memory blocks, permanently deactivating them is a legitimate method to ensure fault tolerance[46]. In the logic level, shadow latches[47] and instruction-level rollback[48][49] can help mitigate timing violations and ensure correct operation in expense of small operating time respectively. The idea of instruction-level rollback is utilized in the experimental part of this thesis.

In other cases, however, where power and timing constraints are more important than correct operation on the binary level, engineers have developed methods to tackle this parametric reliability while also ensuring correct operation on the system level. While things are easier when the applied workload's timing constraints and power necessities are known beforehand, modern embedded systems usually deal with highly dynamic workloads where establishing a scheme to deal with the varying time constraints and power usage can be extremely difficult. Nevertheless, there is an abundance of proposed methods that aim to ensure system dependability. In the scope of this thesis, we discuss techniques that aim to efficiently manage timing dependability while taking into account potential RAS interventions.

Some notable examples of methods targeting timing dependability for dynamic workloads are real-time task scheduling algorithms based on systematic feedback[50][51], PID-based algorithms[52] and a Worst-Case Ready Queue (WCRQ) scheduling algorithm utilizing DVFS[53]. While managing dynamic workloads is a well documented area of study, dealing with RAS interventions at the same time is an open field of research. In spite of utilizing voltage and frequency worst-case operating scenarios to mitigate variability becoming an industry standard[54], modern approaches suggest that variating margins and applying smarter algorithms is necessary to increase performance and minimize power consumption[55][56]. With that in mind, our approach in this thesis is based on previous work[57] that implements a PID-controlled DVFS scheme managing timing delays invoked by rollback-based RAS mechanisms. It is augmented by taking into account the applied workload and ensuring system's dependability.

2.3 Dynamism in embedded systems

It is commonly accepted that state-of-the-art electronic systems are highly dynamic at both the hardware and software level. At the hardware level, process variability introduces variation in the electrical properties of transistors, such as threshold voltage V_{th}, leakage power and channel width, through the mechanisms described above. In turn, temporal characteristics, such as delays and timing responses in the Register Transfer Level, and spatial characteristics of the device, such as availability of memory blocks and interconnect wires, are highly variating from designed values. While industry-standard RAS mechanisms exploit redundancy and worst-case-scenario techniques to minimize these effects[20][45], variability in hardware resources can still propagate through the abstraction layers and impact the end user. Therefore, we are forced to recognize a dynamism in the hardware of the system.

To add to this dynamism, the software of the system also acts dynamically[58]. With either input data fluctuating in time and leading to more computations needed, or user-activated processes causing system resources to become temporarily unavailable, system performance is hindered. In the modern era of connectivity, network resources also threaten the timing constraints of systems as third-party devices impact the performance of systems[59]. A prime example of dynamic system usage is cross-device computing, where hardware and network resources are allocated depending on the relevant workload, and timing response of the entire system may be halted should a single device face a delay.

In conclusion, embedded systems may no longer consider their workloads as static with predictable responses executed in controlled times. On the contrary, it is important they take into account the dynamism presented both from their hardware though RAS mechanisms and their software through user-associated inputs, highly varying applications sharing resources simultaneously and network connections being impacted by third devices. Within this context, our scheme will run alongside an industry-relevant application that simulates dynamic workload and attempts to harass the timing constraints of the system. The controller is asked to ensure dependability of the system while consuming the least possible energy.

2.4 Power and temperature management efforts

It is established by now that process variability is exacerbated by rising system temperatures. In addition, power consumption is also heavily dependent on temperature. In fact, static power, a component of overall power consumption that was negligible until 10 years ago, has an exponential relation to the system's temperature[24]. Therefore managing the system temperature and maintaining it in safe levels has been of vital importance to the embedded industry especially since Dennard's Scaling model stopped applying to transistors. And while consumer desktops and laptops use fans and heat-sink techniques to cool the processors, that option is not available for hand-held devices. Ultimately, we need to look elsewhere for efficient management of temperatures.

Among other heavily used techniques, Dynamic Voltage and Frequency Scaling (DVFS) is a widely used technique to manage the consumed power by switching the operating voltage and frequency of the system. Dynamic power has a quadratic relation to the applied voltage multiplied by the operating frequency, therefore it is immediately affected and heavily managed by DVFS. Moreover, static power, lost by sub-threshold and gate leakage currents of the transistor, has a linear relation to the applied voltage but an exponential relation to the system temperature which affects static power drastically[24].The fine granularity achieved with DVFS is vital for embedded systems that need to simultaneously save as much energy as possible and ensure dependable performance. An interesting work based on the benefits of DVFS is presented in [25]. AMD and Intel have also introduced power-aware methodologies with *"Cool'n'Quiet"* and *"SpeedStep"* technologies respectively[26][27]. At the same time, the Linux kernel CPUFreq governor offers good examples in DVFS approaches, especially in the "on-demand" and "conservative" modes of the governor[28]. In fact, we will compare our proposed PID-based scheme with the power consumption of a realization of the "conservative" governor.

In terms of efficient temperature management, research has focused on task scheduling schemes and workload redistribution on multiple cores in order to meet thermal design specifications[29][30][31]. DVFS plays an important role here as well, as setting system voltage to lower levels can have significant effects on system temperature[3][32]. A temperature prediction mechanism is mentioned in the literature[33] as well as a PI controller for thermal management[34] and a thermal management algorithm using Linux CPUFreq mechanisms on ARM[35]. We will be comparing our efforts with that specific algorithm in Chapter 4. Regarding related work drawn from the industry, Intel Pentium 4 and IBM PowerPC processors have already incorporated temperature management features[36][37].

Chapter 3

A PID controller for ensuring system dependability

We will now proceed to introduce our PID-controlled DVFS scheme and critical aspects on:

a) how we simulate RAS interventions,
b) how we instantiate a variety of voltage and frequency pairs that resemble DVFS states of our system,
c) how exactly our PID controller is set up and
d) we will present the target board on which we run our simulations.

Furthermore, we will discuss and explain how the implementation on a real system differs from simulations and how our PID can handle these irregularities. Lastly, we will discuss an approach where a PID-based DVFS scheme manages the board's system temperature.

3.1 Presenting the HARPA application

Our work is based and expands on previous work[57] that first proposed a PID controller to manage system dependability. The work was introduced as part of the European Commission's FP7-612069-HARPA project (HARnessing Performance variability)[60].

During the project, the board was subjected to elevated temperatures in order to exacerbate the manifestation of Silent Data Corruption errors. When such errors were sensed, the RAS intervention would perform a "hot reboot" of the system, which was performed with a rollback to the latest point of the computation process where the results were proven true. All of the work is presented in [61].

In this thesis we will present an application of the PID controller that will mitigate performance variability of the target board when confronted with software-injected soft errors[1]. The PID controller is instantiated on top of a spectrum sensing application developed by Thales[38], a realistic embedded application from the communications domain whose deadline constraints are critical to the system. In fact, the application is streaming in nature and performs signal characterization of the GSM channels. The idea of the controller is to elevate the operating frequency of the board whenever there is a time delay from the application's deadlines. Therefore, the need to correctly monitor time delays introduced when an error occurs and a rollback is performed is essential for our effort.

[1] These software-injected errors mimic transient errors that could occur if for example our platform was operating in high temperatures or on an environment where particle strikes were possible.

3.1.1 RAS Events - Rollback

To that end, we introduce a methodology to correctly identify when an error has occurred and intervene with a rollback mechanism to the last correct operating point. Firstly, the application in question runs at the board's nominal operating frequency f_{nom} under error-free conditions and a signature configuration collects data from the produced results in certain intervals. It is in the application's nature that at certain points of its run, computations can be repeated if needed. In particular, the application can repeat its sensing algorithm for different GSM channels. That allows for rollback interventions. Data of the error-free run are collected in a file containing timestamps and application-relevant signatures, called ”golden traces”. Next, under normal system operation, signatures of the run are again collected and compared to the golden trace at specific time intervals. The data are stored in a file called ”*signatures.txt*”. The size of the file is less than 2 kilobytes.

Now that the correct results of the application are known to the RAS mechanism, a normal run under PID-controlled DVFS switches can start. An error is sensed when the results computed at the aforementioned specific intervals do not produce the same signature as in the error-free run. We consider the only way that is possible to be a hardware error. To mitigate therefore faulty system operation and ensure functional correctness, we make a rollback to the last correctly computed signature and restart the application from the specific checkpoint. This procedure, however, introduces delay in the overall run and we now run the risk of missing the application's final deadline.

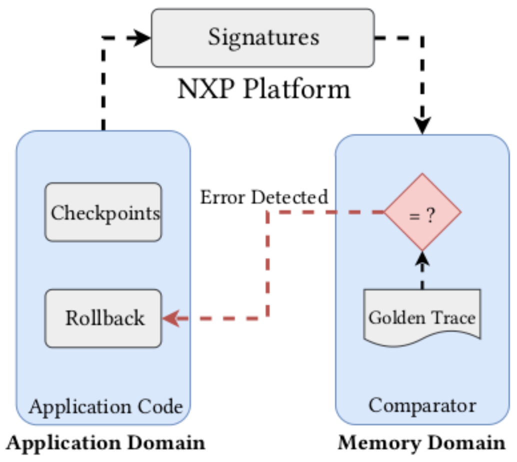

Figure 3.1: The configuration of our RAS mechanism. The system checks for correct signatures against the golden trace and performs a rollback otherwise.

At this point, we introduce a variable our configuration uses to handle time delays.

Definition. *Slack* $(s[n])$ is the difference in execution time for a specific task between an error-free execution, operating at nominal frequency f_{nom}, and the current execution at frequency $f[n]$.

$$s[n] = t_{ref} - t[n] \tag{3.1}$$

where t_{ref} specifies the reference time and $t[n]$ time execution at current frequency. Therefore, a time delay is simulated with a negative value for slack which means that the board needs to speed up in order to catch up while positive slack suggests that we are ahead in time and can therefore slow down in order to consume the least amount of energy. It is evident that optimal use of the scheme will keep the value of slack at 0 at all times.

3.1.2 Custom DVFS driver by Thales

It is apparent by now that the need to switch between multiple values for frequency and voltage is key for our work. However, the default DVFS availability for our target board, the iMX6Q from NXP, is limited. It only offers the nominal frequency of 792 MHz, a slow frequency of 396 MHz and a fast frequency step at 996 MHz. This configuration does not help with our experiments. On the other hand, the board's architecture allows for a maximum of 50 frequency points ranging from 396 MHz to 996 MHz. These points are not accessible with the default drivers of the board. That is why we use a custom driver developed by THALES that enables as many points as we want (up to 50) and handles the DVFS switches. A study on how many points we should choose for optimal performance of our PID can be found later on this Chapter.

We will now present the chosen DVFS points we will use for our experiments. The points used for mitigating the effects of process variability through RAS interventions and maintaining system dependability under excess workload can be found in Table 3.1. For the temperature management experiments we use a different set of points as we want to have specific steps for the applied voltage and frequency. Therefore we choose the points presented in Table 3.2 with a step of around 50 MHz and 25 mV.

3.1.3 PID controller

We will now present the PID controller that decides proper DVFS switches depending on the value of slack. The name "PID" is derived from the initials of "Proportional - Integral - Derivative" controller which describes the following equation used by this type of controller:

$$m[n] = \underbrace{k_p s[n]}_{\text{proportional}} + \underbrace{(s[n] - s[n-1])k_d}_{\text{differential}} + \underbrace{(m[n-1] + s[n]k_i)}_{\text{integral}} \tag{3.2}$$

where s[n] is defined in Equation 3.1. The proportional part, that is directly associated with the gain k_p produces a result that is proportional to the slack measurement. The integral part, described with the gain k_i, contains previous values of the value computed by the PID, therefore constitutes the "memory" of the scheme. The derivative part, modeled with the gain k_d, takes into account the slope of the error and acts as the proactive part of the equation. The computed value, set here as $m[n]$, is the frequency multiplier that will be quantized and rounded towards the closest value that represents a DVFS point in our system.

The PID controller is the industry standard in the systems engineering industry and in workplaces where controllers need to ensure dependability and safety of the user, such as factories and refineries.

Table 3.1: Set of 22 DVFS points for dependability experiments. Highlighted are the default points.

Available DVFS Levels	
f (MHz)	V_{dd} (V)
396	**0.975**
424	0.975
444	1.000
480	1.025
504	1.050
528	1.050
564	1.075
588	1.075
612	1.100
648	1.125
672	1.125
696	1.150
732	1.150
756	1.150
792	**1.175**
816	1.200
840	1.200
864	1.225
900	1.225
924	1.250
956	1.250
996	**1.275**

Table 3.2: Set of 13 DVFS points for temperature management experiments. Highlighted are the default points.

