ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη περιβάλλοντος προγραμματισμού και παραγωγής κώδικα για επεξεργαστή DSP με το MATLAB

Κυριακή Ν. Κωνσταντινίδου

Μεταπτυχιακή Εργασία

Ηράκλειο, 2001

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη περιβάλλοντος προγραμματισμού και παραγωγής κώδικα για επεξεργαστή DSP με το MATLAB

Εργασία που υποβλήθηκε από την
ΚΥΡΙΑΚΗ Ν. ΚΩΝΣΤΑΝΤΙΝΙΔΟΥ
ως μερική εκπλήρωση των απαιτήσεων
για την απόκτηση
ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

Κυριακή Ν. Κωνσταντινίδου
Τμήμα Επιστήμης Υπολογιστών

Εισηγητική Επιτροπή:

Απόστολος Τραγανίτης
Αναπληρωτής Καθηγητής, Επόπτης

Γιώργος Τζιρίτας
Αναπληρωτής Καθηγητής, Μέλος

Πάνος Τραχανιάς
Αναπληρωτής Καθηγητής, Μέλος

Δεκτή:

Πάνος Κωνσταντόπουλος
Καθηγητής
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

Ηράκλειο, 2001

Ευχαριστίες

Στο σημείο αυτό, θα ήθελα να ευχαριστήσω όσους με τον ένα ή τον άλλο τρόπο βοήθησαν στην ολοκλήρωση της εργασίας αυτής.

Κατ'αρχήν, να ευχαριστήσω τον επόπτη καθηγητή μου, κ. Απόστολο Τραγανίτη, που, καθ'όλη τη διάρκεια της εργασίας μου, με ενθάρρυνε και με βοήθησε με τις υποδείξεις του.

Επίσης, θέλω να πω ένα μεγάλο ευχαριστώ στους γονείς μου, που, με υπομονή, με στήριξαν όλα αυτά τα χρόνια οικονομικά και ηθικά. Χωρίς τη δική τους βοήθεια η εργασία αυτή δύσκολα θα είχε ολοκληρωθεί.

Ακόμα, να ευχαριστήσω όλους τους φίλους μου, για την ηθική στήριξη που μου πρόσφεραν και ιδιαίτερα τον Γιάννη Σουρλαντζή, για την αμέριστη συμπαράσταση του αλλά και την ουσιαστική του βοήθεια σε κρίσιμα σημεία της εργασίας μου.

Τέλος, πρέπει να ευχαριστήσω το Τμήμα Επιστήμης Υπολογιστών του Πανεπιστημίου Κρήτης για την υλικοτεχνική υποστήριξη που μου παρείχε κατά τη διάρκεια των μεταπτυχιακών μου σπουδών.

Ανάπτυξη περιβάλλοντος προγραμματισμού και παραγωγής κώδικα για επεξεργαστή DSP με το MATLAB

Κυριακή Κωνσταντινίδου
Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Περίληψη

Οι εφαρμογές των επεξεργαστών ψηφιακού σήματος (digital signal processors, DSPs) αυξάνονται ραγδαία τα τελευταία χρόνια, και εξίσου αυξάνεται η πολυπλοκότητα τους. Η διαδικασία ανάπτυξης μιας νέας εφαρμογής περιλαμβάνει πολλές φάσεις, με πολλές και χρονοβόρες αλληλεπιδράσεις, με αποτέλεσμα να είναι εξαιρετικά μεγάλος ο χρόνος ανάπτυξης του τελικού προϊόντος. Έτσι, γίνεται έντονη η απαίτηση από την πλευρά των σχεδιαστών DSP συστημάτων για την ύπαρξη εργαλείων υψηλού επιπέδου, που θα επιταχύνουν και θα διευκολύνουν την παραπάνω διαδικασία.

Στην παρούσα εργασία, περιγράφεται η δημιουργία ενός εργαλείου υψηλού επιπέδου για τον προγραμματισμό του επεξεργαστή ψηφιακού σήματος TMS320C30 της Texas Instruments. Το εργαλείο αυτό, επιτρέπει την ενοποίηση σταδίων της διαδικασίας σχεδίασης και υλοποίησης συστημάτων μέσα από μια αυτοματοποιημένη διαδικασία παραγωγής κώδικα χαμηλού επιπέδου από σχηματικά διαγράμματα.

Συγκεκριμένα, επιτρέπει τη σχεδίαση και προσομοίωση ενός μοντέλου συστήματος επεξεργασίας σήματος με τη χρήση σχηματικών διαγραμμάτων του Simulink. Τα διαγράμματα αυτά μεταφράζονται σε κώδικα C, ο οποίος στη συνέχεια μεταγλωττίζεται σε γλώσσα χαμηλού επιπέδου για τον συγκεκριμένο επεξεργαστή. Ιδιαίτερη έμφαση έχει δοθεί στην ικανοποίηση προδιαγραφών από τον παραγόμενο κώδικα, που σχετίζονται με το μέγεθος του, την αξιοποίηση της μνήμης και την εκτέλεση του σε πραγματικό χρόνο.

Επόπτης: Απόστολος Τραγανίτης,
Αναπληρωτής καθηγητής,
Πανεπιστήμιο Κρήτης.

A code production development environment for DSPs using MATLAB

Kiriaki Konstantinidou
Master's Thesis

Computer Science Department
University of Crete, Greece

Abstract

Applications of Digital Signal Processors (DSPs) are rapidly increasing and so is their complexity. The development of a new application takes many steps that are interconnected which results in a non acceptable time-to-market. Developers of DSP systems ask for tools that can help them speed up the developing process.

In this thesis, we describe a tool for programming the TMS320C30 DSP by Texas Instruments. Using this tool we integrate development steps of design and implementation using an automated procedure to produce low level assembly language from a high level description of the algorithm.

In particular, we are able to design and simulate a digital signal processing model using block diagrams in Simulink, an industry standard design tool. The model is interpreted to C and then compiled to assembly for TMS320C30. Specifications about the size, memory and real time execution of the produced code are checked.

Advisor: Apostolos Traganitis
Associate Professor,
University of Crete.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

Οι εφαρμογές των επεξεργαστών ψηφιακού σήματος (digital signal processors, DSP) αυξάνονται ραγδαία τα τελευταία χρόνια, με αποτέλεσμα να γίνεται μεγάλη προσπάθεια από την πλευρά των εταιριών του χώρου για δημιουργία γρηγορότερων και αποτελεσματικότερων επεξεργαστών. Η ταχύτατη εξέλιξη της σχετικής τεχνολογίας έχει σαν αποτέλεσμα τη μείωση της διάρκειας ζωής των προϊόντων που σε συνδυασμό με τις έντονες συνθήκες ανταγωνισμού μεταξύ των εταιριών, οδηγεί σε μείωση του χρόνου ανάπτυξης ενός νέου προϊόντος (time-to-market). Σαν αποτέλεσμα, οι σχεδιαστές πολύ συχνά καλούνται να ξεκινήσουν την σχεδίαση και υλοποίηση του προϊόντος πριν ολοκληρωθούν οι προδιαγραφές του. Σε ένα τέτοιο περιβάλλον, οι αλλαγές σε τελευταία στάδια του κύκλου ανάπτυξης είναι πολύ συνηθισμένες ενώ είναι κρίσιμη η πραγματοποίηση τους σε στενά χρονικά περιθώρια.

Κάτι τέτοιο είναι πολύ δύσκολο να επιτευχθεί ακολουθώντας την παραδοσιακή διαδικασία ανάπτυξης ενός νέου συστήματος. Αυτή, περιλαμβάνει πολλές φάσεις (προσδιορισμός των απαιτήσεων και των προδιαγραφών, ανάπτυξη και αξιολόγηση του αλγορίθμου, ανάπτυξη του λογισμικού και/ή του υλικού, συνένωση των τμημάτων και έλεγχος) [1] και αντίστοιχα πολλές ομάδες εργασίας, με αρκετές αλληλεξαρτήσεις μεταξύ τους. Για το λόγο αυτό η ανάπτυξη του προϊόντος είναι χρονοβόρα αλλά και ‘‘δυσκίνητη’’ στις αλλαγές/βελτιώσεις.

Είναι φανερή, επομένως, η ανάγκη για αυτοματοποίηση της διαδικασίας, ώστε αυτή να επιταχυνθεί και να γίνει πιο ευέλικτη. Κάτι τέτοιο είναι εφικτό μόνο με τη χρήση ειδικού λογισμικού, που μέσα από ένα ολοκληρωμένο περιβάλλον θα υποστηρίζει την ενοποίηση των διαφόρων σταδίων της διαδικασίας, προσφέροντας λύσεις για κάθε ένα και δυνατότητες μετάβασης με αυτοματοποιημένο τρόπο στα επόμενα στάδια.

Προς το παρόν, δεν υπάρχουν πολλά εργαλεία που να υποστηρίζουν την παραπάνω

λειτουργικότητα. Οι περισσότερες προσπάθειες που έχουν γίνει εστιάζουν στην συνένωση των σταδίων της σχεδίασης του αλγορίθμου και της ανάπτυξης του λογισμικού, ενώ λιγότερες επεκτείνονται και στο στάδιο της ανάπτυξης του υλικού. Το περιβάλλον που παρέχεται, από αυτά τα εργαλεία, επιτρέπει συνήθως την επαναληπτική σχεδίαση και προσομοίωση (rapid prototyping) του συστήματος και στη συνέχεια την αυτόματη παραγωγή κώδικα σε γλώσσα μηχανής ή των προδιαγραφών του υλικού σε κατάλληλη γλώσσα περιγραφής (VHDL). Η μοντελοποίηση του συστήματος γίνεται χρησιμοποιώντας κάποια γλώσσα υψηλού επιπέδου, όπως C, C++, Ada ή σχηματικά διαγράμματα. Ειδικά τα σχηματικά διαγράμματα, καθώς θεωρούνται ένας πολύ φυσικός τρόπος έκφρασης αλγορίθμων ψηφιακής επεξεργασίας σήματος, κερδίζουν έδαφος τελευταία, έναντι των υπόλοιπων γλωσσών.

Ένα μειονέκτημα, που σχετίζεται με τη χρήση των εργαλείων αυτών για τον προγραμματισμό DSP επεξεργαστών, είναι ότι ο παραγόμενος κώδικας χαμηλού επιπέδου θεωρείται λιγότερο αποτελεσματικός από τον αντίστοιχο γραμμένο από προγραμματιστές. Η ανάγκη όμως για ταχύτερη ανάπτυξη εφαρμογών, η αύξηση της πολυπλοκότητας των ίδιων των εφαρμογών που δυσκολεύει το έργο του προγραμματιστή γλώσσας χαμηλού επιπέδου, καθώς και η αύξηση των διαθέσιμων πόρων των επεξεργαστών (σε μνήμη και σε συχνότητα ρολογιού) συνηγορούν τελικά υπέρ της χρήσης αυτών των εργαλείων.

Οριοθέτηση της εργασίας

Στην παρούσα εργασία, περιγράφεται η δημιουργία ενός εργαλείου υψηλού επιπέδου για τον προγραμματισμό του επεξεργαστή ψηφιακού σήματος TMS320C30 της Texas Instruments. Το εργαλείο αυτό, επιτρέπει την ενοποίηση των σταδίων της σχεδίασης και της υλοποίησης συστημάτων μέσα από μια αυτοματοποιημένη διαδικασία παραγωγής κώδικα χαμηλού επιπέδου από σχηματικά διαγράμματα.

Συγκεκριμένα, επιτρέπει τη σχεδίαση και προσομοίωση ενός μοντέλου συστήματος επεξεργασίας σήματος με τη χρήση σχηματικών διαγραμμάτων. Τα διαγράμματα αυτά μεταφράζονται σε κώδικα C, ο οποίος στη συνέχεια μεταγλωττίζεται σε γλώσσα χαμηλού επιπέδου για τον συγκεκριμένο επεξεργαστή. Τέλος, ο κώδικας του μοντέλου συνδέεται με κατάλληλο κώδικα ελέγχου της εκτέλεσής του για την παραγωγή του τελικού εκτελέσιμου. Ιδιαίτερη έμφαση έχει δοθεί στην ικανοποίηση προδιαγραφών από τον παραγόμενο κώδικα που σχετίζονται με το μέγεθος του, την αξιοποίηση της μνήμης και την εκτέλεση του σε πραγματικό χρόνο.

Για την υλοποίηση της συγκεκριμένης λειτουργικότητας συνδυάστηκαν διάφορα εργαλεία. Συγκεκριμένα για την σχεδίαση και προσομοίωση του συστήματος χρησιμοποιείται το εργαλείο Simulink από την εταιρία The Mathworks Inc. ενώ για την παραγωγή κώδικα χαμηλού επιπέδου από την C χρησιμοποιείται ο μεταγλωττιστής, που παρέχει η κατασκευάστρια εταιρία του επεξεργαστή. Για την παραγωγή κώδικα C από το σχηματικό διάγραμμα χρησιμοποιήθηκε το πλαίσιο παραγωγής κώδικα που

παρέχει το εργαλείο Real-Time Workshop της The Mathworks Inc.

Για περιβάλλον σχεδίασης και προσομοίωσης επιλέχθηκε το εργαλείο Simulink, γιατί είναι ένα από τα δημοφιλέστερα στον τομέα αυτό στην ερευνητική κοινότητα. Ταυτόχρονα, με την προσθήκη του εργαλείου Real-Time Workshop, προσφέρει ένα καλά καθορισμένο και ευέλικτο πλαίσιο για την παραγωγή κώδικα για μια νέα πλατφόρμα υλικού. Η πλατφόρμα αυτή επιλέχθηκε να είναι η πλακέτα αποτίμησης υλικού και λογισμικού EVM TMS320C30 της Texas Instruments, γιατί ήταν διαθέσιμη στο Τμήμα Επιστήμης Υπολογιστών. Ο επεξεργαστής στον οποίο βασίζεται η πλακέτα, ο TMS320C30, προσφέροντας καλή ισσοροπία μεταξύ κόστους και απόδοσης, υπήρξε για πολλά χρόνια εξαιρετικά δημοφιλής. Έτσι έχει χρησιμοποιηθεί ευρύτατα στην παραγωγή αλλά και σαν εκπαιδευτικό εργαλείο εισαγωγής στην τεχνολογία και τον προγραμματισμό επεξεργαστών ψηφιακού σήματος σε πανεπιστημιακά εργαστήρια και σε συγγράμματα [2, 3, 4].

Οργάνωση του κειμένου

Στο Κεφάλαιο 2, παρουσιάζεται η θεωρία στην οποία βασίζονται τα εργαλεία ανάπτυξης εφαρμογών επεξεργασίας σήματος. Αναφερόμαστε στη χρήση μοντέλων υπολογισμού ροής δεδομένων για την μοντελοποίηση συστημάτων και στις τεχνικές που έχουν αναπτυχθεί για σύνθεση λογισμικού (παράγραφος 2.1). Στη συνέχεια, παρουσιάζουμε τις δυσκολίες που σχετίζονται με τη μεταγλώττιση κώδικα για επεξεργαστές ψηφιακού σήματος καθώς και τεχνικές που έχουν προταθεί για την παραγωγή αποδοτικότερου κώδικα (2.2). Στο Κεφάλαιο 3, γίνεται μια σύντομη παρουσίαση του εργαλείου που παρέχει το περιβάλλον σχεδίασης και κυρίως των χαρακτηριστικών αυτού που είναι χρήσιμα στη σχεδίαση συστημάτων επεξεργασίας σήματος. Στο Κεφάλαιο 4, αναφερόμαστε στο πλαίσιο παραγωγής κώδικα, που παρέχεται από το Real-Time Workshop με έμφαση στα σημεία που απαιτήθηκε επέμβαση για την ανάπτυξη του εργαλείου μας. Στο Κεφάλαιο 5, περιγράφουμε την πλατφόρμα υλικού μας και στο Κεφάλαιο 6, η υλοποίηση μας. Συγκεκριμένα, παρουσιάζουμε τα διάφορα τμήματα κώδικα που χρειάστηκε να αναπτύξουμε και τον τρόπο που αυτά συνεργάζονται με το Simulink και το Real-Time Workshop για τη δημιουργία ενός ολοκληρωμένου περιβάλλοντος (6.1). Στη συνέχεια, αναφερόμαστε στις δυνατότητες και στους περιορισμούς που έχει το εργαλείο μας (6.2) και παρουσιάζουμε τη λειτουργία του μέσα από ορισμένα παραδείγματα (6.4). Τέλος, στο Κεφάλαιο 7, αναφέρουμε συμπεράσματα από τη χρήση του εργαλείου μας και πιθανές μελλοντικές επεκτάσεις.

Κεφάλαιο 2

Εργαλεία ανάπτυξης εφαρμογών επεξεργασίας σήματος

Όπως αναφέρθηκε στην εισαγωγή, η απαίτηση για αυτοματοποιημένα εργαλεία, που ενισχύουν την διαδικασία σχεδίασης και υλοποίησης ενός DSP συστήματος, είναι ισχυρή. Αρκετά τέτοια εργαλεία έχουν εμφανιστεί με παρόμοια λειτουργικότητα.

Αυτά, παρέχουν συνήθως τη δυνατότητα σχεδίασης του συστήματος χρησιμοποιώντας κάποια γλώσσα μοντελοποίησης και στη συνέχεια την αξιολόγηση του με τη βοήθεια προσομοίωσης. Ακολούθως, μπορεί να παραχθεί κώδικας χαμηλού επιπέδου για συγκεκριμένο επεξεργαστή ψηφιακού σήματος χρησιμοποιώντας δύο στάδια μεταγλώττισης.

Αυτά τα δύο στάδια μεταγλώττισης (*coarse-grain* και *fine-grain*) αναφέρονται ως *σύνθεση λογισμικού* και *παραγωγή κώδικα* αντίστοιχα (*software synthesis* και *code production*). Πιο συγκεκριμένα με τον όρο σύνθεση λογισμικού, εννοούμε την αυτόματη παραγωγή λογισμικού σε κάποια γλώσσα προγραμματισμού (π.χ. ένα πρόγραμμα σε C), από μια περιγραφή του συστήματος σε γλώσσα μοντελοποίησης. Από την άλλη, με τον όρο παραγωγή κώδικα εννοούμε την μεταγλώττιση του λογισμικού αυτού σε ισοδύναμο πρόγραμμα γραμμένο σε γλώσσα μηχανής για κάποιο συγκεκριμένο επεξεργαστή.

Οι τεχνολογίες μεταγλωττιστών σύνθεσης κώδικα και οι αντίστοιχες παραγωγής κώδικα, που εφαρμόζονται στα υπάρχοντα εργαλεία προγραμματισμού, δεν παρέχουν ακόμα πλήρως ικανοποιητικές υλοποιήσεις. Αρκετή ερευνητική δουλειά γίνεται προς αυτή την κατεύθυνση, διότι υπάρχουν ακόμα αρκετά προβλήματα που πρέπει να αντιμετωπιστούν.

Στο κεφάλαιο αυτό παρουσιάζονται οι δύο αυτές τεχνολογίες. Θα δούμε ζητήματα που αφορούν την αποδοτική σύνθεση και παραγωγή κώδικα, καθώς και τρόπους

αντιμετώπισης των προβλημάτων που σχετίζονται με αυτά. Επίσης θα δούμε πως οι δύο τεχνολογίες αλληλεπιδρούν και συνεισφέρουν στην ποιότητα του τελικού κώδικα. Στο τέλος του κεφαλαίου θα δούμε υπάρχοντα εργαλεία σχεδίασης και υλοποίησης συστημάτων DSP, που χρησιμοποιούν τις τεχνικές αυτές.

2.1 Σύνθεση λογισμικού από μοντέλα συστημάτων

Η αποδοτική σύνθεση λογισμικού εξαρτάται ισχυρά από τη γλώσσα μοντελοποίησης που χρησιμοποιείται. Ανάμεσα στις γλώσσες υψηλού επιπέδου (HLL), που έχουν χρησιμοποιηθεί κατά καιρούς για τη σχεδίαση του συστήματος, οι πιο δημοφιλείς είναι οι γλώσσες σχηματικών διαγραμμάτων. Τα σχετικά εργαλεία αποτελούν προέκταση του φυσικού τρόπου σκέψης των σχεδιαστών συστημάτων ψηφιακής επεξεργασίας σήματος, που έχουν συνηθίσει να σκέφτονται και να αποτυπώνουν στο χαρτί τα συστήματα τους με τη μορφή αφηρημένων γραφημάτων. Πέρα από το παραπάνω προφανές πλεονέκτημα τους, η περιγραφή ενός συστήματος με τη μορφή διασυνδεδεμένων υποσυστημάτων γνωστής λειτουργικότητας, ενισχύει την μεταφερσιμότητα (modularity) και την δυνατότητα επαναχρησιμοποίησης (reusability) τμημάτων του συστήματος. Επιπλέον, η υψηλού επιπέδου αναπαράσταση των στοιχείων του συστήματος, αυξάνει την αναγνωσιμότητα του συστήματος για μελλοντικές επεμβάσεις και μειώνει τον χρόνο ανάπτυξης έναντι των υπόλοιπων HLL.

Για να πραγματοποιήσουμε προσομοίωση, επαλήθευση ή παραγωγή κώδικα από ένα σχηματικό διάγραμμα, χρειαζόμαστε μια σημασιολογία που να ορίζει ακριβώς τις αλληλεπιδράσεις μεταξύ των block του διαγράμματος. Η σημασιολογία αυτή παρέχεται από το μοντέλο υπολογισμού *ροής δεδομένων* (*dataflow*) [5], στο οποίο βασίζονται σήμερα όλα τα εργαλεία σχεδίασης με χρήση σχηματικών διαγραμμάτων. Το μοντέλο ροής δεδομένων, που πρωτογενώς αναπτύχθηκε για την υλοποίηση παράλληλων υπολογισμών, δίνει πληροφορίες για τη δομή του συστήματος, η οποία στη συνέχεια μπορεί να αναλυθεί για να ενισχύσει την ποιότητα του παραγόμενου κώδικα.

Παρακάτω παρουσιάζεται θεωρία, σχετική με τη χρήση μοντέλων ροής δεδομένων υπολογισμού, τόσο στην μοντελοποίηση του συστήματος όσο και στην σύνθεση κώδικα.

2.1.1 Μοντελοποίηση συστημάτων με χρήση διαγραμμάτων ροής δεδομένων

Σύμφωνα με το μοντέλο υπολογισμών ροής δεδομένων, μια εφαρμογή είναι ένας κατευθυνόμενος γράφος, όπου οι κορυφές αναπαριστούν υπολογισμούς και οι ακμές λογικά κανάλια επικοινωνίας μεταξύ των υπολογισμών.