Available DVFS levels	
f (MHz)	V_{dd} (V)
396	**0.975**
444	1.000
492	1.025
544	1.050
588	1.075
636	1.100
696	1.125
744	1.150
792	**1.175**
840	1.200
888	1.225
936	1.250
996	**1.275**

Adding to this, is the easy-to-use nature of the equation. It can be understood and developed by engineers easily and is to this day a very reliable controller, despite many more sophisticated controllers already designed and produced. Next, follow some key characteristics of the function of a PID controller, modified to suit the purposes of our experiments.

The percentage of how much the error surpasses the target value when the PID tries to converge it to the target value is called *overshoot*. In our case, overshoot describes how faster we are operating compared to the error-free run, therefore we are wasting energy. It is vital in this sense to keep overshoot as small as possible and preferably eliminate it.

The time it takes for slack to converge to its target value, within a limit of 10%, from the point where the error first occurred is called *settling time*. Keeping a small settling time will mean the system reacts fast and mitigates errors quickly, therefore is good practice for our system. However, researchers in general find out that minimizing overshoot and settling time at the same time can be quite challenging and is often a matter of trade-offs. Accomplishing both can be done by fine-tuning the PID gains. Our work is presented in Figure 3.2, where we can see various combinations of the gains and select the one with the smallest overshoot and settling time.

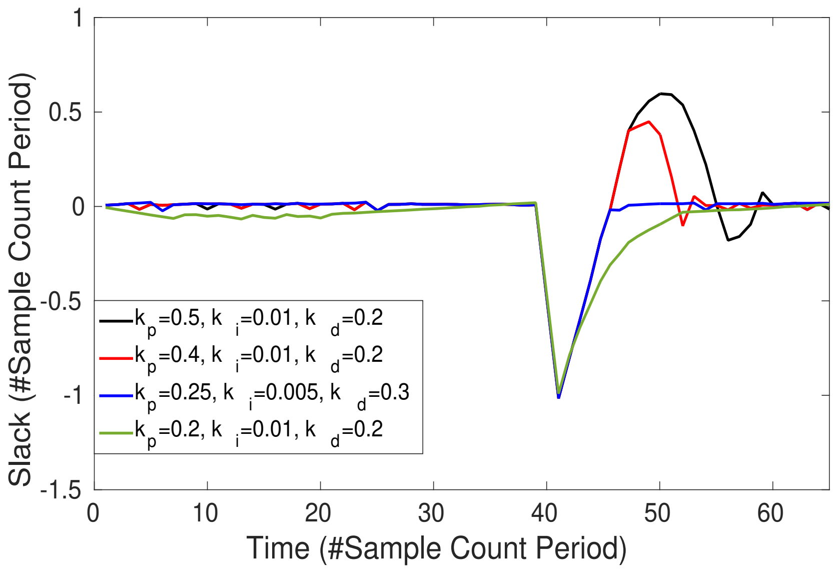

Figure 3.2: Tuning the PID gains to minimize overshoot and settling time

Another point to be noticed in our implementation: the quantization we perform on the frequency multiplier, although necessary for software solutions such as ours, takes away from the analog nature of the PID and introduces limitations to our scheme. As specific frequencies have specific effects on slack (assuming same workload), a need to introduce as many steps as possible arises. We have chosen to compare a varying number of DVFS points against the settling time they can accomplish. In other words, given that we are able to change the number of DVFS points with our custom driver, we try to achieve maximum performance for our scheme. The results of 10 measurements per number of points are presented in Figure 3.3 for a 95% confidence interval.

We notice that the default configuration of the board with 3 DVFS steps cannot achieve any settling of slack, as frequency will be either too fast or too slow. We also notice that after we pass 17 points, settling time is barely affected because we have achieved the minimum granularity needed for our experiments. In conclusion, we select to choose at least 17 points for our dependability experiments,

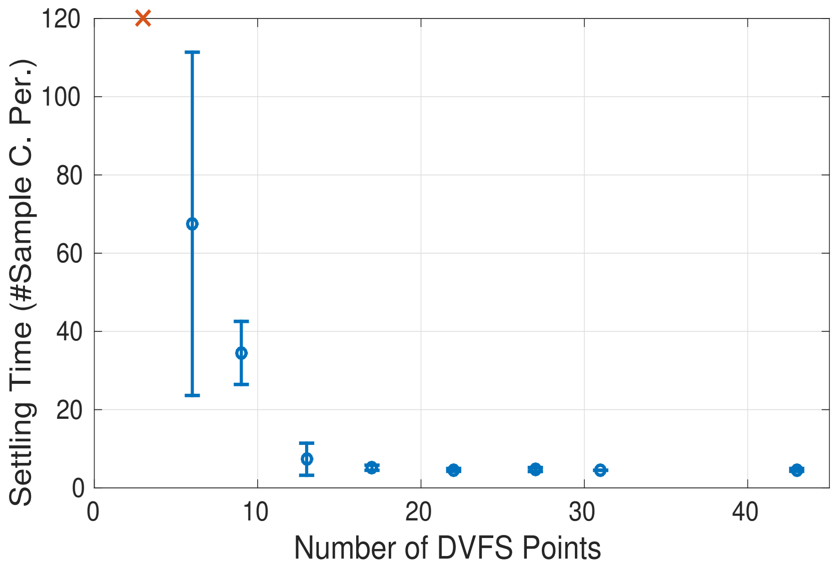

Figure 3.3: Number of DVFS points compared to the mean settling time they can achieve for a 95% confidence interval.

while noting that the 13 points, chosen for the temperature experiments, also achieve acceptable settling times.

3.1.4 Energy estimations

After presenting and discussing the performance of our PID-controlled DVFS mechanism for various experiments, we intend to compare our work with novel industry-related approaches that aim to accomplish the same goals. Given how important energy consumption is for modern embedded systems, it is unavoidable that we compare the energy consumption of our effort and the competing approach, at least for the dependability experiments. It is common knowledge that any effort to challenge a well established approach must always stay within the energy budget of that approach. Therefore, we need to make some clarifications on how we model the consumed system power.

We take into account both the dynamic power, consumed by wires and transistors in the Register Transfer Level of the chip, and the static power, leaked by the sub-threshold current of the transistors of the chip and add them. The equation used to model the dynamic power consumption is:

$$P_{dyn}[n] = \alpha C V_{dd}^2[n] f[n] \tag{3.3}$$

where α is the activity factor, represents the probability that the circuit node transitions from 0 to 1 and depends on the workload. C is the capacitance coming from the wires and transistors in a circuit and $V_{dd}[n]$, $f[n]$ stand for the supply voltage and frequency at the current instance[24]. All constants are kept the same for all experiments and therefore do not affect the result. In addition, static power is modeled with:

$$P_{static}[n] = V_{dd}[n] I_0 e^{\frac{-V_{th}}{\eta V_T}} \tag{3.4}$$

where I_0 is the sub-threshold leakage, η is a technology parameter and V_τ is the thermal voltage. Therefore in static power we also model a linear dependence with voltage supply[24]. Finally, total energy consumption for all discrete n steps is the sum of static and dynamic components (Equation 3.6) can be calculated as in Equation 3.5.

$$E = \int_0^T P(t)dt = \sum_{m=1}^{m=n} P[n_m]\Delta t[n_m] = P[n_1]\Delta t[n_1] + P[n_2]\Delta t[n_2] + ... + P[n_n]\Delta t[n_n] \quad (3.5)$$

where

$$P[n] = P_{static}[n] + P_{dyn}[n] \quad (3.6)$$

At this point, we need to point out that while our energy estimations take into account dynamic and static power, they do not account for the energy overhead required to switch between 2 DVFS points. It is our understanding that the bigger the difference between the 2 points, the higher the overhead, however, calculations based on this overhead are beyond the scope of this thesis since we lacked sufficient data for such computations.

3.2 Freescale board and experimental setup

We are now ready to present the target board we use to conduct our experiments. It is the iMX6Q-SABRE-SD developed by Freescale and NXP[6]. It has a quad-core ARM-CORTEX-A9 with nominal frequency at 792 MHz. The board runs a Linux kernel, version 3.0.35. The boot-loading sequence is run from an SD card. Among other sensors of the board, we utilize the temperature monitor located on the CPU[62]. Detailed instructions to set up the connection and the board can be found in Appendix A.

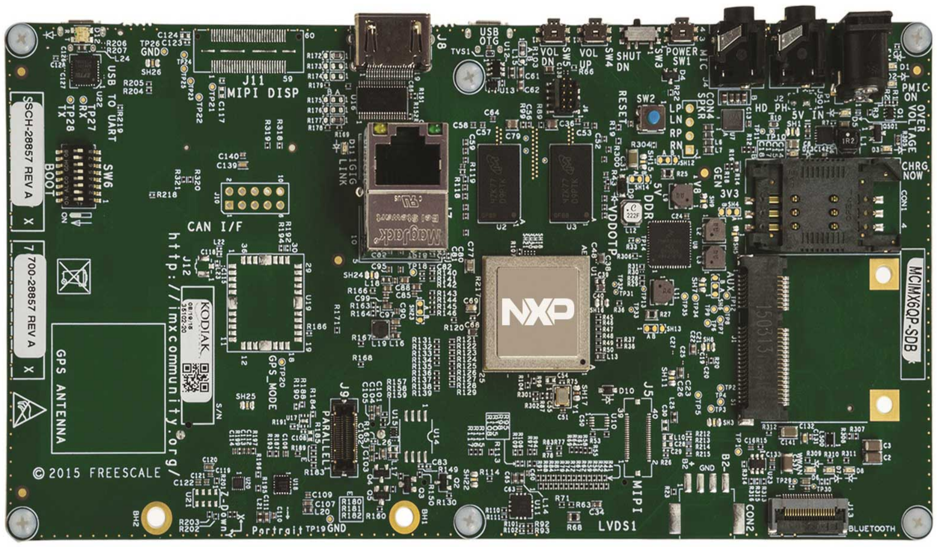

Figure 3.4: The iMX6Q board

3.2.1 Ringing

At this point, we should stress the benefits of our work being tested and run on a real platform and distinguish ourselves with simulations run on a computer. Our scheme takes into account all the side effects a real-world application faces and aims to handle them. One of those effects is the fact that when running a computation-heavy and time sensitive application such as ours one needs to take into consideration that the estimated execution time will be affected by other processes of the operating system such as data transfers through the SSH connection, memory I/O, stack calls etc. Ethernet connections and data traveling through them can also add small delays. Therefore, when comparing the timestamps in our application we will see variation around the expected value which represents small workload applied by the OS.

In some cases, the OS may need to set up another background process to satisfy an SSH connection, for example, which will make slack slightly more negative. In other cases, the OS closes processes that have ended, thus freeing hardware sources for our main application which makes slack slightly more positive. In both cases, the PID is monitoring slack at all times and makes decisions on whether or not to switch to a different frequency as to return slack close to 0.