Για να είναι πρακτικά υλοποιήσιμο το μοντέλο ροής δεδομένων ενός συστήματος πρέπει να τηρεί προδιαγραφές *συνέπειας* (*consistency*). Σε ένα τέτοιο μοντέλο, ένας υπολογισμός μπορεί να εκτελεστεί όταν υπάρχουν επαρκή δεδομένα εισόδου. Όταν υπάρχουν λιγότερα δεδομένα εισόδου από όσα χρειάζονται (*buffer underflow*) ή περισσότερα από όσα ο υπολογισμός προλαβαίνει να αξιοποιήσει (*unbounded data*

accumulation) το μοντέλο δεν είναι συνεπές. Το μοντέλο ροής δεδομένων δεν εισάγει άλλους περιορισμούς στην σειρά εκτέλεσης των υπολογισμών εκτός από τους παραπάνω. Αντίθετα, σε μια επιτακτική (imperative) γλώσσα προγραμματισμού, (όπως η C), η σειρά των δηλώσεων αλλά και οι εντολές ελέγχου ροής του προγράμματος εισάγουν περισσότερους περιορισμούς από όσους απαιτεί η εξάρτηση των δεδομένων. Έτσι, τα μοντέλα ροής δεδομένων, επιτρέπουν στα εργαλεία σύνθεσης να έχουν πλήρη ευελιξία στην απόφαση της σειράς εκτέλεσης των υπολογισμών, ώστε αυτή να ικανοποιεί κριτήρια βελτιστοποίησης και περιορισμούς της υλοποίησης.

Μια από τις σημαντικότερες εργασίες που εκτελούνται στη σύνθεση κώδικα από σχηματικά διαγράμματα είναι αυτή του προσδιορισμού της σειράς εκτέλεσης των υπολογισμών (*scheduling*). Δύο βασικά σχήματα υπάρχουν, το *δυναμικό* και το *στατικό*. Στο στατικό, η σειρά εκτέλεσης των υπολογισμών αποφασίζεται στη φάση της σύνθεσης κώδικα και είναι σταθερή κατά την εκτέλεση, δηλαδή είναι ανεξάρτητη των δεδομένων. Ένα στατικό σχήμα, που είναι και *περιοδικό*, μπορεί να υλοποιηθεί χρησιμοποιώντας πεπερασμένη μνήμη ενώ ο κώδικας του προγράμματος εκτελείται σε μια συνεχή επανάληψη. Στο δυναμικό σχήμα, η σειρά εκτέλεσης των υπολογισμών μπορεί να αλλάζει κατά την εκτέλεση του μοντέλου. Απαιτείται επομένως ένας μηχανισμός απόφασης, που σε χρόνο εκτέλεσης θα αποφασίζει για την δυνατότητα εκτέλεσης ενός υπολογισμού με βάση την ύπαρξη ή όχι των απαιτούμενων δεδομένων εισόδου. Για την πλειοψηφία των DSP συστημάτων, όπου η συμπεριφορά των υποσυστημάτων είναι προβλεπόμενη, είναι επιθυμητή η αποφυγή του κόστους της δυναμικής απόφασης της σειράς εκτέλεσης.

Πολλές παραλαγές του μοντέλου υπολογισμών ροής δεδομένων, μπορούν να δημιουργηθούν ανάλογα με τον τρόπο που οι υπολογισμοί παράγουν και χρησιμοποιούν (consume) δεδομένα, τον σχήμα scheduling που χρησιμοποιείται κλπ. Η πιο δημοφιλής τέτοια παραλαγή, που χρησιμοποιείται σήμερα από τα περισσότερα διαθέσιμα εργαλεία σχεδίασης DSP συστημάτων, είναι το *synchronous dataflow* (SDF) [6]. Το μοντέλο αυτό ορίζει ότι το μέγεθος των δεδομένων εισόδου και εξόδου είναι σταθερό κατά τη διάρκεια της εκτέλεσης, η σειρά εκτέλεσης των υπολογισμών είναι περιοδική και στατική και η δέσμευση μνήμης γίνεται στατικά. Εξαιτίας αυτών των χαρακτηριστικών, το μοντέλο SDF διευκολύνει την επαλήθευση της συνέπειας του συστήματος. Παραδείγματα εργαλείων που χρησιμοποιούν μοντέλα υπολογισμών SDF είναι το *Simulink* της The MathWorks Inc. και το *Signal Processing Worksystem* της Cadence. Επίσης χρησιμοποιείται από σχετικά ερευνητικά εργαλεία όπως το Gabriel [7], το ASSIGN [8] και το Ptolemy [9].

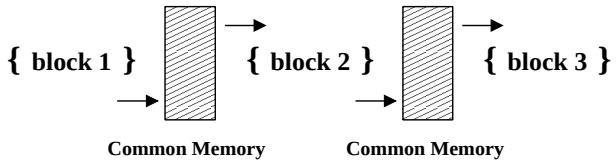
Επεκτάσεις του μοντέλου SDF χρησιμοποιούνται, επίσης, σε ευρέως διαδομένα εργαλεία σχεδίασης DSP συστημάτων. Αυτές είναι οι cyclo-static dataflow [10, 11], όπου το μέγεθος των δεδομένων που παράγει και χρησιμοποιεί ένας κόμβος υπολογισμού μπορεί να ποικίλει αρκεί η διαβάθμιση αυτή να επαναλαμβάνεται περιοδικά και scalable synchronous dataflow [12], που επιτρέπει τη δημιουργία *διανυσματικών* (*vectorized*) υλοποιήσεων, που αυξάνουν την απόδοση του κώδικα σε ορισμένες εφαρμογές. Με τον

όρο *διανυσματικότητα* (*vectorization*) εννοούμε την ομαδοποίηση πολλών κλήσεων του ίδιου κόμβου υπολογισμού ή διαδοχικών κόμβων για την μείωση του κόστους κλήσης (context-switch overhead). Το πρώτο μοντέλο χρησιμοποιείται στα εργαλεία *DSP Canvas* της Angeles Design Systems και *Synchro* της Eonic Systems ενώ το δεύτερο στο *COSSAP* της Synopsis.

Αρκετά ακόμα πειραματικά μοντέλα ροής δεδομένων έχουν προταθεί τα τελευταία χρόνια. Η εφαρμογή τους, όμως, σε πρακτικά συστήματα απαιτεί ακόμα αρκετή μελέτη. Κάποια από αυτά είναι τα multidimensional synchronous dataflow [13, 14], που επεκτείνει το μοντέλο SDF σε εφαρμογές με πολυδιάστατα σήματα όπως στην επεξεργασία εικόνας και βίντεο, well-behaved dataflow [15], που εισάγει κάποιες εξαρτώμενες από τα δεδομένα εντολές ελέγχου της ροής του προγράμματος και άρα απαιτεί δυναμική απόφαση της σειράς εκτέλεσης των υπολογισμών και boolean dataflow [16], που είναι ένα παράδειγμα μοντέλου, το οποίο δεν τηρεί τις προδιαγραφές συνέπειας, που αναφέρθηκαν παραπάνω.

2.1.2 Σύνθεση κώδικα από μοντέλα ροής δεδομένων

Η σύνθεση κώδικα από ένα SDF μοντέλο του συστήματος, γίνεται συνδέοντας τμήματα κώδικα από μια προϋπάρχουσα βιβλιοθήκη. Αυτά τα τμήματα κώδικα είναι γραμμένα στην γλώσσα που υποστηρίζει το εργαλείο (C, C++, Assembly κλπ). Η σύνθεση του κώδικα γίνεται συνήθως χρησιμοποιώντας την μέθοδο *threading*. Σύμφωνα με αυτήν, το εργαλείο ξεκινά κατασκευάζοντας μια περιοδική σειρά εκτέλεσης των υπολογισμών. Στη συνέχεια, διασχίζει τη σειρά αυτή και για κάθε κόμβο υπολογισμού που συναντάει, εισάγει τον αντίστοιχο κώδικα (*inline threading*) ή εισάγει μια κλήση σε μια υπορουτίνα που καλεί αυτόν τον κώδικα (*subprogram threading*). Στη φάση αυτή μπορεί να χρησιμοποιηθεί αποκλειστικά inline threading, subprogram threading ή ένας συνδυασμός τους (*hybrid threading*). Η ακολουθία κώδικα που παράγεται, περνάει στη συνέχεια από μια φάση επεξεργασίας, όπου εισάγονται οι απαραίτητες δηλώσεις της γλώσσας για την ανταλλαγή δεδομένων μεταξύ των κόμβων υπολογισμού (Σχήμα 2.1).



Σχήμα 2.1: Η μέθοδος σύνθεσης *threading*

Ο κώδικας για κάθε κόμβο υπολογισμού εισάγεται σειριακά και ανταλλάσει πληροφορία με τον προηγούμενο και τον επόμενο μέσω κοινής μνήμης

Ο καθορισμός της σειράς εκτέλεσης των υπολογισμών είναι μια πολύ κρίσιμη φάση της διαδικασίας σύνθεσης κώδικα. Οι αποφάσεις που λαμβάνονται εκεί έχουν μεγάλη επίδραση σε χαρακτηριστικά της τελικής υλοποίησης, όπως είναι οι απαιτήσεις

σε μνήμη δεδομένων και προγράμματος, η απόδοση και η ενεργειακή κατανάλωση. Έτσι υπάρχει ένα μεγάλο εύρος παραγόντων εξισσορόπησης (trade-offs) που πρέπει να ληφθούν υπόψη. Διάφορες τεχνικές έχουν αναπτυχθεί για τον χειρισμό αυτών των παραγόντων. Αυτές μπορούν να χωριστούν στις τεχνικές για ελαχιστοποίηση της χρήσης μνήμης (μειώνοντας είτε το μέγεθος του κώδικα είτε το μέγεθος των δεδομένων είτε και τα δύο), για βελτιστοποίηση της ροής δεδομένων (throughput) και για χρήση υβριδικού threading, δηλαδή βέλτιστη εισαγωγή κλήσεων συναρτήσεων σε επίπεδο (inlined) κώδικα[1]. Μια σύντομη παρουσίαση σχετικών τεχνικών μπορεί να βρεθεί στο [17].

Ο συνδυασμός μοντέλων ροής δεδομένων και επιτακτικών γλωσσών προγραμματισμού, έχει μεγάλο ενδιαφέρον για τους ερευνητικούς τομείς της σχεδίασης και υλοποίησης συστημάτων επεξεργασίας σήματος. Με αυτό το συνδυασμό, coarse-grain αλληλεπιδράσεις υποπρογραμμάτων περιγράφονται με το μοντέλο ροής δεδομένων ενώ η λειτουργικότητα των υποπρογραμμάτων περιγράφεται σε C, για παράδειγμα. Έτσι, οι τεχνικές σύνθεσης κώδικα χρησιμοποιούνται για να βελτιώσουν την τελική υλοποίηση σε επίπεδο επικοινωνίας υποπρογραμμάτων ενώ οι τεχνολογίες μεταγλωττιστών (που θα δούμε παρακάτω) εκτελούν fine-grain βελτιστοποιήσεις εντός των υποσυστημάτων.