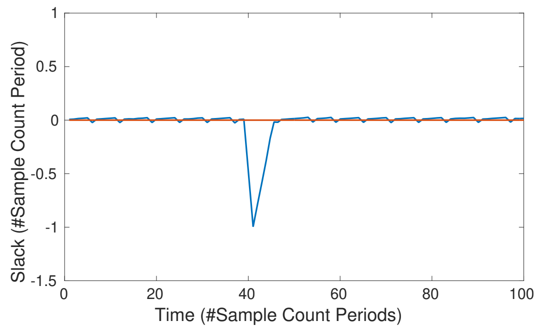

Figure 3.5: Slack forced close to 0

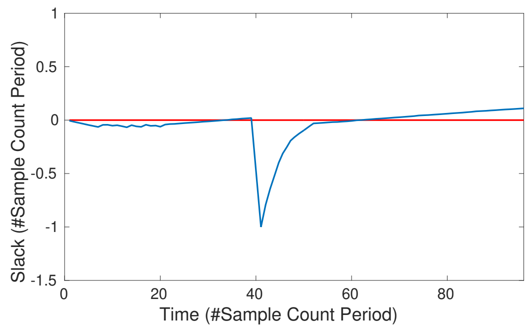

Figure 3.6: Slack without strict forcing to 0

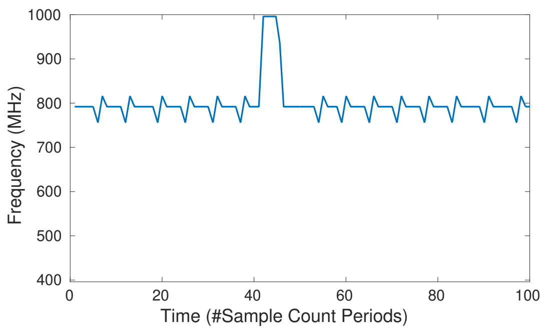

Figure 3.7: PID decisions: Ringing

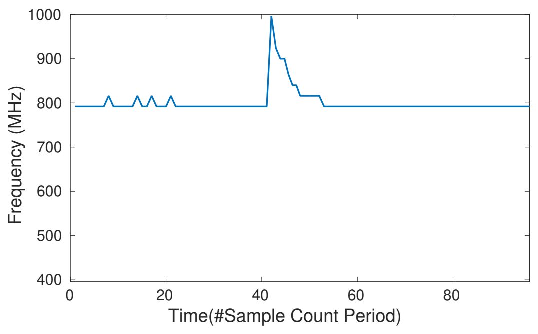

Figure 3.8: PID decisions: Minimal Ringing

As we can see in Figure 3.6, slack variates slightly from 0 either towards negative or positive values. Depending on the configuration of the PID gains, we can choose how reactive the controller is towards these variations. In Figures 3.5 and 3.7 we see an approach that mitigates these variations and keeps slack close to 0 at the expense of more DVFS switches. These DVFS switches around the nominal frequency, that only serve the purpose of keeping slack close to 0, are called *ringing effect* and are a

”necessary evil” of our scheme that arises from the real-world nature of our effort. Nevertheless, it is up to the user to decide how evident ringing will be. In applications where slack can be within a large margin of its target value without problem, ringing can be minimal as depicted in Figure 3.8. For the remainder of this thesis we intend to use a very reactive PID similar to the one in Figure 3.7.

3.2.2 Thermal management on the Freescale board

As discussed earlier, we intend to test how a PID-controlled DVFS scheme can handle system temperature. The idea, that is based in discussions in Chapter 2, is to operate as fast as possible without surpassing a specific temperature or act as an emergency mechanism to return the system temperature to safe levels in case the system has overheated.

We monitor the temperature of our quad-core CPU with the on-board temperature monitor[62] which provides a mean value of the temperature of all cores. The developers of the board deem that this approximation is enough for our type of experiments[62], especially because there is a metal lid on top of the CPU which prevents the creation of thermal hot-spots and evens out the temperature. The command used to read the temperature monitor through the OS is:

```
cat /sys/class/thermal/thermal_zone0/temp
```

For clarification, this part of our work is not associated with the HARPA application. Instead, the PID controller is instantiated around the recorded temperature. We keep using the custom DVFS driver of Thales, however, as it is tightly connected with our ability to efficiently control the temperature through the DVFS points presented in Table 3.2. After a target temperature T_{tar} is selected by the user, the system temperature is monitored through the sensor and an error is calculated:

$$e[n] = T_{tar} - T[n] \tag{3.7}$$

All of this leads to a new equation for the PID controller with new k_p, k_i, k_d gains different from the ones used in the dependability analysis:

$$m[n] = \underbrace{k_p e[n]}_{\text{proportional}} + \underbrace{(e[n] - e[n-1])k_d}_{\text{differential}} + \underbrace{(m[n-1] + e[n]k_i)}_{\text{integral}} \tag{3.8}$$

where t[n] symbolizes the error between the recorded temperature and the target temperature. A full temperature profile for our board is part of our future work and beyond the scope of this thesis.

The workload that is used to elevate the temperature of our board is developed by THALES and performs power analysis of GSM channels similarly to the HARPA application. It is instantiated on all 4 cores to elevate the temperature as much as possible.

Chapter 4

Experimental results

4.1 Mitigating the effects of process variability

The first type of experiments we present is how DVFS mitigates the effects of RAS-induced rollback delays. Taking into consideration the mechanism presented in Chapter 3, when an error is injected, a delay of 1 Sample Count Period is monitored by slack and the PID boosts the operating frequency in order to speed up the application and catch up.

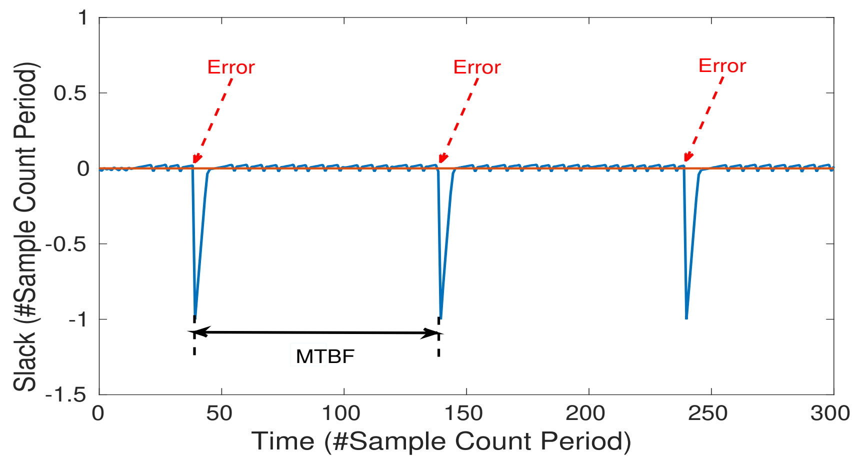

Figure 4.1: Slack over time after rollback

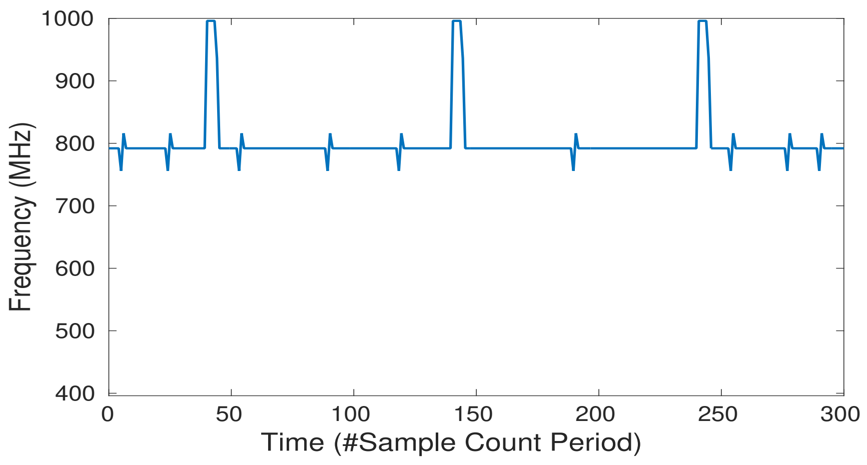

Figure 4.2: Frequency over time after rollback

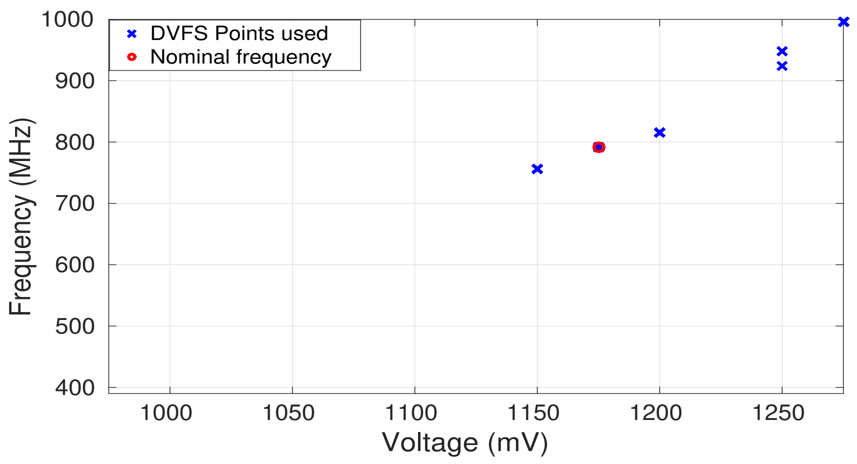

Figure 4.3: DVFS points used by the PID

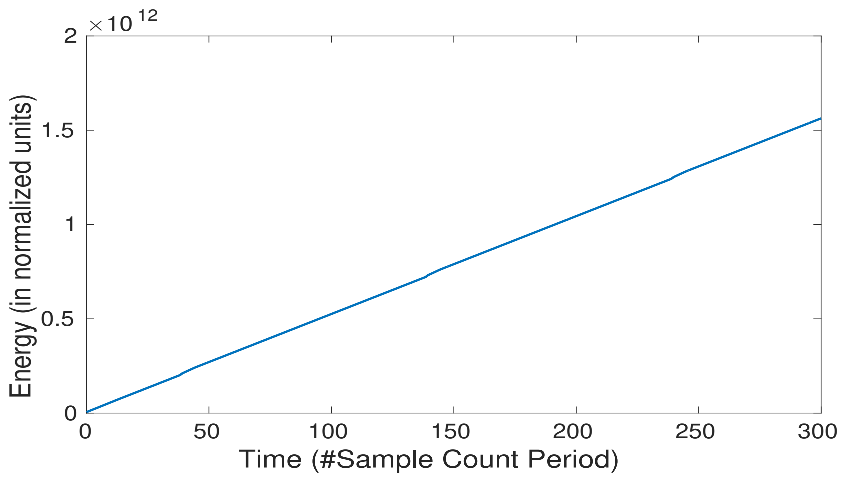

Figure 4.4: Total energy consumed for 3 rollbacks

In Figure 4.1 we have pointed to the points where we injected the errors and caused the delay. We notice through the Figure 4.2 that the PID immediately switches to the maximum frequency for as long as necessary for the slack to return to 0 as fast as possible. We should note that it is the user's choice to tune the gains of the PID for minimum settling time and maximum performance and tolerate possibly bigger energy consumption. In another configuration, we could tolerate bigger settling times and therefore worse performance but further minimize the energy consumed during the transition period of the controller. In any case, it is a matter of trade-offs and slower response times do not guarantee smeller energy consumption. In Figure 4.3 we present the exact pairs of frequency and voltage the PID chose to switch to. In Figure 4.4 we present the total energy consumed by the system for the time frame and the error rate depicted in Figure 4.1 based on the Equation 3.5.

4.1.1 Minimum MTBF

At this point, it is evident that the PID-controlled DVFS scheme we propose has a significant limitation in how fast it can respond to errors. Given that the system's maximum frequency is 996 MHz there is a certain minimum Mean Time Between Failure (MTBF) the system can handle, in which the PID

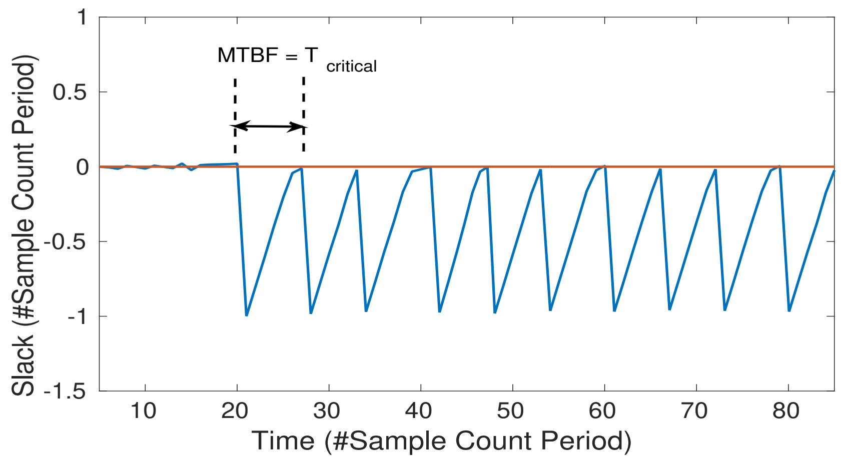

Figure 4.5: Slack at the minimum MTBF our scheme can handle

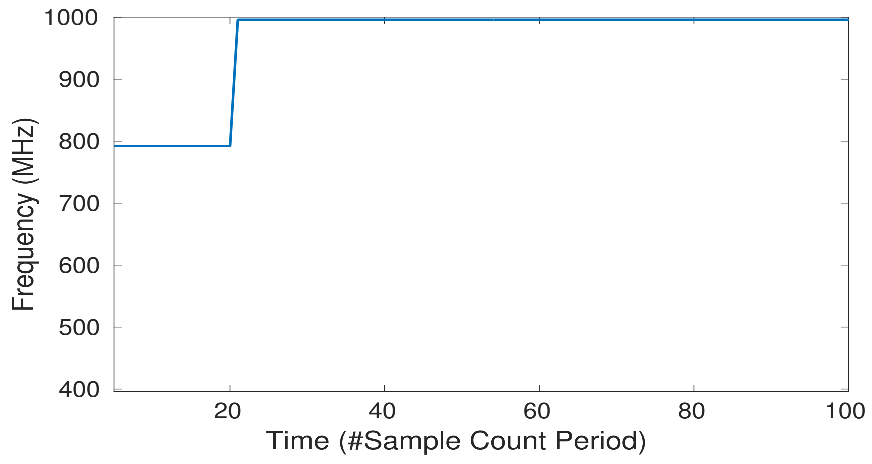

Figure 4.6: Frequency when operating at the minimum MTBF

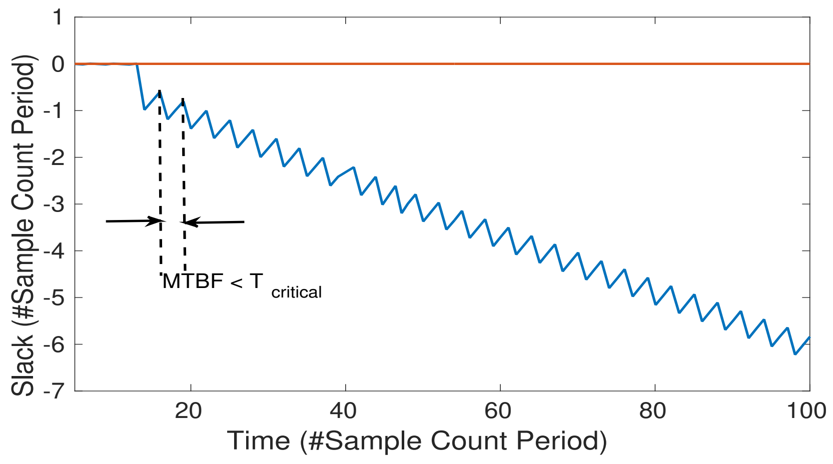

Figure 4.7: Slack when we operate at a failure rate faster than we can handle. Dependability is not guaranteed

barely manages to converge slack to zero while the frequency is unavoidably set to 996 MHz due to constant rollbacks. This operation is presented in Figures 4.5 and 4.6, where we can see that while the

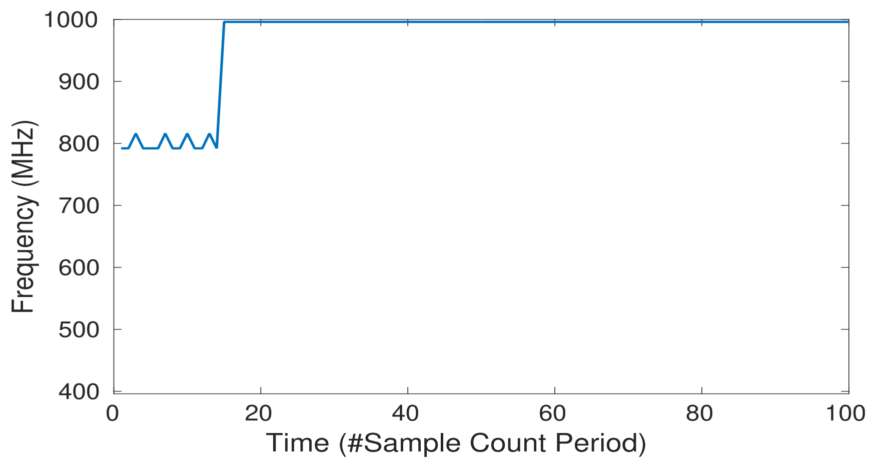

Figure 4.8: Frequency when operating at an MTBF less than the minimum we can handle

system frequency is at maximum 996 MHz, the slack only reaches zero before a new rollback causes it to fall again.

In this scenario, the dependability of the system is challenged, as the proposed scheme fails to meet the deadline constraints while in the transitional period. However, it needs to be pointed out that accelerated tests under elevated temperatures conducted on the board have suggested that the average MTBF of the board is higher than 10^5s[61]. Therefore, for operating in normal temperatures and voltage levels we can expect significantly lower error rates. That is why the errors discussed here are in fact *software-injected* and this scenario is only an extreme case.

In Figures 4.7 and 4.8, we present an operation where the rate at which we were injecting errors was much higher than the system could handle. That led slack to keep falling constantly to more negative values, despite running at the highest frequency, and dependability of the system was not ensured.

4.2 Managing dependability under extra workload stress

The next set of experiments we conducted aims to investigate how the system responds to extra applied workload that introduces performance variability, in the sense that the strict deadline constraints may not be met by the system. Therefore, the PID controller need to adjust the operating frequency to satisfy both workloads and meet the deadlines by keeping slack close to 0. An easy solution for our problems would be to set the frequency to the highest value and make slack positive while at the same time spending unnecessary energy. However, we make an effort to not waste excessive energy as the PID will find the minimum operating frequency to satisfy both workloads. At this type of experiments we assume error-free conditions, however, should rollbacks occur, PID operation is not affected and dependability will be ultimately managed.

As we can see in Figures 4.9 and 4.10, when the extra workload is applied it creates a negative slack that forces the PID to boost the frequency to higher levels. Ultimately, slack settles near zero at 894 MHz as this is the frequency the PID found to be able to handle both workloads. When the workload

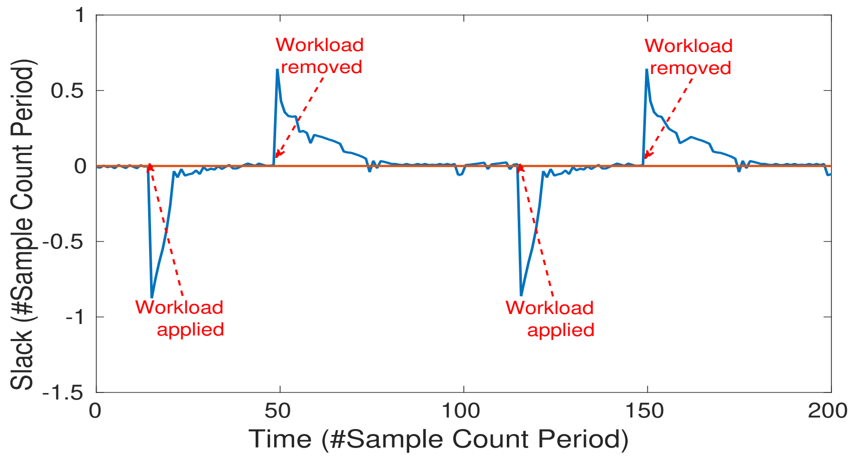

Figure 4.9: Slack over time for extra workload

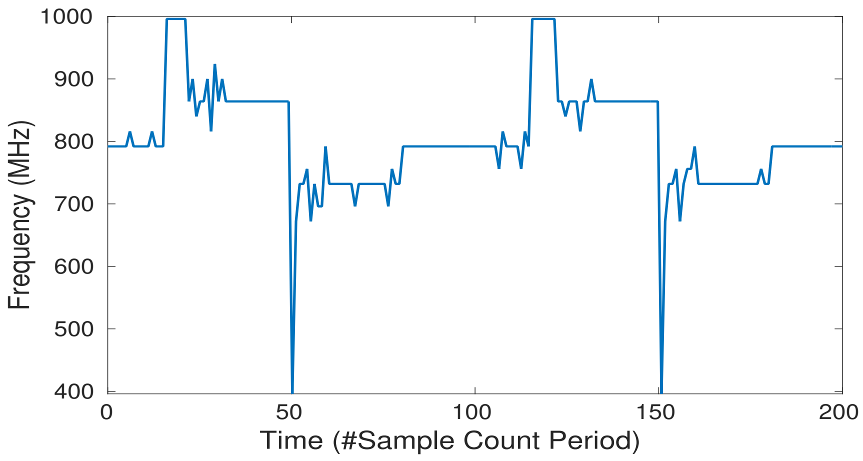

Figure 4.10: Frequency over time for extra workload

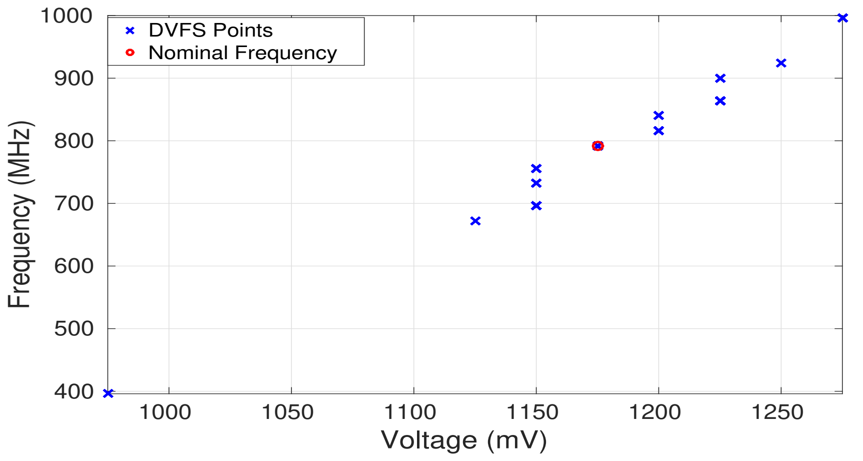

Figure 4.11: DVFS points used by the PID for extra workload

is removed, slack immediately is raised to positive values because the system operates in unnecessary high frequency levels. In turn, the PID will slowly slow down to the nominal frequency returning slack to 0 at the same time, thus saving power. The decisions of the PID are presented in Figure 4.11 and the

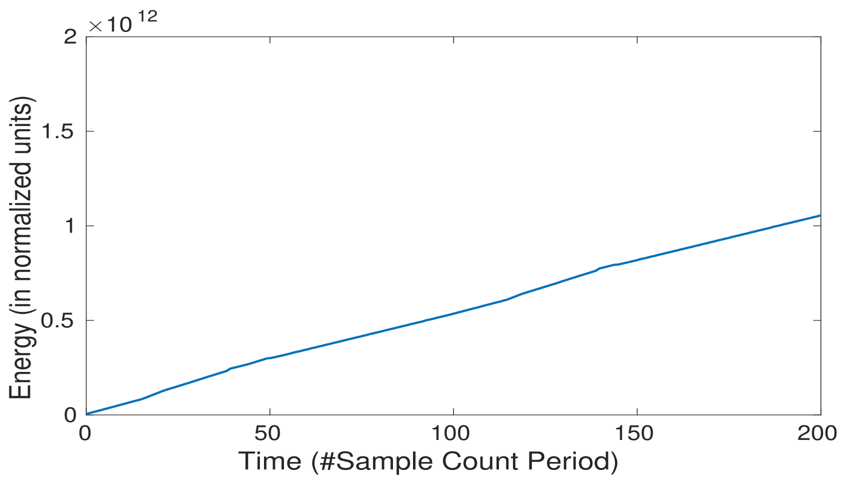

Figure 4.12: Total energy spent under workload stress

total energy consumed for the duration of 2 consecutive experiments is presented in Figure 4.12.

4.3 Comparison with the *"conservative"* Linux CPUFreq governor

We move on to discuss how our scheme compares to novel, industry-related techniques that perform DVF scaling to meet timing constraints and handle workload-related variability. The algorithm we challenge ourselves with is based on the *CPUFreq* governor of the Linux kernel, and more specifically the *"conservative"* mode of the governor[28]. A flavor of the governor is implemented for the target board and presented below.

The main idea behind the "conservative" governor is to perform the minimum required, cautious DVFS on the system depending on the delays it faces. Because of that, it will perform a graceful DVFS, changing to neighboring frequencies and testing whether this change was enough. It takes energy consumption into account and heavily tries to minimize it. In fact, all other modes of the Linux CPUFreq governor can be outperformed easily either in terms of power consumption or performance.

As per the suggestions of the manual[28], we set key values of the algorithm as follows:

1. *SamplingRate*: the rate at which we check the slack/CPU usage. It is usually measured in microseconds (μsec) and has a value of 10000 or 0.01 secs. In our case, we have selected a value of 1 #Sample Count Period or *Tnom*, which is based on the CPU's performance at 792 MHz.

2. *FreqStep*: the percentage of maximum system frequency that describes the step with which the governor moves to the higher or lower frequencies. It is set by default at 5% but given the capabilities of our custom driver we have selected a step of 2.5% of the maximum frequency or approximately 25 MHz.