2.2 Μεταγλώττιση κώδικα σε γλώσσα μηχανής

Το δεύτερο τμήμα του προγραμματιστικού εργαλείου, ασχολείται με την μεταγλώττιση λογισμικού, που έχει συντεθεί από ένα μοντέλο ροής δεδομένων του συστήματος. Η δυσκολία που σχετίζεται με αυτή τη φάση, όπως έχει καταδειχθεί από διάφορες πρακτικές μελέτες [18, 19], είναι η παραγωγή αποτελεσματικού κώδικα, που να συγκρίνεται με τον αντίστοιχο γραμμένο από προγραμματιστές. Η δυσκολία αυτή σχετίζεται από τη μία με την ιδιαίτερη αρχιτεκτονική των επεξεργαστών ψηφιακού σήματος και από την άλλη με τις τεχνικές παραγωγής κώδικα που χρησιμοποιούν οι μεταγλωττιστές.

2.2.1 Αρχιτεκτονική επεξεργαστών ψηφιακού σήματος

Προκειμένου να επιτύχουν τις υψηλές αποδόσεις που είναι απαραίτητες για την εκτέλεση εφαρμογών σε πραγματικό χρόνο, οι επεξεργαστές ψηφιακού σήματος έχουν εμπλουτισμένη αρχιτεκτονική έναντι των επεξεργαστών γενικού σκοπού [20, 21]. Η σκοπιμότητα αυτής της διαφοροποίησης, είναι η διευκόλυνση του επεξεργαστή στην εκτέλεση υπολογισμών, που είναι πολύ συνηθισμένοι στην επεξεργασία σήματος, όπως είναι οι υπολογισμοί φίλτρων και οι μετασχηματισμοί. Οι σημαντικότερες καινοτομίες είναι:

- Η ύπαρξη πολλαπλών αριθμητικών μονάδων, με ξεχωριστά register files, που μπορούν να δουλεύουν παράλληλα για την πραγματοποίηση ταχύτατων υπολογισμών. Για ένα μεταγλωττιστή είναι δύσκολο να τοποθετήσει τα δεδομένα με

[1]Επίπεδος είναι ένας κώδικας που δεν περιέχει καθόλου κλήσεις υπορουτινών

βέλτιστο τρόπο στους καταχωρητές ώστε να αξιοποιήσει στο μέγιστο τις παράλληλες λειτουργικές μονάδες. Πολλές φορές κάτι τέτοιο είναι δύσκολο ακόμα και για τον προγραμματιστή (π.χ. ο TMS320C6x της TI, διαθέτει 8 αριθμητικές και λογικές μονάδες).

- Η αύξηση του εύρους της μνήμης (memory bandwidth) χρησιμοποιώντας πολλαπλές παράλληλες τράπεζες μνήμης (έως και 6 έχουν εμφανιστεί) και αντίστοιχα πολλά datapaths για την προσπέλαση τους. Η προσέγγιση αυτή επιτρέπει την ταυτόχρονη προσπέλαση πολλαπλών θέσεων μνήμης σε ένα κύκλο ρολογιού, χρησιμοποιώντας απλές και φτηνές μνήμες. Έτσι για παράδειγμα, ο υπολογισμός ενός σημείου (tap) ενός φίλτρου FIR, που απαιτεί την ανάγνωση μιας εντολής και δύο δεδομένων από τη μνήμη μπορεί να πραγματοποιηθεί σε ένα κύκλο ρολογιού. Φυσικά, η αξιοποίηση της δυνατότητας αυτής μπορεί να γίνει μόνο με κατάλληλη κατανομή των δεδομένων στις τράπεζες μνήμης, πράγμα που συχνά απαιτεί την επέμβαση του προγραμματιστή.

- Η υποστήριξη περίπλοκων τρόπων διευθυνσιοδότησης των δεδομένων (addressing modes), όπως circular buffering και bit-reversed addresssing. Οι μεταγλωττιστές συνήθως δεν μπορούν να αξιοποιήσουν αυτές τις πολύ αποτελεσματικές μεθόδους διευθυνσιοδότησης.

2.2.2 Τεχνικές μεταγλώττισης

Οι πρώτοι μεταγλωττιστές, που εμφανίστηκαν για επεξεργαστές ψηφιακού σήματος, ακολουθούσαν τεχνικές μεταγλώττισης που αναπτύχθηκαν για RISC ή CISC επεξεργαστές γενικού σκοπού. Αν και οι τεχνικές αυτές είναι επαρκείς για τους επεξεργαστές γενικού σκοπού (η πτώση απόδοσης σε σύγκριση με τον αντίστοιχο κώδικα γραμμένο σε assembly δεν ξεπερνάει το 100%) είναι ανεπαρκείς για τους επεξεργαστές ψηφιακού σήματος, όπου η πτώση απόδοσης που παρατηρείται ξεπερνάει το 700% [18]. Σύμφωνα με μια άλλη μελέτη, η απόδοση των μεταγλωττιστών ψηφιακού σήματος είναι φτωχή όταν ο κώδικας δεν είναι προσαρμοσμένος στις ιδιαιτερότητες του επεξεργαστή και του μεταγλωττιστή [19].

Οι παραπάνω μετρήσεις αφορούν τη χρήση της γλώσσας C για τον προγραμματισμό DSP επεξεργαστών. Δεδομένου ότι η C είναι μια γλώσσα ‘‘γενικής χρήσεως’’, οι πρώτες προσπάθειες που έγιναν για βελτίωση της απόδοσης των μεταγλωττιστών αφορούσαν καταλληλότερες γλώσσες προγραμματισμού για DSP.

Νέες γλώσσες υψηλού επιπέδου προτάθηκαν, όπως η DFL [22] και η Silage. Παρά την αποτελεσματικότητα τους, οι γλώσσες αυτές δε συνάντησαν αποδοχή, κυρίως γιατί η φιλοσοφία τους απείχε από την αντίστοιχη γλωσσών όπως η C, με την οποία οι περισσότεροι προγραμματιστές είναι εξοικιωμένοι. Επιπλέον υποστηρίχθηκαν ανεπαρκώς από τους ίδιους τους κατασκευαστές εργαλείων για προγραμματισμό DSP.

Κάποιες άλλες προτάσεις, αφορούσαν την επέκταση του πρότυπου ANSI C, επιτρέ-

ποντας τη χρήση συγκεκριμένων ντιρεκτίβων για DSP, δηλαδή ειδικών δηλώσεων που αναγνωρίζονται μόνο από τον συγκεκριμένο μεταγλωττιστή [23, 24]. Αυτές οι προσθήκες επέτρεπαν στον μεταγλωττιστή να αξιοποιήσει κάποιες από τις ιδιαιτερότητες της αρχιτεκτονικής των DSP. Και αυτή η τακτική δεν έγινε ευρέως αποδεκτή καθώς είχε το μειονέκτημα της μη μεταφερσιμότητας.

Έτσι σήμερα, η πιο δημοφιλής γλώσσα για προγραμματισμό DSP παραμένει η C. Στα πλεονεκτήματα της περιλαμβάνεται η ύπαρξη πολλών υλοποιήσεων για συστήματα DSP (φίλτρα, μετασχηματισμοί κλπ).

Με αυτό το δεδομένο, άρχισαν να αναπτύσσονται διάφορες τεχνικές για βελτίωση του κώδικα που παράγεται από μεταγλωττιστές, και οι οποίες αξιοποιούσαν τις ιδιαιτερότητες της DSP αρχιτεκτονικής. Για κάθε ένα από τα χαρακτηριστικά που αναφέρθηκαν παραπάνω, έχουν προταθεί τεχνικές βελτιστοποίησης. Πιο συγκεκριμένα, στο [25] προτείνονται τεχνικές για την κατανομή των δεδομένων στις διαφορετικές τράπεζες μνήμης, για επεξεργαστές με δύο τράπεζες μνήμης, στο [26], τεχνικές για μείωση του μεγέθους κώδικα για επεξεργαστές VLIW (very long instruction word) ενώ στο [27], τεχνικές μείωσης του μεγέθους κώδικα λαμβάνοντας υπόψη και χρονικούς περιορισμους (εκτέλεση σε πραγματικό χρόνο).

Το βασικότερο πρόβλημα, όμως, που πρέπει να αντιμετωπίσουν οι νέοι μεταγλωττιστές είναι αυτό της *σύζευξης φάσεων* (*phase coupling*). Συγκεκριμένα, η παραδοσιακή διαδικασία μεταγλώττισης περιλαμβάνει φάσεις οι οποίες εκτελούνται σειριακά και ανεξάρτητα. Το γεγονός αυτό έχει επιπτώσεις στην ποιότητα του κώδικα, που δεν είναι αποδεκτές για έναν επεξεργαστή ψηφιακού σήματος. Το ζητούμενο είναι η αλληλοκάλυψη των φάσεων ώστε, να είναι δυνατή η ανταλλαγή πληροφοριών με στόχο το βέλτιστο δυνατό αποτέλεσμα. Στο [17], υπάρχει μια περίληψη σχετικών τεχνικών που έχουν αναπτυχθεί.

2.3 Εμπορικά διαθέσιμα περιβάλλοντα προγραμματισμού

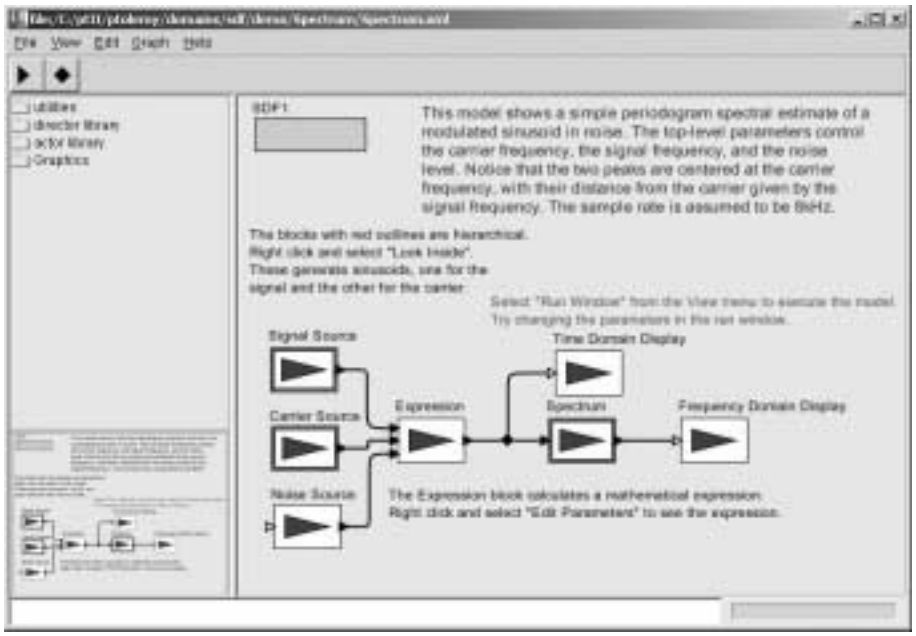
Στο κεφάλαιο αυτό θα αναφέρουμε ορισμένα παραδείγματα κατά κύριο λόγο εμπορικών προγραμματιστικών εργαλείων, που απευθύνονται στους σχεδιαστές συστημάτων επεξεργασίας σήματος. Για κάθε ένα, θα αναφέρουμε τις δυνατότητες του και τα βασικότερα χαρακτηριστικά του.

2.3.1 Ptolemy

Το εργαλείο αυτό διαφέρει από τα άλλα στο ότι αναπτύσεται από ένα πανεπιστημιακό ίδρυμα στα πλαίσια ερευνητικών δραστηριοτήτων στον τομέα. Συγκεκριμένα αναπτύσσεται από το Τμήμα Ηλεκτρολόγων Μηχανικών και Επιστήμης Υπολογιστών του Πανεπιστημίου της Καλιφόρνιας στο Berkeley. Στόχος του έργου είναι η μελέτη της μοντελοποίησης, προσομοίωσης και σχεδίασης ενσωματωμένων (embedded) συστημάτων πραγματικού χρόνου, που αποτελούνται από παράλληλα υποσυστήματα. Έμφαση

δίνεται στον τρόπο διασύνδεσης των υποσυστημάτων, που βασίζεται στη χρήση καλά καθορισμένων μοντέλων υπολογισμού.