3. *Scaling down factor*: the rate at which we perform DVFS switches. In our case we have set it equal to the SamplingRate, which means we perform any necessary DVFS switches every time we check the slack.

4. *UpThreshold*: the upper limit of slack at which the governor will need to decrease the frequency. We have set it at 15% of target slack or 0.15 #Sample Count Periods.

5. *DownThreshold*: the lower limit of slack at which the governor will need to increase the frequency. We have set it at -15% of target slack or -0.15 #Sample Count Periods. Therefore, if the slack remains between the UpThreshold and the DownThreshold the governor will consider that there is no workload and return the frequency to nominal.

A comparison of the CPUFreq governor and our PID is presented below.

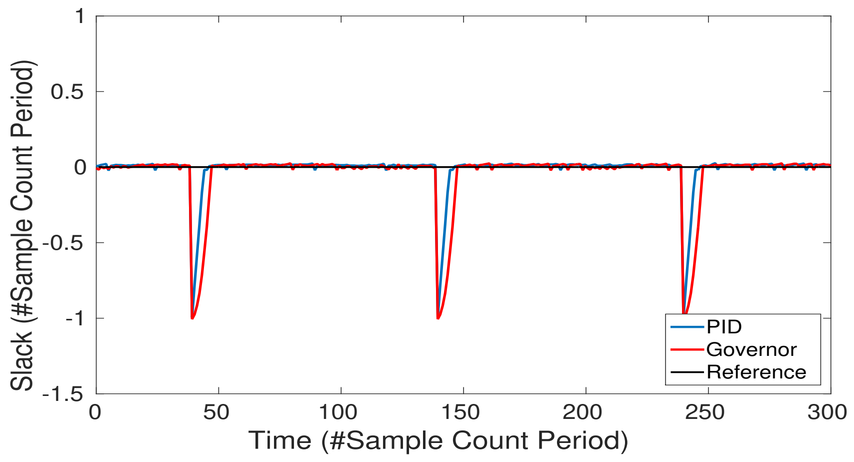

Figure 4.13: Slack comparison for the PID and the CPUFreq governor

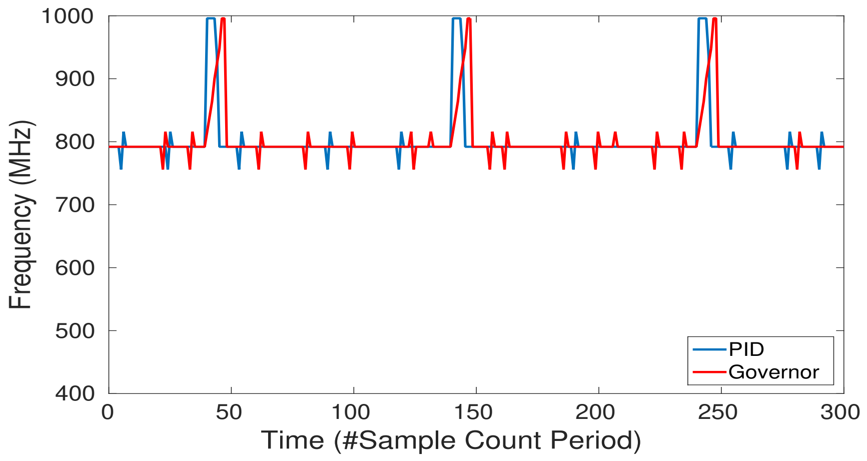

Figure 4.14: Frequency comparison for the PID and the CPUFreq governor

We notice in Figures 4.13 and 4.14 that the PID handles the injected errors faster that the governor, as the smooth transition to higher frequencies of the governor gives a time advantage to the PID that jumps to the necessary frequency immediately. We notice that it takes 6 #Sample Count Periods for the PID to return slack to 0, while the governor needs specifically 9, resulting in a performance gain of 33.3%. In Figure 4.15, we notice that the governor accesses more DVFS points in order to mitigate the errors while the PID stays in the necessary frequencies. This leads to the total energy consumed

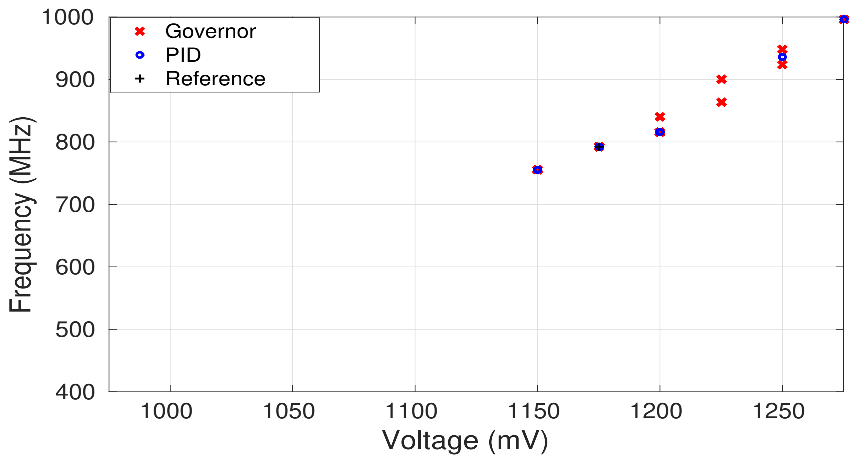

Figure 4.15: DVFS points used by the PID and the CPUFreq governor

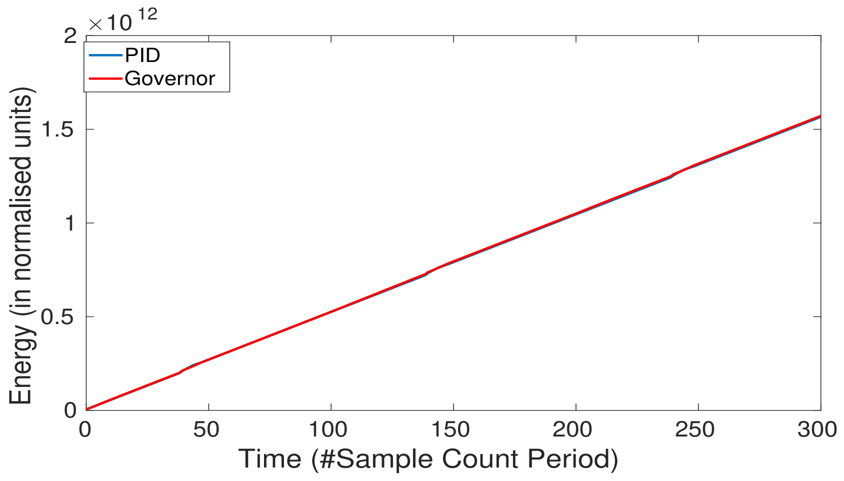

Figure 4.16: Total energy spent by the PID and the CPUFreq governor

by the two methodologies to be almost the same, with any difference noticed being negligible, as seen in Figure 4.16. Therefore, our scheme achieves better performance while remaining within the energy budget of the governor.

4.4 Thermal management using DVFS

Utilizing the board's temperature sensor, we proceed with expanding the use of a PID-controlled DVFS idea to control the board's temperature and keep it within safe limits. We apply a computation-heavy workload on all cores to elevate the temperature of the system and monitor it every 10 seconds[1]. We also change our DVFS configuration to the one presented in Table 3.2.

We have assumed 2 different uses of our scheme. In the first one, the PID will act as an emergency mechanism and will attempt to lower the system's temperature as fast as possible from a dangerously high temperature to a preselected target temperature that corresponds to a "safe-to-operate-in" tempera-

[1] A full-scale timing analysis of how often we should check the temperature and how often we should change DVFS states is beyond the scope of this thesis and part of our future work

ture. In the second use, the PID will always remain active and will attempt to not allow the temperature to exceed a preselected upper temperature which corresponds to the highest safe temperature of the system.

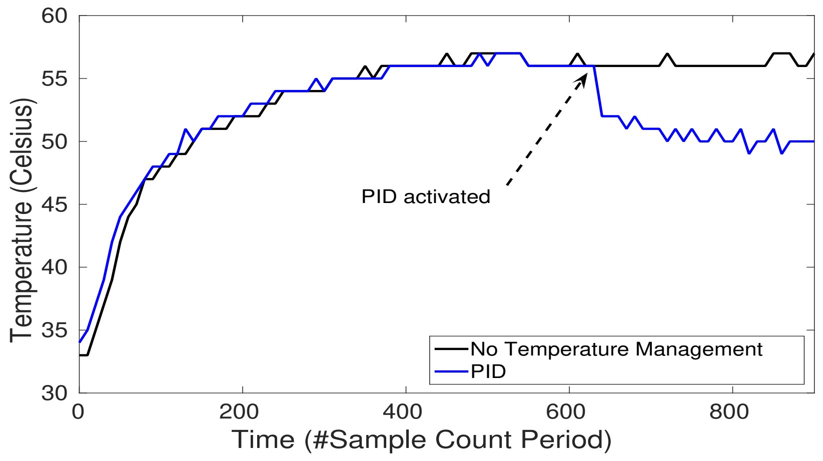

Figure 4.17: PID-controlled DVFS acting as cooling

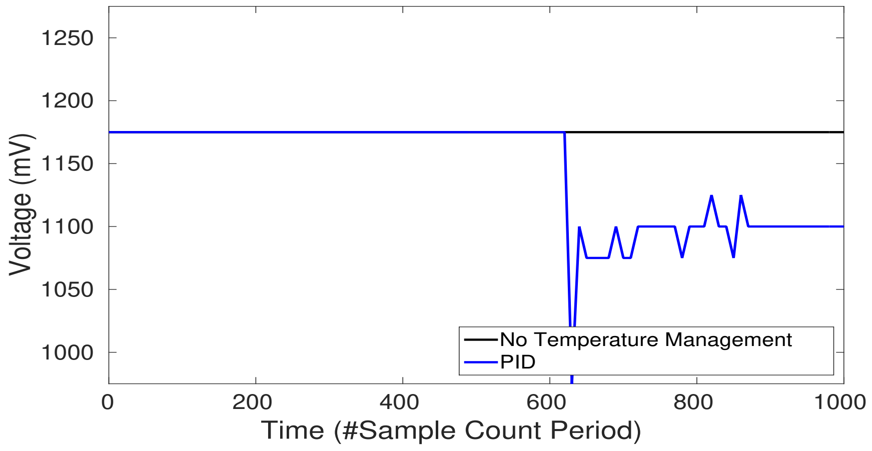

Figure 4.18: Voltage applied by PID-controlled DVFS acting as cooling

A demonstration of the capabilities of the PID controller is presented in Figure 4.17. We initialize the PID at around 650 seconds after the system was running at nominal voltage and set the target temperature at $50°C$, which we already knew it had surpassed. From Figure 4.18, we see that the PID immediately decreases the applied voltage to decrease the system temperature as fast as possible. The PID ultimately decides that in order to remain at $50°C$, the optimal voltage is around 1 Volt. The performance of the PID for the second case is best demonstrated against an already published method to control system temperature by Zhou et al.[35].

4.4.1 Comparison with a temperature manager

Zhou et al.[35] proposed a temperature manager to keep the system's temperature from exceeding a specific target temperature. Their method was instantiated on the same board we conduct our own experiments which allows for direct comparisons and replicating their results. A flow chart of their algorithm is presented in Figure 4.19.

The main idea behind their algorithm is to monitor the system temperature using the onboard sensor and compare it with a target temperature T_{tar} which they have preselected. When the operating temperature reaches the T_{tar}, the manager starts operating and decreases the operating frequency and voltage. From that point on, when the temperature drops by $1°C$ the manager raises the frequency and voltage to the next available point. When the temperature is raised by $1°C$ the manager drops the frequency and voltage. This leads to the temperature being averaged out at the T_{tar}.

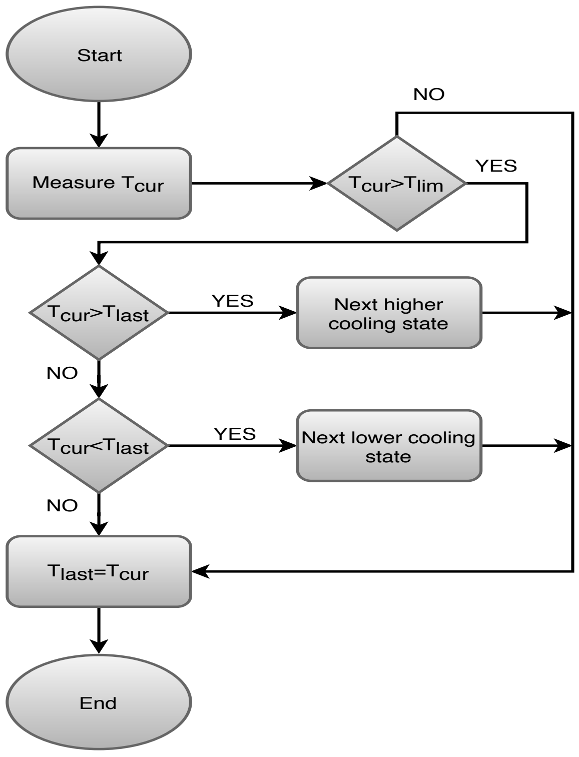

Figure 4.19: The thermal manager proposed by Zhou et al [35].