Το εργαλείο Ptolemy II, υποστηρίζει διαφορετικά μοντέλα υπολογισμών, (τα οποία ονομάζει domains), μεταξύ των οποίων είναι και τα μοντέλα synchronous dataflow, process networks, finite state machines, discrete-event modeling, continuous-time modeling. Σε πειραματική φάση βρίσκεται η ενσωμάτωση αρκετών ακόμα μοντέλων. Για τη σχεδίαση του συστήματος παρέχεται ένας συντάκτης γραφικού περιβάλλοντος και μια βιβλιοθήκη από υποσυστήματα, που γενικά μπορούν να χρησιμοποιηθούν με διάφορα μοντέλα υπολογισμού. Στο [9] μπορεί να βρεθεί μια γενική παρουσίαση του εργαλείου με αναφορές για τα διάφορα μοντέλα υπολογισμών ενώ στο Σχήμα 2.2 φαίνεται μια άποψη του περιβάλλοντος σχεδίασης ενός συστήματος.

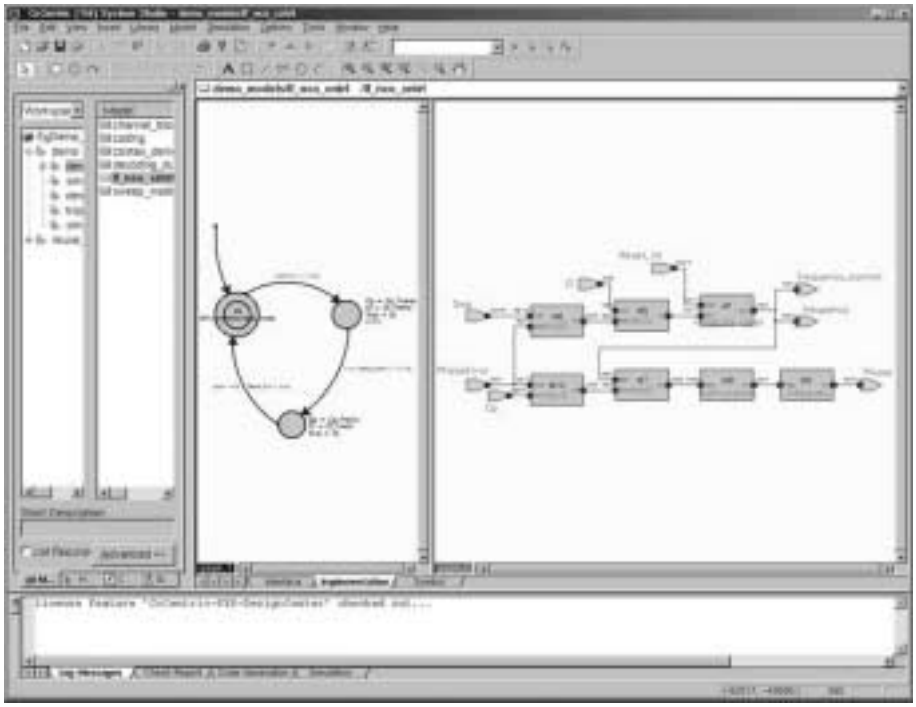


Σχήμα 2.2: Το περιβάλλον σχεδίασης του Ptolemy II, που αναπτύσσεται στο Πανεπιστήμιο της Καλιφόρνιας στο Berkeley

2.3.2 CoCentric System Studio

Το λογισμικό CoCentric System Studio της Synopsis, δίνει τη δυνατότητα σχεδίασης και προσομοίωσης ενός συστήματος ψηφιακής επεξεργασίας μέσα από ένα ολοκληρωμένο περιβάλλον (Σχήμα 2.3). Για την σχεδίαση του συστήματος μπορούν να χρησιμοποιηθούν μοντέλα από βιβλιοθήκες που διαθέτει το ίδιο το εργαλείο ή να δημιουργηθούν νέα σε C, C++ και SystemC. Για την υλοποίηση του συστήματος παρέχεται η δυνατότητα παραγωγής κώδικα C, SystemC (για υλοποίηση λογισμικού) και VHDL, Verilog (για υλοποίηση σε υλικό). Η σελίδα του προϊόντος στο διαδίκτυο μπορεί να βρεθεί στον

τόπο www.synopsis.com.



Σχήμα 2.3: Το περιβάλλον σχεδίασης και προσομοίωσης CoCentric System Studio της Synopsis

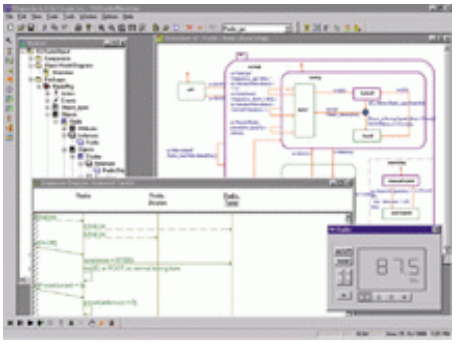
2.3.3 TI EVM C67x Developer's Kit

Το περιβάλλον προγραμματισμού TI EVM C67x Developer's Kit, δημιουργήθηκε από μια συνεργασία της Mathworks με την Texas Instruments. Παρέχει τη δυνατότητα παραγωγής κώδικα για την οικογένεια επεξεργαστών TMS320C67x της TI. Η λειτουργικότητα του εργαλείου αυτού και το πλαίσιο παραγωγής κώδικα που παρέχει είναι πολύ παρόμοια με τα αντίστοιχα του δικού μας εργαλείου. Η πρώτη παρουσίαση του, δε, έγινε πολύ πρόσφατα (Φεβρουάριος 2001) και αφού είχε ολοκληρωθεί η ανάπτυξη του προγραμματιστικού περιβάλλοντος που προτείνουμε στην παρούσα εργασία. Η σχεδίαση του συστήματος επεξεργασίας σήματος γίνεται στο Simulink ενώ για την παραγωγή κώδικα χρησιμοποιείται το Real-Time Workshop. Ο παραγόμενος κώδικας, μπορεί στη συνέχεια να ''φορτωθεί'' σε ένα προγραμματιστικό εργαλείο που παρέχεται από την TI, το Code Composer Studio (CCS). Μέσω του εργαλείου αυτού, είναι δυνατή η άμεση ανάλυση των αποτελεσμάτων που προκύπτουν από την εκτέλεση του συστήματος (π.χ. οπτικοποίηση). Επιπλέον, είναι δυνατή η αλλαγή παραμέτρων του συστήματος ενώ αυτό εκτελείται σε πραγματικό χρόνο, χρησιμοποιώντας σαν επιφάνεια διεπαφής το σχηματικό διάγραμμα του μοντέλου του συστήματος στο Simulink. Στο Σχήμα 2.4 βλέπουμε ένα παράδειγμα χρήσης του εργαλείου αυτού με το MATLAB.

Σχήμα 2.4: Το προγραμματιστικό περιβάλλον TI EVM C67x Developer's Kit

2.3.4 Rhapsody

Το λογισμικό Rhapsody (Σχήμα 2.5) της εταιρίας I-Logix παρέχει ένα περιβάλλον σχεδίασης συστημάτων πραγματικού χρόνου και παραγωγής κώδικα βασισμένο στη γλώσσα μοντελοποίησης UML (Unified Modeling Language).



Σχήμα 2.5: Το προγραμματιστικό περιβάλλον Rhapsody της I-Logix

Για τη σχεδίαση του συστήματος παρέχει πολλαπλά εργαλεία, όπως είναι το διάγραμμα αντικειμένων του μοντέλου (object model diagram), το διάγραμμα καταστάσεων (statechart), το διάγραμμα δραστηριότητας (activity diagram), το διάγραμμα συνεργασίας (collaboration diagram), το διάγραμμα ακολουθιών (sequence diagram) και άλλα. Η σελίδα του προϊόντος στο διαδίκτυο μπορεί να βρεθεί στον τόπο www.ilogix.com.

2.3.5 Motorola DSP Developer's Kit

Πρόκειται για μια συνεργασία της Mathworks με την Motorola για τη δημιουργία ενός περιβάλλοντος σχεδιασμού και ελέγχου DSP κώδικα για τις οικογένειες επεξεργαστών

56300 και 56600 με χρήση του Simulink και του Real-Time Workshop.

2.3.6 Math-Link EDS

Το λογισμικό Math-Link EDS της εταιρίας Mango DSP προσφέρει ένα ολοκληρωμένο περιβάλλον σχεδίασης, ελέγχου και παραγωγής κώδικα για μια πληθώρα πλατφόρμων υλικού και λογισμικού (μεταξύ αυτών και οι πλατφόρμες EVM C67x και C54x της Texas Instruments και τα Smart Core DSPs της DSPG) με χρήση του MATLAB και του Simulink.

Κεφάλαιο 3

Simulink

Το Simulink είναι ένα πακέτο λογισμικού, που επιτρέπει την μοντελοποίηση, προσομοίωση και ανάλυση δυναμικών συστημάτων, δηλαδή συστημάτων των οποίων οι είσοδοι, οι έξοδοι και οι καταστάσεις άλλάζουν με το χρόνο. Το Simulink μπορεί να χρησιμοποιηθεί για τη διερεύνηση της συμπεριφοράς ενός ευρέος φάσματος συστημάτων, που περιλαμβάνει ηλεκτρικά, μηχανικά και θερμοδυναμικά συστήματα.

Ο πυρήνας του Simulink, μπορεί να επεκταθεί με την προσθήκη βιβλιοθηκών (που ονομάζονται *blocksets*), όπως είναι η βιβλιοθήκη Ψηφιακής Επεξεργασίας Σήματος και η βιβλιοθήκη Τηλεπικοινωνιών, ή με την προσθήκη εργαλείων, όπως είναι το Real Time Workshop. Με την προσθήκη των βιβλιοθηκών, οι δυνατότητες προσομοίωσης του Simulink επεκτείνονται για να συμπεριλάβουν συστήματα τηλεπικοινωνιακά, επεξεργασίας σήματος κλπ. Με την προσθήκη εργαλείων, επεκτείνονται οι δυνατότητες του, πέρα από την προσομοίωση συστημάτων.

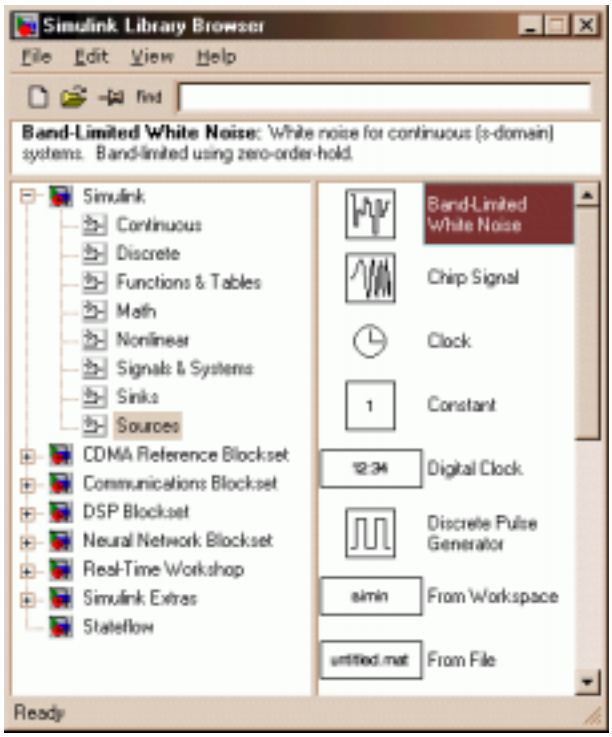
Η προσομοίωση ενός δυναμικού συστήματος είναι μια διαδικασία δύο βημάτων. Αρχικά, με τη χρήση ενός συντάκτη (editor) μοντέλων, δημιουργείται το ιεραρχικό μοντέλο του προς προσομοίωση συστήματος. Το μοντέλο αναπαριστά γραφικά τις εξαρτώμενες από το χρόνο μαθηματικές σχέσεις ανάμεσα στις εισόδους, καταστάσεις και εξόδους του συστήματος. Στη συνέχεια, προσομοιώνεται η συμπεριφορά του συστήματος για ένα καθορισμένο χρονικό διάστημα, χρησιμοποιώντας πληροφορία που έχει εισαχθεί στο μοντέλο.

Στις επόμενες παραγράφους, θα παρουσιάσουμε τις βασικές έννοιες, που είναι απαραίτητες για την μοντελοποίηση και προσομοίωση ενός συστήματος επεξεργασίας σήματος στο Simulink. Θα ξεκινήσουμε αναφέροντας τις βασικές δομικές μονάδες και τα χαρακτηριστικά ενός γενικού μοντέλου και θα περιγράψουμε τις λειτουργίες που εκτελούνται για την προσομοίωση του. Στη συνέχεια, θα εστιάσουμε σε διακριτά

συστήματα και θα δούμε ζητήματα που σχετίζονται με την ορθή προσομοίωση τους. Τέλος, θα αναφερθούμε στις δυνατότητες που παρέχονται για δημιουργία νέων blocks.

3.1 Μοντελοποίηση συστημάτων

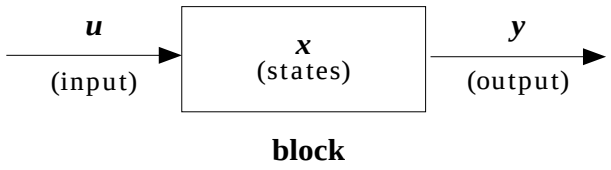
Ένα δυναμικό σύστημα παριστάνεται με τη βοήθεια ενός *σχηματικού διαγράμματος* (*block diagram*). Το διάγραμμα αυτό αποτελείται από ένα σύνολο συμβόλων, που ονομάζονται *block* και τα οποία διασυνδέονται με γραμμές. Τα blocks παριστάνουν στοιχειώδη δυναμικά συστήματα, που το Simulink γνωρίζει πως να προσομοιώσει, ενώ οι γραμμές αναπαριστούν συνδέσεις των εξόδων ενός block στις εισόδους ενός άλλου. Για τη δημιουργία του διαγράμματος παρέχονται βιβλιοθήκες από blocks (Σχήμα 3.1) και ένας συντάκτης γραφικού περιβάλλοντος (graphical editor), που επιτρέπει τη διασύνδεση των blocks με γραμμές. Ορισμένα από τα εργαλεία και οι δυνατότητες, που προσφέρει το Simulink για την μοντελοποίηση ενός συστήματος, περιγράφονται παρακάτω.



Σχήμα 3.1: Η βασική βιβλιοθήκη του Simulink

3.1.1 Blocks

Η στοιχειώδης δομική μονάδα ενός διαγράμματος, το block, αποτελείται γενικά από ένα σύνολο εισόδων, εξόδων και καταστάσεων (Σχήμα 3.2).



Σχήμα 3.2: Ένα block για το Simulink είναι ένα σύνολο εισόδων, καταστάσεων και εξόδων

Οι μεταξύ τους σχέσεις προσδιορίζονται από ένα σύνολο συναρτήσεων, που ονομάζονται *συναρτήσεις συστήματος* (*system functions*), και που είναι διαφορετικές για κάθε τύπο block. Οι συναρτήσεις αυτές περιλαμβάνουν:

- Μια *συνάρτηση εξόδου* (*output function*), που συσχετίζει τις εξόδους του block με τις εισόδους του, τις καταστάσεις του και τον χρόνο.
- Μια *συνάρτηση ενημέρωσης* (*update function*), που συσχετίζει τις μελλοντικές τιμές των διακριτών καταστάσεων του block με τις παρούσες τιμές των καταστάσεων και των εισόδων του.
- Μια *συνάρτηση παραγώγισης* (*derivative function*), που συσχετίζει τις παραγώγους των συνεχών καταστάσεων του block με τις παρούσες τιμές των καταστάσεων και των εισόδων του.

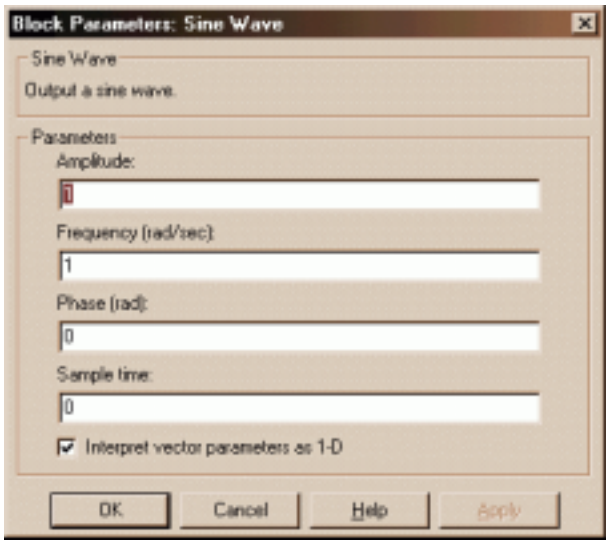
Ένα block περιγράφεται από μια ή περισσότερες από τις παραπάνω συναρτήσεις ανάλογα με τον τύπο του. Για παράδειγμα ένα block χωρίς καταστάσεις περιγράφεται μόνο από την συνάρτηση εξόδου. Οι συναρτήσεις αυτές είναι η μόνη πληροφορία που χρειάζεται το Simulink για να εκτελέσει το block κατά τη διάρκεια της προσομοίωσης.

Παράμετροι των block

Τα περισσότερα blocks δέχονται ως παράμετρο βασικές ιδιότητες τους. Για παράδειγμα η γεννήτρια ημιτονοειδών σημάτων δέχεται ως παράμετρο το πλάτος, τη συχνότητα και τη φάση του σήματος (Σχήμα 3.3). Κάθε παραμετρικό block διαθέτει ένα παράθυρο διαλόγου (dialog box) που επιτρέπει την εισαγωγή παραμέτρων κατά τη δημιουργία ή την προσομοίωση του μοντέλου.

Σε ένα διάγραμμα είναι δυνατό να υπάρχουν πολλαπλά στιγμιότυπα[1] (instances) του ίδιου τύπου block. Σε αυτή την περίπτωση το κάθε στιγμιότυπο μπορεί να έχει διαφορετικές τιμές παραμέτρων. Για παράδειγμα, μπορεί να υπάρχουν δύο στιγμιότυπα της γεννήτριας ημιτονοειδών σημάτων, όπου το ένα παράγει ένα ημίτονο συχνότητας f_1 ενώ το άλλο ένα συνημίτονο συχνότητας f_2.

[1] Εδώ ο όρος στιγμιότυπο χρησιμοποιείται με την έννοια που του δίνουν οι αντικειμενοστραφείς γλώσσες προγραμματισμού.



Σχήμα 3.3: Ενδεικτικό παράθυρο διαλόγου για την εισαγωγή παραμέτρων σε ένα block

Καταστάσεις και μέθοδοι επίλυσης

Η *κατάσταση* (*state*) ενός block είναι ένα σύνολο μεταβλητών που χρησιμοποιούνται στον υπολογισμό της εξόδου του block και που η τρέχουσα τιμή τους είναι συνάρτηση των προηγούμενων καταστάσεων ή/και εισόδων. Ένα block που η έξοδος του εξαρτάται μόνο από την τρέχουσα είσοδο δεν έχει καταστάσεις. Όταν ένα block έχει καταστάσεις πρέπει να κρατάει τις τιμές των προηγούμενων καταστάσεων του ή/και των εισόδων του για να υπολογίσει την τρέχουσα κατάσταση.

Ένα block μπορεί να έχει διακριτές και συνεχείς καταστάσεις. Όταν έχει συνεχείς καταστάσεις αποθηκεύει μόνο τις παραγώγους τους και όχι τις πραγματικές τους τιμές. Έτσι κατά τη διάρκεια της προσομοίωσης οι πραγματικές τιμές των καταστάσεων πρέπει να υπολογίζονται ολοκληρώνοντας αριθμητικά τις παραγώγους τους. Το Simulink παρέχει υλοποιήσεις διαφόρων μεθόδων αριθμητικής ολοκλήρωσης, που ονομάζει μεθόδους επίλυσης συνήθων διαφορικών εξισώσεων (ordinary differential equation (ODE) solvers). Κατά την προσομοίωση του συστήματος πρέπει να επιλεγεί η κατάλληλη *μέθοδος επίλυσης* (*solver*).

Οι μέθοδοι επίλυσης κατατάσσονται σε δύο κατηγορίες. Στις μεθόδους *σταθερού βήματος* (*fixed-step*) και στις μεθόδους *μεταβλητού βήματος* (*variable-step*). Οι δύο αυτές μέθοδοι διαφέρουν στον τρόπο υπολογισμού του βήματος της προσομοίωσης. Η πρώτη, το διατηρεί σταθερό σε όλη τη διάρκεια της προσομοίωσης, ενώ η δεύτερη, το προσαρμόζει δυναμικά ανάλογα με τις ανάγκες του μοντέλου. Περισσότερα γι'αυτό θα δούμε στην παράγραφο 3.3.2.

3.1.2 Σήματα και τύποι δεδομένων

Σε ένα μοντέλο Simulink, ο όρος *σήμα* (*signal*) αναφέρεται στις εξόδους των blocks. Είναι δυνατό να οριστούν διάφορα χαρακτηριστικά ενός σήματος, όπως το όνομα, ο τύπος αποθήκευσης (8-bit, 16-bit, 32-bit ακέραιος κλπ), ο αριθμητικός τύπος (πραγματικός ή μιγαδικός), η διάσταση κλπ. Πολλά blocks δέχονται ως είσοδο ή παράγουν σήματα οποιουδήποτε τύπου ή διάστασης και προσαρμόζονται αυτόματα, ενώ άλλα εισάγουν περιορισμούς στα χαρακτηριστικά των σημάτων εισόδου και εξόδου. Ο ορισμός κατάλληλων χαρακτηριστικών στα σήματα ενός μοντέλου αφενός επιτρέπει την όσο το δυνατόν πιο πιστή περιγραφή του πραγματικού συστήματος και αφετέρου μπορεί σε πολλές περιπτώσεις (ιδιαίτερα σε μεγάλα μοντέλα) να βελτιώσει σημαντικά τον χρόνο εκτέλεσης της προσομοίωσης.