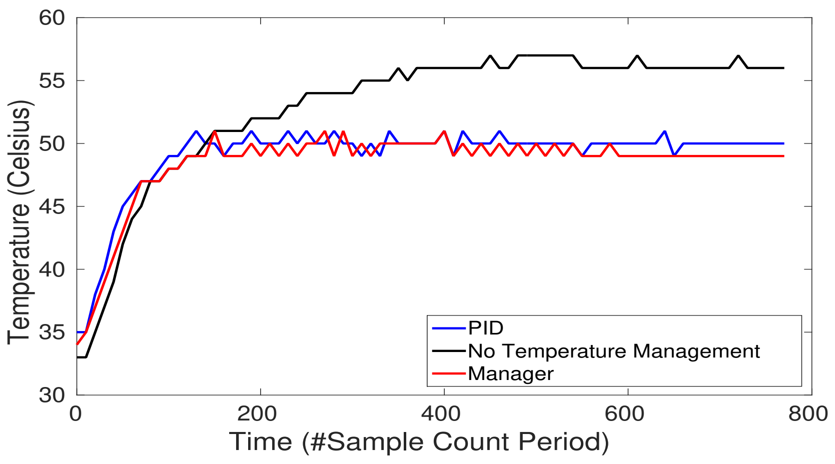

Figure 4.20: PID versus thermal manager by Zhou et al[35].

In Figure 4.20, we can clearly see that both schemes can efficiently prevent the system temperature from exceeding the T_{tar} of $50°C$. However, it should be pointed out that the thermal manager is not designed to act as an emergency mechanism to cool down the CPU. If the system ever finds itself in a temperature higher than the T_{tar}, the algorithm cannot be applied. This is due to the design of the

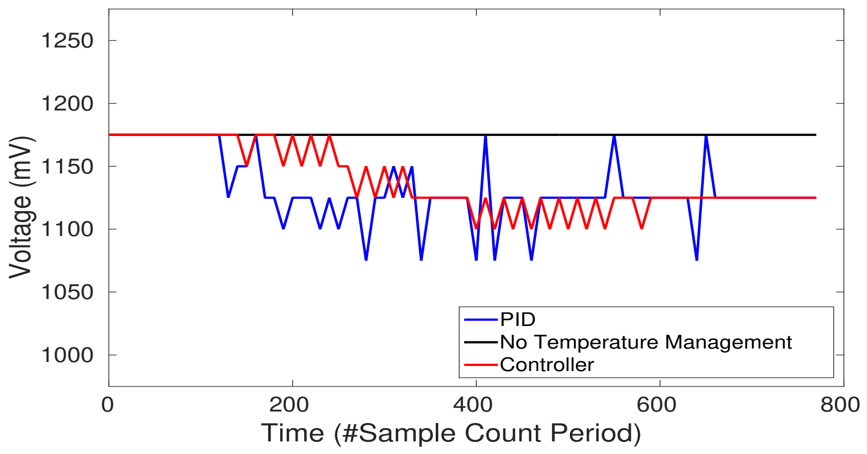

Figure 4.21: Voltage decisions by PID and thermal manager.

algorithm and is not an oversight of the authors. This proves that the PID-controlled DVFS is a more generic algorithm than the thermal manager and can act both as a proactive, preventing mechanism to keep temperature at safe levels and as a reactive, emergency mechanism to cool down the CPU to acceptable temperatures.

Chapter 5

Conclusion and Future Work

In this thesis we presented a PID controller that utilizes DVFS techniques to handle timing constraints in embedded systems caused by either RAS-related rollback events or dynamic workloads. In both cases we showed that the PID controller can mitigate their effects while at the same time being aware of the consumed energy. We compared our controller with a version of industry-standard, workload-aware Linux kernel governor, which is widely used in CPUs worldwide. We showed that while managing to stay within the governor's energy budget we achieve 33.3% better performance gains.

Moreover, we presented a version of the PID controller that aims to manage system temperature. We showed its capabilities both as an emergency cooling mechanism and as a preventing mechanism that will stop the CPU from overheating while operating at the maximum possible frequency. We compared with previously published techniques and managed to show that the PID acts as a generalization of the proposed schemes.

The key point of this thesis is that it does not limit itself to simulations of the proposed scheme but rather uses a real-world embedded platform to test itself on. This provides experimental data for future researchers.

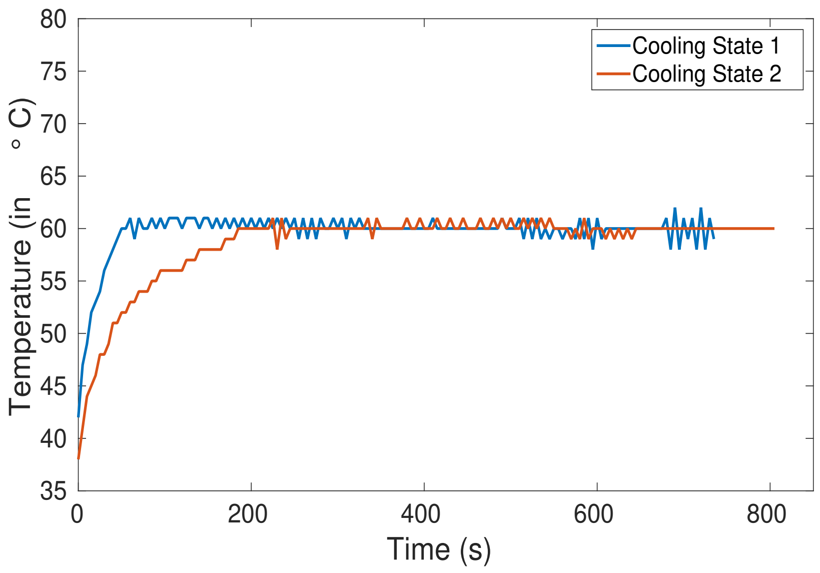

Figure 5.1: Cooling State 1 represents all 4 cores being active where Cooling State 2 represents a hot-plug mechanism with one CPU core inactive.

There still are things to be tested and documented regarding a PID-controlled DVFS scheme that will manage system temperature. Firstly, there should be a full temperature profile of the iMX6Q board depicting temperatures and voltages that cause them for a number of workloads. Secondly, an integration of the PID controller with other state-of-the-art techniques, such as CPU hot-plug, could be studied. As seen in Figure 5.1, PID-based temperature management in parallel with CPU hot-plug presents an interesting topic for scientific research.

Furthermore, there should be a recalculation of the above mentioned energy consumptions adding power lost during the DVFS switch. This will provide better energy comparisons. Finally, a timing analysis of the PID temperature controller should be studied discussing how often we should perform DVFS switches given that temperature takes a while to dissipate.

Appendix A

Setting up the NXP Board

In this chapter we present the exact steps we followed to establish an SSH connection between the laptop (hereafter mentioned as ”the host”) running Ubuntu Linux 16.04 LTS and the i.MX6Quad SABRE-SD NXP board[6] (hereafter mentioned as ”the target”). We start with building a working SD card image that contains the bootloading sequence built by the Linux Target Image Builder (LTIB)[1]. We have also acquired the root file system (along with all relevant BSP files) that will be loaded on the target and the kernel image with a working DVFS driver from Thales[38] that should run on the target.

A.1 Description of the experimental setup

Our main goal here is to test the application on an embedded platform and specifically, for experimental purposes, on the NXP board. Moreover, the application needs to run on top of a Linux operating system (and not on the ARM processor’s bare metal) since it performs numerous system calls. Therefore the board should have a Linux kernel and a root file system at its disposal. Moreover it needs a bootloader binary that implements a bootloading sequence which will identify the kernel and the file system. Using the LTIB tool and taking into consideration the official NXP user guide[64], we build the Linux 3.0.35 kernel configured specifically for the ARM architecture and our target board. Additionally, we create an SD card image that when written into an SD card, using the instructions below, will serve as a bootloader.

At this point we need to address that it is not appropriate to permanently mount neither the kernel nor the file system onto the board. Such an approach would require valuable hardware resources (processing time from our ARM processor and physical memory from the board) which are limited on our target board. To this end we choose to establish a TFTP connection[2] between the board and the host laptop so that the bootloader can find the kernel from a directory on the host laptop.

Finally we use the same implementation idea to set up the board’s file system. We will treat it as a file system to be exported from the host laptop to the board using the NFS protocol. The NFS protocol allows a computer client(in our case the board) to access a file system that is physically located on a computer server(in our case the laptop) over a computer network(in our case an Ethernet connection)[65].

[1] The Linux Target Image builder, or LTIB for short, is an open-source tool mainly used to develop and deploy Board Support Packages, which include Linux kernels and bootloaders as well as numerous other packages, for a variety of architectures including ARM.[63]

[2] The Trivial File Transfer Protocol is a well established method to transfer files between systems. It is easy to implement and its only drawback might be security of the transfer.

We should note that since both host laptop and target board are computing devices, a crossover Ethernet cable is required to allow bidirectional file transfer between the board and the laptop.

We can, therefore, present the exact steps of the booting up procedure from switching on the board to running commands on the board's terminal.

1. When switching on the board, the bootloading binary in the SD card starts to execute itself.
2. It is set to search for the kernel image binary at a specific directory on the host. That binary is in fact the binary form of the Linux 3.0.35 kernel that we created with LTIB. It is also necessary that the TFTP connection between the board and the laptop is already established, otherwise the bootloader will not locate the relevant binary.
3. While booting, the kernel is set to look for a file system at the /tools/rootfs directory of the laptop. This needs to be a file system already exported through the NFS protocol. This file system was created with the LTIB tool. In order for both the host laptop and the target board to be able to read, write and execute files and executables on this file system, we will need to pay close attention to the permissions of the /tools/rootfs directory and its subdirectories.

A.2 SD card creation, TFTP and NFS setup, connecting to the board

Let us now go into detail in the above steps of the experintal setup. First, one should run the following commands on a terminal:

```
$> sudo apt-get update
$> sudo apt-get upgrade
```

These commands will keep his system up-to-date and make sure he or she does not run into any compatibility errors.

The first thing we need to do is create the SD card from which the target will boot. This image was created by LTIB specifically for the target board. We should insert an empty SD card of at least 8 GB into the host laptop and then open a terminal and run the command:

```
$> lsblk
```

You should identify the name of the SD card. In our case it was "sdb". You may now change to the directory where the -already prebuilt by LTIB binary- *sdcard_image_dvfs_working_NFS_TFTBOOT.img* resides and create the SD card with the command:

```
$> sudo dd if=sdcard_image_dvfs_working_NFS_TFTBOOT of=/dev/sdb bs=4M
```

Note: The ”of=/dev/sdb” part of the command may vary from system to system and that is why we identified the name of the SD card with the ”lsblk” command. This command is going to take a lot of time to run and it does not provide feedback about the progress on the terminal.

We now need to set up a TFTP server so that the board can find and read the *uImage* file that contains the Linux kernel image that the board will run. We set up the server by running the following command:

```
$> sudo apt-get install xinetd tftpd tftp
```

Then we should create a file */etc/xinetd.d/tftp* with a text editor and put in the following entry:

```
service tftp
{
protocol      = udp
port          = 69
socket_type   = dgram
wait          = yes
user          = nobody
server        = /usr/sbin/in.tftpd
server_args   = /tftpboot
disable       = no
}
```

We now have to create the directory that we declared at the server_args parameter, in our case */tftp-boot*, and change it’s permissions so that any user can access it.

```
$> sudo mkdir /tftpboot
$> sudo chmod -R 777 /tftpboot
```

We may now copy the *uImage* file to the */tftpboot* directory and the board will be able to access it and boot the image.

Finally we need to restart the xinetd service[3]:

```
$> sudo service xinetd start
or
$> sudo service xinetd restart
or
$> sudo /etc/init.d/xinetd restart
```

[3] The xinetd service, or extended internet services daemon as it is formally known, is a process that basically acts as a super-server because it listens to all service ports listed in its configuration file and when a request comes in it opens the appropriate server.[66]

At this point our OS on the target has a working kernel but no root file system to work with. It is already set to search for a file system on the host at the specific location */tools/rootfs/* that it will use through the Ethernet connection. Therefore we should set up the location on the host. We start by extracting the *rootfs.tar.gz*. This is the prebuilt file system created with LTIB. It will contain a folder called rootfs. This can be done by simply clicking on the tar file and pressing ”Extract” when the file manager pops up.

We will now create a directory */tools*.

```
$> sudo mkdir /tools
```

We need to change the ownership of the */tools* directory from root to the user. This would also give access to the target’s OS. This is done with the command:

```
$> sudo chown -R <user> /tools/
```

We will also give read, write and executable access to the directory.

```
$> sudo chmod -R 777 /tools
```

We can now copy the *rootfs* folder we extracted in the */tools* folder.

```
$> sudo cp -a <the_folder_that_rootfs_is_in>/rootfs /tools/
```

You should now have a folder */tools/rootfs/* that is owned by the user and not by root. Try to verify that you have successfully executed the above by creating a directory or a txt file without using the sudo command. In any other case use the ”chmod” command to change the permissions of the directory.

Note: We will later realise that the SSH server[4] of the target’s OS needs a specific folder of the rootfs directory to not be writable,readable and executable by everyone. Therefore it is a good idea to change the permissions of the folder. You may use the command:

```
$> sudo chmod -R 444 /tools/rootfs/var/empty/
```

Now that we have our file system ready we need to change some options in the */tools/rootfs/etc/ssh/sshd_config* so that we can connect through SSH to the board. Specifically we need to comment in and modify the following options:

```
PermitRootLogin yes
IgnoreUserKnownHosts no
```

[4] The SSH protocol ensures secure transfer of data between a server and a client using key or password verification. In our case the laptop is the client and the board is the server we connect to. However, the data transfer is bidirectional.[67]

We now have our directories set, with the right permissions, ready to be exported. We should inform the host's NFS server that the folder */tools/rootfs* is allowed to be exported. By opening with a text editor the */etc/exports* file we add the following line at the end of the file:

```
/tools/rootfs *(rw,no_root_squash)
```

This line adds the file system located in the */tools/rootfs* directory in the list of file systems that can be exported to NFS clients. However we still need to initialize the ”master export table” with the contents of the */etc/exports* file.[68] Therefore after we close the file we can run the command:

```
$> sudo exportfs -a
```

We are now ready to set the Ethernet connection through which the target will use the file system we set up earlier. The board is using default static IP addresses. The IP of the board is set at 192.168.0.30 and the IP of the host should be set at 192.168.0.10. We need to edit a new ethernet connection to connect to the board. We should open with an editor the /etc/network/interfaces file and add the following lines:

```
iface <name_of_ethernet_port> inet static
                address 192.168.0.10
                netmask 255.255.255.0
                gateway 255.255.255.0
```

To find out the name of your ethernet port you may run the command:

```
$> ifconfig
```

Usually it is ”eth0” or, in our case, ”enp2s0f0”.

Right after we connect the Ethernet cable and everytime we connect to the board, we should run the following command:

```
$> sudo ifup <name_of_ethernet_port>
```

We should now be able to connect to the board with an SSH connection. We need to switch on the board, wait for about 60 seconds and from the same terminal we executed the above command run:

```
$> ssh root@192.168.0.30
```

The password is root.

If something has gone wrong and after following the instructions you still were not able to connect with SSH to the board we offer some advice that could potentially help you.

You can use the following commands to install the PuTTY client[5] and establish a serial connection to the board so that you will be able to see the progress of the booting-up process and identify where the exact problem is.

First of all, after PuTTY has installed successfully, you may open up a terminal and run the following commands:

```
$> sudo chmod a+wrx /dev/ttyUSB0
$> sudo ifup <name_of_ethernet_port>
```

Then you should proceed by opening up the client with sudo privileges (**$> sudo putty**) and choosing the option ”Serial”.Afterwards, at serial line type ”/dev/ttyUSB0” and finally select a speed of 115200. You should now be able to connect to the board. You will be able to see the booting process and identify at which point the error occurs.

If you try to connect via SSH and the board does not recognize the password ”root”, you can follow these steps to ensure that the password is set correctly.You may connect to the board with a serial connection via PuTTY with the procedure described above. Then proceed to execute the following command:

```
$> passwd root
```

This command allows you to change the password of the ”root” user. Type in the new password and verify that you have commented-in the following options in the /etc/ssh/sshd_config file:

```
PermitRootLogin yes
IgnoreUserKnownHosts no
```

You should now be able to log in as root via SSH.

A.3 Cross-Compiling the application

At this point you should be able to connect to the board via SSH and navigate its file system. We will now move on to the configuration of the host laptop in order to be able to cross-compile the application and successfully run it on the board.

In order to successfully run the application we need a certain version of C libraries and the correct version of the gcc compilers for compatibility reasons. That means that the pre-built libraries and compilers that come with modern standard installations of Linux distributions are incompatible. We used the Linaro cross-compiler toolchain with gcc version 4.6.2 and glibc[6] version 2.13. The glibc version

[5] PuTTY is an open-source SSH and telnet client developed for various operating systems.[69]

[6] The glibc library, or the GNU C library as it is formally known, is the most widely used collection of the ”standard C libraries” used on a Linux system. It is considered a low-level prerequisite when using libraries in an application and is

in particular must completely match the glibc version on the board. Otherwise the application will not find compatible linkers and dynamic loaders as well as compatible libraries included in the source code and ultimately not run, despite the compilation on the laptop being successful. Another thing we need to do in preparation is move the HARPA application found in Bitbucket repositories to a directory that we have read and write permissions. For example we are going to use the laptop's Desktop. In this case we set, for reference reasons, a macro as:

$HARPA = /directory/to/laptop's/Desktop/harpa.git

The first thing we need to do is cross-compile the DVFS driver created by THALES. Change to the directory of the driver on the laptop with the following command:

$> cd $HARPA/src/dvfs_driver

Open with a text editor the recompile_locally.sh file. This is the main bash file we will run to set the paths necessary for the makefiles to run. You might have to edit the LTIB_BASE, CROSS_COMPILE and KDIR variables depending on your system configuration. The KDIR variable points to the location of the Linux 3.0.35 kernel created by LTIB specifically for our Freescale board. We highly suggest that you copy or download the exact kernel we used or copy our configuration file because there is a high chance the application will not run on a slightly different configuration of a kernel. Our intent is to build a DVFS (Dynamic Voltage and Frequency Scaling) driver that will replace the system's default driver thus allowing more precise control of the voltages and frequencies in use. The exact path of the kernel is needed because our makefiles build and load kernel specific modules that allow this modification of the hardware's operational frequency and voltage.

The second variable we need to pay attention to is the CROSS_COMPILE variable. This variable is set to point to the exact location on the host laptop where the selected cross-compiler resides. We have chosen the *linaro* toolchain that uses the *arm-fsl-linux-gnueabi-** cross-compilers. The name of the cross-compiler provides information about various characteristics of the produced executables. In our case the use of the *arm-fsl-linux-gnueabi-** compiler implies that our executables should run on ARM architectures, that they expect to run on a Linux operating system (and not on bare metal) and that they expect to find GNU support. This entails that they expect to find runtime dynamic loaders and linkers and the C standard libraries on the operating system, all on a compatible version with the libraries and compilers that they were built with. Here lies the main advantage of using a prebuilt toolchain like *linaro*. It uses a legacy version of the gcc compiler, specificaly the 4.6.2 version, and not the host laptop's gcc compiler which could be newer and therefore incompatible with the configuration of the board. Moreover it uses the glibc version 2.13 which provides the exact same dynamic loaders, linkers and C libraries that the board uses.

After you set these variables to your desired directories and add them to your system's $PATH variable, you can run the recompile_locally.sh script.

therefore used by a plethora of programs.[70]

```
$> export $PATH=CROSS_COMPILE:$PATH
$> ./recompile_locally.sh
```

It will build the driver executing all necessary makefiles. It will create three static libraries, specifically libdvfs.a, libhds3_linux_ccm_analog.a and libhds3_linux_i2c.a, that are set to be automatically copied to the directory of the main application. They contain all object files necessary to implement the PID controller and modify the board's frequency and voltage.

The next step in preparing the application before copied into the board's file system is to build the main application. After you have built the driver change to the directory of the application on the host laptop.

```
$> cd $HARPA/src/harpa_TCS_sensing_delivery_20140917__MODIFIED/
$> cd Sequential_version/Source
```

The above mentioned libraries of the DVFS driver that we just built should be located here. We can also locate all necessary files to build the graphical user interface (GUI) for the application. Once you are in this directory simply run:

```
$> make
```

Your application is now ready to run on the board. On your host laptop change to the directory of the board's file system, copy the entire harpa.git folder from your Desktop and change its permissions so that all users can read,write and execute files.

```
$> cd /tools/rootfs/home/user
$> sudo cp -R $HARPA ./
$> sudo chmod -R 777 harpa.git
```

We are now ready to connect to the board using the steps described earlier. Plug in the Ethernet cable, run the **sudo ifup** command, power up the board, wait a while for the SSH server to start and connect to the board using **ssh**. When you connect to the board change to the directory of the harpa application:

```
$> cd /home/user/harpa.git
```

We now need to source the source.me script located in the scripts/ folder. This script will deactivate the default DVFS driver of the Linux kernel and substitute it with our custom driver. Change to the directory of the script and run it:

```
$> cd scripts
$> source source.me
```

Once that is done, change to the directory of the application.

```
$> cd ../src/harpa_TCS_sensing_delivery_20140917__MODIFIED/
$> cd Sequential_version/Source
```

At this point we need to open a second terminal on the host laptop, navigate to our harpa.git folder on our Desktop, enter the directory of the application and build GUI and open the TCP connection that will allow transfer of data from the board to the laptop.

```
$> cd $HARPA/src/harpa_TCS_sensing_delivery_20140917__MODIFIED/
$> cd Sequential_version/Source
$> make gui_ref or $> make gui
```

The difference between **$> make gui_ref** and **$> make gui** is that the second commands expects to find a *signatures.txt* file in the workspace that contains the signature values we use as reference on our experiments. Therefore we should firstly run **$> make gui_ref** that builds the signatures.txt and later **$> make gui**. Follow the instructions provided on the terminal and run the application on the board.

We should note that establishing the TCP connection can present one particular problem. Although the makefiles identify the name of the host laptop and prompt you to use the same name when you run the application on the board, the board expects the laptop's name to correspond to 192.168.0.10 IP address. If by any chance the laptop does not recognise this particular IP address when referring to itself the board will not connect. This happens because the 192.168.0.10 address is hardcoded into the application's source code. The solution to this is adding the following line in the the /etc/hosts file of the host laptop.

```
192.168.0.10            <laptop's_name_recognised_by_the_application>
```

Please follow the official user guide of the application for further experimentation.[71]

Bibliography

[1] S. Borkar, “Designing reliable systems from unreliable components: the challenges of transistor variability and degradation,” *IEEE Micro*, vol. 25, pp. 10–16, Nov 2005.

[2] K. J. Kuhn, M. D. Giles, D. Becher, P. Kolar, A. Kornfeld, R. Kotlyar, S. T. Ma, A. Maheshwari, and S. Mudanai, “Process technology variation,” *IEEE Transactions on Electron Devices*, vol. 58, pp. 2197–2208, Aug 2011.

[3] F. Kriebel, M. Shafique, S. Rehman, J. Henkel, and S. Garg, “Variability and reliability awareness in the age of dark silicon,” *IEEE Design Test*, vol. 33, pp. 59–67, April 2016.

[4] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, “Dark silicon and the end of multicore scaling,” *IEEE Micro*, vol. 32, pp. 122–134, May 2012.

[5] D. P. Siewiorek and R. S. Swarz, *Reliable computer systems: design and evaluation*. MA, USA: A. K. Peters, Ltd, 3rd ed., 1998.

[6] “IMX6Q-SDB-Datasheet,” *www.freescale.com*, July 2015.

[7] L. Capodieci, “From optical proximity correction to lithography-driven physical design (1996-2006): 10 years of resolution enhancement technology and the roadmap enablers for the next decade,” 2006.

[8] A. Asenov, S. Kaya, and A. R. Brown, “Intrinsic parameter fluctuations in decananometer mosfets introduced by gate line edge roughness,” *IEEE Transactions on Electron Devices*, vol. 50, pp. 1254–1260, May 2003.