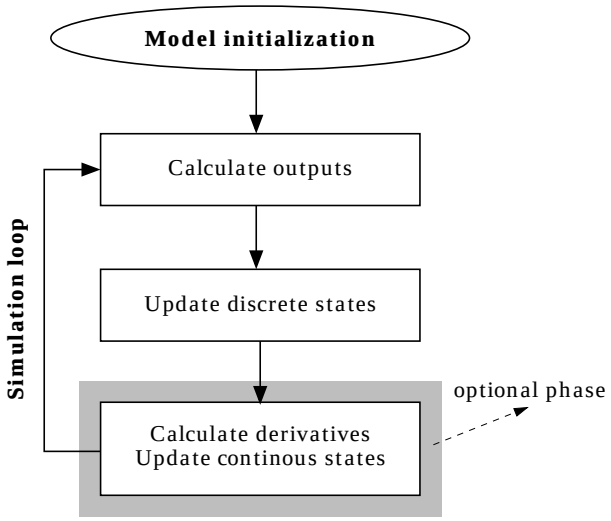
3.2 Προσομοίωση συστημάτων

Η προσομοίωση ενός συστήματος αναφέρεται στη διαδικασία του υπολογισμού των καταστάσεων και των εξόδων ενός συστήματος για ένα διάστημα του χρόνου, χρησιμοποιώντας πληροφορία, που παρέχεται από το μοντέλο του συστήματος. Η προσομοίωση ενός συστήματος περιλαμβάνει τη φάση της αρχικοποίησης και τη φάση της εκτέλεσης.

Στη φάση της αρχικοποίησης ελέγχονται οι παράμετροι του συστήματος, τίθενται οι τιμές παραμέτρων που δεν έχουν οριστεί ρητά από το χρήστη και αποφασίζεται η σειρά εκτέλεσης των blocks, που μπορεί να είναι γενικά διαφορετική από την σειρά με την οποία είναι τοποθετημένα στο μοντέλο Επειδή η τελευταία απόφαση είναι σημαντική για την ορθή εκτέλεση της προσομοίωσης, ακολουθείται αλγόριθμος που λαμβάνει υπόψη του τις αλληλεξαρτήσεις μεταξύ των blocks του μοντέλου.

Κατά την εκτέλεση ενός μοντέλου, το Simulink εκτελεί ένα βρόγχο (loop), κάθε επανάληψη του οποίου αναφέρεται ως *βήμα της προσομοίωσης* (*simulation step*). Όπως ήδη αναφέρθηκε, το βήμα της προσομοίωσης εξαρτάται από τη μέθοδο επίλυσης που χρησιμοποιείται. Είναι φανερό, ότι η επιλογή του βήματος επηρεάζει τόσο την ακρίβεια της προσομοίωσης όσο και το χρόνο εκτέλεσης της.

Σε κάθε βήμα της προσομοίωσης εκτελούνται τα blocks του μοντέλου με βάση τη σειρά που καθορίστηκε στην φάση της αρχικοποίησης. Για να προσομοιώσει ένα block, το Simulink καλεί τις συναρτήσεις συστήματός του με τη σειρά που φαίνεται στο Σχήμα 3.4. Αρχικά εκτελείται η συνάρτηση εξόδου, που ανανεώνει την έξοδο του block. Στην συνέχεια καλείται η συνάρτηση ενημέρωσης, που ανανεώνει τις διακριτές καταστάσεις του block και τέλος καλείται η συνάρτηση παραγώγισης για να υπολογίσει τις παραγώγους των νέων καταστάσεων του. Η διαδικασία αυτή συνεχίζεται μέχρι το τέλος του χρόνου της προσομοίωσης. Στο Σχήμα φαίνονται οι ενέργειες που εκτελούνται σε κάθε βήμα της προσομοίωσης, για ένα γενικό μοντέλο με συνεχείς και διακριτές καταστάσεις.



Σχήμα 3.4: Ο βρόγχος της προσομοίωσης για κάθε block

3.3 Διακριτά συστήματα στο Simulink

Όπως ήδη αναφέρθηκε, το Simulink επιτρέπει την προσομοίωση διακριτών συστημάτων. Τα μοντέλα διακριτών συστημάτων κατασκευάζονται κατά κανόνα χρησιμοποιώντας διακριτά blocks, αν και μπορεί κάποια μοντέλα να είναι υβριδικά (hybrid), δηλαδή να περιέχουν ένα συνδιασμό διακριτών και συνεχών blocks.

Ένα *διακριτό* (*discrete*) block διαφέρει από ένα συνεχές στο ότι αντιδρά στις αλλαγές της εισόδου μόνο σε ακέραια πολλαπλάσια ενός καθορισμένου χρονικού διαστήματος που ονομάζεται *χρόνος δειγματοληψίας* ή *περίοδος δειγματοληψίας* (*sample time* ή *sample period*) του block. Οι χρονικές στιγμές, στις οποίες αλλάζει η έξοδος του block, ονομάζονται *στιγμές δειγματοληψίας* (*sample time hits*).

Οι τιμές που παίρνει το σήμα που παράγεται από ένα διακριτό block (διακριτό σήμα από εδώ και στο εξής), μεταξύ των στιγμών δειγματοληψίας, εξαρτώνται από τις παραμέτρους της προσομοίωσης, όπως θα δούμε παρακάτω. Κάθε διακριτό block περιλαμβάνει μια παράμετρο που επιτρέπει τον καθορισμό του χρόνου δειγματοληψίας του. Αν δεν υπάρχει αυτή η παράμετρος, το block κληρονομεί τον χρόνο δειγματοληψίας από το σύστημα, σύμφωνα με σχετικό αλγόριθμο.

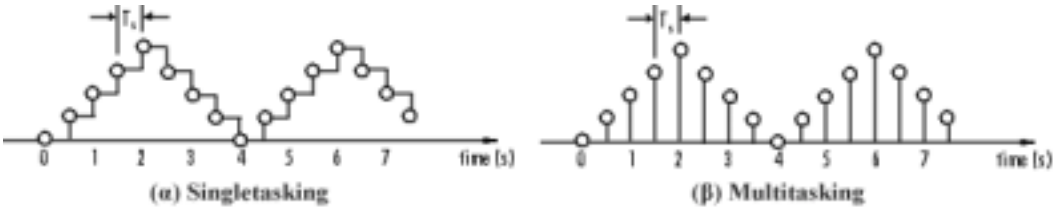
Σε ένα μοντέλο διακριτού συστήματος, μπορεί όλα τα blocks να έχουν τον ίδιο χρόνο δειγματοληψίας, οπότε το μοντέλο ονομάζεται *singlerate*, ή όχι, οπότε το μοντέλο ονομάζεται *multirate*.

3.3.1 Παράμετροι προσομοίωσης

Τα διακριτά blocks δεν έχουν συνεχείς καταστάσεις και έτσι δεν χρειάζονται τη συνάρτηση παραγώγισης. Κατά συνέπεια, για την προσομοίωση ενός αμιγώς διακριτού

συστήματος (ενός συστήματος, δηλαδή, που δεν περιέχει συνεχή στοιχεία), μπορεί να χρησιμοποιηθεί οποιαδήποτε από τις μεθόδους επίλυσης, χωρίς να υπάρχει διαφορά στο αποτέλεσμα. Όμως, για καλύτερη προσέγγιση των διακριτών χαρακτηριστικών του συστήματος είναι προτιμότερη η χρήση μιας διακριτής μεθόδου επίλυσης είτε από την κατηγορία των μεθόδων σταθερού βήματος είτε από την κατηγορία των μεθόδων μεταβλητού βήματος.

Η μέθοδος σταθερού βήματος προσφέρει άλλες δύο επιλογές για την προσομοίωση του συστήματος. Την επιλογή *singletasking* και την επιλογή *multitasking*. Η διαφορά τους εστιάζεται στον τρόπο που ορίζεται ένα διακριτό σήμα, στα χρονικά διαστήματα μεταξύ διαδοχικών στιγμών δειγματοληψίας. Συγκεκριμένα, η singletasking επιλογή θεωρεί ότι το σήμα διατηρεί την τιμή της αμέσως προηγούμενης δειγματοληψίας (sample-and-hold), ενώ η multitasking επιλογή δεν το ορίζει καθόλου. Η διαφορά αυτή φαίνεται καλύτερα στο Σχήμα 3.5.



Σχήμα 3.5: Ένα διακριτό σήμα στο Simulink

(α) Με την επιλογή προσομοίωσης singletasking το σήμα διατηρεί την τιμή της τελευταίας δειγματοληψίας στο διάστημα μιας περιόδου, (β) με την επιλογή multitasking είναι μη ορισμένο για το ίδιο διάστημα

Η επιλογή αυτή έχει συνέπειες στα multirate μοντέλα, όπου συχνά χρειάζεται να γίνουν πράξεις μεταξύ δύο σημάτων με διαφορετικό χρόνο δειγματοληψίας. Στην περίπτωση αυτή η επιλογή multitasking, δεν επιτρέπει τις πράξεις (γιατί το ένα από τα δύο σήματα δεν ορίζεται σε όλα τα σημεία που ορίζεται το άλλο) αν δεν μεσολαβήσει κάποιο block μετατροπής του χρόνου δειγματοληψίας (π.χ. zero-order-hold, unit-delay κλπ).

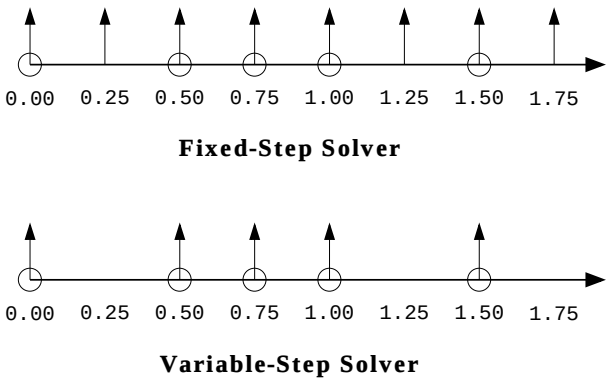
Εάν δεν τεθεί ρητά από τον χρήστη η μια από τις δύο επιλογές, το Simulink χρησιμοποιεί για τα singlerate μοντέλα τη μέθοδο singletasking ενώ για τα multirate μοντέλα την μέθοδο multitasking. Η επιλογή multirate-singletasking μπορεί να γίνει για μοντέλα, όπου δεν απαιτείται ρητή μετατροπή των χρόνων δειγματοληψίας των blocks και είναι η προτεινόμενη επιλογή για μοντέλα διακριτών συστημάτων. Η επιλογή singlerate-multitasking δεν επιτρέπεται.

Οι δύο αυτές επιλογές δεν επηρεάζουν μόνο τον τρόπο εκτέλεσης της προσομοίωσης αλλά, μέσω του κώδικα που παράγεται για το μοντέλο από το Real-Time Workshop, επηρεάζουν και τον τρόπο εκτέλεσης του μοντέλου σε πραγματικό χρόνο. Στο επόμενο κεφάλαιο, θα δούμε περισσότερα για αυτό το θέμα, καθώς και για τυχόν αλλαγές που πρέπει να γίνουν στον μοντέλο για την ορθή εκτέλεσή του.