[9] X. Jiang, R. Wang, T. Yu, J. Chen, and R. Huang, “Investigations on line-edge roughness (ler) and line-width roughness (lwr) in nanoscale cmos technology: Part i – modeling and simulation method,” *IEEE Transactions on Electron Devices*, vol. 60, pp. 3669–3675, November 2013.

[10] M. Koh, W. Mizubayashi, K. Iwamoto, H. Murakami, T. Ono, M. Tsuno, T. Mihara, K. Shibahara, S. Miyazaki, and M. Hirose, “Limit of gate oxide thickness scaling in mosfets due to apparent threshold voltage fluctuation induced by tunnel leakage current,” *IEEE Transactions on Electron Devices*, vol. 48, pp. 259–264, Feb 2001.

[11] M. Miranda, “The threat of semiconductor variability.” IEEE SPECTRUM, June 2012.

[12] P. A. Stolk, F. P. Widdershoven, and D. B. M. Klaassen, “Modeling statistical dopant fluctuations in mos transistors,” *IEEE Transactions on Electron Devices*, vol. 45, pp. 1960–1971, Sep. 1998.

[13] D. Neamen, *An Introduction to Semiconductor Devices*. McGraw–Hill, 2006.

[14] K.-L. Chen, S. A. Saller, I. A. Groves, and D. B. Scott, “Reliability effects on mos transistors due to hot-carrier injection,” *IEEE Transactions on Electron Devices*, vol. 32, pp. 386–393, Feb 1985.

[15] D. Rodopoulos, P. Weckx, M. Noltsis, F. Catthoor, and D. Soudris, “Atomistic pseudo-transient bti simulation with inherent workload memory,” *IEEE Transactions on Device and Materials Reliability*, vol. 14, pp. 704–714, June 2014.

[16] D. Rodopoulos, S. B. Mahato, V. V. de Almeida Camargo, B. Kaczer, F. Catthoor, S. Cosemans, G. Groeseneken, A. Papanikolaou, and D. Soudris, “Time and workload dependent device variability in circuit simulations,” in *2011 IEEE International Conference on IC Design Technology*, pp. 1–4, May 2011.

[17] D. K. Schroder, “Negative bias temperature instability: What do we understand,” *Microelectronics Reliability*, vol. 47, no. 6, pp. 841–852, 2007.

[18] C. Nunes, P. F. Butzen, A. I. Reis, and R. P.Ribas, “Bti, hci and tddb aging impact in flip–flops,” *Microelectronics Reliability*, vol. 53, 2013.

[19] J. J. Clement, “Electromigration modeling for integrated circuit interconnect reliability analysis,” *IEEE Transactions on Device and Materials Reliability*, vol. 1, pp. 33–42, March 2001.

[20] S. Mukherjee, *Architecture Design for Soft Errors*. Morgan Kaufmann Publishers Inc. CA, USA, 1st ed., 2008.

[21] T. Grasser, B. Kaczer, W. Goes, H. Reisinger, T. Aichinger, P. Hehenberger, P. Wagner, F. Schanovsky, J. Franco, M. Toledano Luque, and M. Nelhiebel, “The paradigm shift in understanding the bias temperature instability: From reaction–diffusion to switching oxide traps,” *IEEE Transactions on Electron Devices*, vol. 58, pp. 3652–3666, Nov 2011.

[22] R. Wittmann, *NBTI Reliability Analysis, Reaction-Diffusion Model*. Miniaturization Problems in CMOS Technology: Investigation of Doping Profiles and Reliability.

[23] R. C. Baumann, “Radiation-induced soft errors in advanced semiconductor technologies,” *IEEE Transactions on Device and Materials Reliability*, vol. 5, pp. 305–316, Sep. 2005.

[24] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. USA: Addison-Wesley Publishing Company, 4th ed., 2010.

[25] P. Pillai and K. G. Shin, “Real-time dynamic voltage scaling for low-power embedded operating systems,” *SIGOPS Oper. Syst. Rev.*, vol. 35, pp. 89–102, Oct. 2001.

[26] AMD, “Cool ‘n’ quiet technology installation guide for amd athlon 64 processor based systems.” `https://web.archive.org/web/20070409045621/http://www.amd.com/us-en/assets/content_type/DownloadableAssets/Cool_N_Quiet_Installation_Guide3.pdf`, June 2004. White Paper.

[27] Intel, “Enhanced intel speedstep technology for the intel pentium m processor.” `https://web.archive.org/web/20150812030010/http://download.intel.com/design/network/papers/30117401.pdf`, March 2004. White Paper.

[28] D. Brodowski, R. J. Wysocki, and V. Kumar, “Linux cpufreq governors.” `https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt`, 2017. CPU frequency and voltage scaling code in the Linux(TM) kernel.

[29] G. Liu, M. Fan, and G. Quan, “Neighbor-aware dynamic thermal management for multi-core platform,” in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 187–192, March 2012.

[30] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, “Dynamic thermal management through task scheduling,” in *ISPASS 2008 - IEEE International Symposium on Performance Analysis of Systems and software*, pp. 191–201, April 2008.

[31] I. Yeo, C. C. Liu, and E. J. Kim, “Predictive dynamic thermal management for multicore systems,” in *2008 45th ACM/IEEE Design Automation Conference*, pp. 734–739, June 2008.

[32] I. Yeo, C. C. Liu, and E. J. Kim, “Predictive dynamic thermal management for multicore systems,” in *2008 45th ACM/IEEE Design Automation Conference*, pp. 734–739, June 2008.

[33] G. Liu, M. Fan, and G. Quan, “Neighbor-aware dynamic thermal management for multi-core platform,” in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 187–192, March 2012.

[34] J. Donald and M. Martonosi, “Techniques for multicore thermal management: Classification and new exploration,” in *33rd International Symposium on Computer Architecture (ISCA ’06)*, pp. 78–88, June 2006.

[35] L. Zhou and S. Guo, “Thermal management of arm socs using linux cpufreq as cooling device,” in *Computer Modelling and New Technologies*, pp. 162–167, 2015.

[36] E. Rotem, A. Naveh, M. Moffie, and A. Mendelson, “Analysis of thermal monitor features of intel pentium m processor,” 01 2004.

[37] H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, and J. Alvarez, “Thermal management system for high performance powerpc/sup tm/ microprocessors,” in *Proceedings IEEE COMPCON 97. Digest of Papers*, pp. 325–330, Feb 1997.

[38] “Thales communication and security. (2015) spectrum monitoring and homeland security.” `https://www.thalesgroup.com/en`.

[39] E. Wyrwas, L. Condra, and A. Hava, “Accurate quantitative physics-of-failure approach to integrated circuit reliability,” *IPC APEX EXPO Technical Conference 2011*, vol. 3, January 2011.

[40] M. Noltsis, P. Weckx, D. Rodopoulos, F. Cathoor, and D. Soudris, “Accuracy of quasi-monte carlo technique in failure probability estimations,” in *2016 International Conference on IC Design and Technology (ICICDT)*, pp. 1–4, June 2016.

[41] `www.vishay.com/docs/80116/80116.pdf`, February 2002. Vishay Semiconductors, Reliability.

[42] E. R. Hnatek, *Practical Reliability of Electronic Equipment and Products*. Marcel Dekker, Inc, 2003.

[43] G. Psychou, D. Rodopoulos, M. M. Sabry, T. Gemmeke, D. Atienza, T. G. Noll, and F. Catthoor, “Classification of resilience techniques against functional errors at higher abstraction layers of digital systems,” *ACM Comput. Surv.*, vol. 50, pp. 50:1–50:38, Oct. 2017.

[44] R. W. Hamming, “Error detecting and error correcting codes,” *The Bell System Technical Journal*, vol. 29, pp. 147–160, April 1950.

[45] T. J. Dysart and P. M. Kogge, “Reliability impact of n-modular redundancy in qca,” *IEEE Transactions on Nanotechnology*, vol. 10, pp. 1015–1022, Sep. 2011.

[46] G. S. Sohi, “Cache memory organization to enhance the yield of high performance vlsi processors,” *IEEE Transactions on Computers*, vol. 38, pp. 484–492, April 1989.

[47] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, “Razor: a low-power pipeline based on circuit-level timing speculation,” in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pp. 7–18, Dec 2003.

[48] N. J. Alewine, S.-K. Chen, W. K. Fuchs, and W. . W. Hwu, “Compiler-assisted multiple instruction rollback recovery using a read buffer,” *IEEE Transactions on Computers*, vol. 44, pp. 1096–1107, Sep. 1995.

[49] M. M. Sabry, D. Atienza, and F. Catthoor, "A hybrid hw-sw approach for intermittent error mitigation in streaming-based embedded systems," in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1110–1113, March 2012.

[50] T. Abdelzaher, J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback performance control in software services," *Control Systems, IEEE,*, 12 2003.

[51] C. Lu, J. A. Stankovic, S. Son, and G. Tao, "Feedback control real-time scheduling: Framework, modeling, and algorithms*," *Real-Time Systems*, vol. 23, pp. 85–126, 07 2002.

[52] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son, "Design and evaluation of a feedback control edf scheduling algorithm," in *Proceedings 20th IEEE Real-Time Systems Symposium (Cat. No.99CB37054)*, pp. 56–67, Dec 1999.

[53] K. Lampka and B. Forsberg, "Keep it slow and in time: Online dvfs with hard real-time workloads," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 385–390, March 2016.

[54] N. James, P. Restle, J. Friedrich, B. Huott, and B. McCredie, "Comparison of split-versus connected-core supplies in the power6 microprocessor," in *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, pp. 298–604, Feb 2007.

[55] V. J. Reddi, S. Kanev, W. Kim, S. Campanoni, M. D. Smith, G. Wei, and D. Brooks, "Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 77–88, Dec 2010.

[56] D. Bull, S. Das, K. Shivshankar, G. Dasika, K. Flautner, and D. Blaauw, "A power-efficient 32b arm isa processor using timing-error detection and correction for transient-error tolerance and adaptation to pvt variation," in *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*, pp. 284–285, Feb 2010.

[57] D. Rodopoulos, F. Catthoor, and D. Soudris, "Tackling performance variability due to ras mechanisms with pid-controlled dvfs," *IEEE Computer Architecture Letters*, vol. 14, pp. 156–159, July 2015.

[58] S. C. Andrea Acquaviva, Andrea Alimonda and M. Pittau, "Assessing task migration impact on embedded soft real-time streaming multimedia applications," *EURASIP Journal on Embedded Systems - Operating System Support for Embedded Real-Time Applications*, January 2008.

[59] K. Georgiou, S. Xavier-de-Souza, and K. Eder, "The iot energy challenge: A software perspective," *IEEE Embedded Systems Letters*, vol. 10, pp. 53–56, Sep. 2018.

[60] N. Zompakis, M. Noltsis, L. Ndreu, Z. Hadjilambrou, P. Englezakis, P. Nikolaou, A. Portero, S. Libutti, G. Massari, F. Sassi, A. Bacchini, C. Nicopoulos, Y. Sazeides, R. Vavrik, M. Golasowski, J. Sevcik, V. Vondrak, F. Catthoor, W. Fornaciari, and D. Soudris, "Harpa: Tackling physically induced performance variability," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pp. 97–102, March 2017.

[61] S. Corbetta, W. Meeus, D. Rodopoulos, E. Cappe, F. Catthoor, and A. Fritsch, "System-wide reliability analysis on real processor and application under vdd and t stress," in *SELSE– Silicon Errors in Logic –System Effects*, 2016.

[62] NXP, "Temperature monitor of imx6q board." `https://www.nxp.com/docs/en/application-note/AN5215.pdf`.

[63] "Homepage of Linux Target Image Builder," *http://ltib.org/*.

[64] “i.MX_6Dual6Quad_SABRE-SD_Linux_User’s_Guide,” *www.freescale.com*, 2013.

[65] “NFS Protocol,” *https://tools.ietf.org/html/rfc1094*.

[66] “Linux manual page for the xinetd service,” *https://www.systutorials.com/docs/linux/man/8-xinetd/*.

[67] “SSH Protocol description,” *https://www.ssh.com/ssh/server*.

[68] “Linux manual page for the exportfs command,” *https://linux.die.net/man/8/exportfs*.

[69] “PuTTY client,” *https://putty.org/*.

[70] “Linux manual page for the standard c libraries,” *http://man7.org/linux/man-pages/man7/libc.7.html*.

[71] S. Corbetta, D. Rodopoulos, and W. Meeus, *HARPA Experiments: User Guide*. 2015.