3.3.2 Το βήμα της προσομοίωσης στα διακριτά συστήματα

Το βήμα της προσομοίωσης στη μοντελοποίηση ενός διακριτού συστήματος εξαρτάται από τη μέθοδο επίλυσης που χρησιμοποιείται και από τον *θεμελιώδη χρόνο δειγματοληψίας* (*fundamental sample time*) του συστήματος. Ο θεμελιώδης χρόνος δειγματοληψίας ορίζεται ως ο μέγιστος διαιρέτης των χρόνων δειγματοληψίας του συστήματος που δίνει ακέραιο πηλίκο. Για παράδειγμα, ο θεμελιώδης χρόνος δειγματοληψίας ενός multirate συστήματος, που έχει χρόνους δειγματοληψίας 0,25 και 0,5 δευτερόλεπτα, είναι 0,25 δευτερόλεπτα. Από την άλλη, αν οι χρόνοι δειγματοληψίας είναι 0,5 και 0,75 δευτερόλεπτα, τότε ο θεμελιώδης χρονος δειγματοληψίας του συστήματος είναι πάλι τα 0,25 δευτερόλεπτα. Τέλος, σε singlerate μοντέλο, ταυτίζεται με τον μοναδικό χρόνο δειγματοληψίας του συστήματος.

Το βήμα της προσομοίωσης επηρεάζεται από τον τύπο της μεθόδου επίλυσης ως εξής: Η μέθοδος σταθερού βήματος το θέτει ίσο με τον θεμελιώδη χρόνο δειγματοληψίας του συστήματος ενώ η μέθοδος μεταβλητού βήματος το μεταβάλει ώστε να είναι κάθε φορά ίσο με τη απόσταση μεταξύ δύο διαδοχικών στιγμών δειγματοληψίας. Αυτές οι δύο προσεγγίσεις διαφοροποιούνται μόνο στην περίπτωση multirate συστημάτων όπου ο θεμελιώδης χρόνος δειγματοληψίας είναι μικρότερος από όλους τους χρόνους δειγματοληψίας του συστήματος, όπως στη δεύτερη περίπτωση του προηγούμενου παραδείγματος. Στο Σχήμα 3.6 φαίνεται η διαφορά των δύο προσεγγίσεων για το ίδιο παράδειγμα. Τα βέλη αντιστοιχούν σε βήματα της προσομοίωσης ενώ οι κύκλοι αντιστοιχούν σε στιγμές δειγματοληψίας.

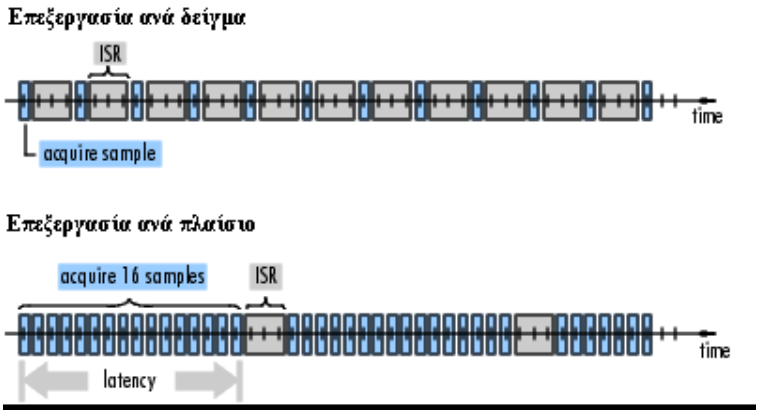


Σχήμα 3.6: Το βήμα της προσομοίωσης

Είναι ξεκάθαρο από το παραπάνω σχήμα, ότι η μέθοδος μεταβλητού βήματος απαιτεί λιγότερα βήματα προσομοίωσης για την προσομοίωση του συστήματος. Από την άλλη, η μέθοδος σταθερού βήματος απαιτεί λιγότερη μνήμη και είναι γρηγορότερη αν ένας από τους χρόνους δειγματοληψίας του συστήματος είναι θεμελιώδης.

3.3.3 Επεξεργασία δειγμάτων ανά πλαίσιο

Η ομαδοποίηση δειγμάτων σε πλαίσια για την επεξεργασία τους, είναι ένα φαινόμενο πολύ συνηθισμένο σε πραγματικά συστήματα, γιατί επιτρέπει την κατανομή του δεδομένου φόρτου επεξεργασίας ενός δείγματος σε πολλά αυξάνοντας έτσι την απόδοση του συστήματος. Ένα πλεονέκτημα της ανά πλαίσιο επεξεργασίας είναι ότι αυξάνει τον ρυθμό ανάγνωσης δειγμάτων (sample throughput) ενώ ένα μειονέκτημα ότι εισάγει μια αρχική καθυστέρηση ίση με το χρόνο ανάγνωσης του πρώτου πλαισίου. Ανάλογα όμως με το μέγεθος του πλαισίου και τον τύπο της εφαρμογής η καθυστέρηση αυτή μπορεί να είναι αποδεκτή. Τα δύο αυτά χαρακτηριστικά της ανά πλαίσιο επεξεργασίας φαίνονται στο Σχήμα 3.7.



Σχήμα 3.7: Επεξεργασία σήματος ανά δείγμα και ανά πλαίσιο

Το Simulink επιτρέπει τόσο την ανά δείγμα επεξεργασία (sample-based processing) όσο και την ανά πλαίσιο επεξεργασία (frame-baced processing) για την καλύτερη προσέγγιση ενός πραγματικού συστήματος.

3.4 Δημιουργία νέων blocks

Το Simulink επιτρέπει τη δημιουργία νέων blocks από τον χρήστη με δύο τρόπους: είτε χρησιμοποιώντας τον συντάκτη γραφικού περιβάλλοντος είτε χρησιμοποιώντας προγραμματισμό.

Σύμφωνα με τον πρώτο και απλούστερο τρόπο, δημιουργείται ένα σχηματικό διάγραμμα που προσομοιώνει την συμπεριφορά του block και το οποίο οργανώνεται σαν υποσύστημα χρησιμοποιώντας τη σχετική δυνατότητα του Simulink.

Σύμφωνα με τον δεύτερο τρόπο, χρησιμοποιείται κάποια κοινή γλώσσα προγραμματισμού για την περιγραφή του block. Όπως αναφέρθηκε νωρίτερα το Simulink γνωρίζει πως να προσομοιώσει ένα block αν διαθέτει τις συναρτήσεις συστήματος του block. Για τη συνεργασία του block αυτού με το υπόλοιπο μοντέλο πρέπει επίσης να γνωρίζει

τον αριθμό εισόδων και εξόδων του block, τον τύπο τους και τη διάσταση τους, αν το block είναι διακριτό ή συνεχές και στην περίπτωση που είναι διακριτό το ρυθμό δειγματοληψίας του. Όλες αυτές οι πληροφορίες (και πιθανόν άλλες για πολυπλοκότερα blocks) παρέχονται στο Simulink χρησιμοποιώντας ένα καλά καθορισμένο API (Application Interface). Το API αυτό ονομάζεται S-function API, και η περιγραφή του block που προκύπτει με αυτό τον τρόπο, *S-function* (από το S-ystem functions = συναρτήσεις συστήματος). Περισσότερα για τις συμβάσεις του API και τις λειτουργίες που πρέπει να υλοποιεί μια S-function υπάρχουν στο [30].

Αφού δημιουργηθεί η περιγραφή αυτή, μεταγλωττίζεται στη μορφή δυναμικά συνδεόμενης βιβλιοθήκης (dynamically linked library) με τη βοήθεια ενός εργαλείου που παρέχεται και συνδέεται δυναμικά με το Simulink όταν χρειάζεται. Στο μοντέλο μπορεί να ενσωματωθεί χρησιμοποιώντας το ομώνυμο block της βασικής βιβλιοθήκης του Simulink. Οι γλώσσες που υποστηρίζονται περιλαμβάνουν τις C, C++, Fortran, Ada και MATLAB.

Παρόλο που η δεύτερη μέθοδος είναι δυσκολότερη στην υλοποίηση, προσφέρει κάποια πλεονεκτήματα έναντι της πρώτης και έτσι χρησιμοποιείται ευρύτατα. Είναι χαρακτηριστικό ότι όλα τα βασικά blocks της βιβλιοθήκης ‘‘Ψηφιακής Επεξεργασίας Σήματος’’ είναι γραμμένα σαν C S-functions. Οι χρήσεις των S-functions περιλαμβάνουν:

- Τη δημιουργία block, που αναπαριστούν οδηγούς συσκευών. Ένα τέτοιο block είναι αδύνατο να δημιουργηθεί με άλλη μέθοδο. Περισσότερα για αυτό το θέμα θα αναφέρουμε στο επόμενο κεφάλαιο.

- Την ενσωμάτωση δεδομένου κώδικα σε ένα μοντέλο. Πολλές φορές διαθέτουμε υλοποιημένο σε κάποια γλώσσα προγραμματισμού έναν αλγόριθμο που μας ενδιαφέρει να ενσωματώσουμε στο μοντέλο μας. Αντί να τον υλοποιήσουμε ξανά χρησιμοποιώντας blocks, μπορούμε να τον προσαρμόσουμε ώστε να ακολουθεί το S-function API.

- Την περιγραφή ενός συστήματος ως ένα σύνολο μαθηματικών εξισώσεων.

Στα νέα blocks, που δημιουργούνται είτε με τον ένα είτε με τον άλλο τρόπο, μπορεί προαιρετικά να προστεθεί κάποιο παράθυρο διαλόγου για εισαγωγή παραμέτρων ενώ επιπλέον μπορούν να εισαχθούν σε μια custom βιβλιοθήκη για ευκολότερη πρόσβαση.

Κεφάλαιο 4

Παραγωγή κώδικα από μοντέλα Simulink

Για την παραγωγή κώδικα από ένα μοντέλο Simulink, παρέχεται το εργαλείο Real-Time Workshop (RTW). Το RTW παρέχει τη δυνατότητα για σύνθεση κώδικα σε γλώσσα υψηλού επιπέδου (C ή Ada) και στη συνέχεια για μεταγλώττιση του σε αυτόνομη εφαρμογή πραγματικού χρόνου για κάποια *πλατφόρμα εκτέλεσης* (*target platform*) υλικού ή λογισμικού (π.χ. λειτουργικό σύστημα ή πλακέτα υλικού).

Νέες πλατφόρμες εκτέλεσης είναι δυνατό να προστεθούν χάρη στην ανοιχτή αρχιτεκτονική του εργαλείου. Βασικό χαρακτηριστικό αυτής είναι η τμηματοποίηση (segmentation) του κώδικα σε τρεις κατηγορίες (κώδικας σχετικός με το μοντέλο, κώδικας σχετικός με την πλατφόρμα εκτέλεσης και κώδικας ανεξάρτητος τόσο του μοντέλου όσο και της πλατφόρμας) και ο σαφής καθορισμός του τρόπου που αυτές αλληλεπιδρούν.

Παρακάτω θα αναλύσουμε αυτή την τμηματοποίηση. Θα δούμε τις βασικές λειτουργίες που πρέπει να εκτελεί το κάθε τμήμα κώδικα, καθώς και πως αυτά αλληλεπιδρούν μεταξύ τους για την ανταλλαγή πληροφοριών (δομές δεδομένων, μεταβλητές κλπ). Επίσης, θα αναφερθούμε στον τρόπο που γίνεται η σύνθεση κώδικα και θα δούμε περιληπτικά τη διαδικασία που ακολουθείται για την παραγωγή του τελικού εκτελέσιμου. Θα ασχοληθούμε με ζητήματα που σχετίζονται με τις απαιτήσεις του μοντέλου σε μνήμη και με την ορθή εκτέλεση του σε πραγματικό χρόνο. Τέλος, θα δούμε λίγο πιο συγκεκριμένα τα συστατικά μιας νέας πλατφόρμας εκτέλεσης και οδηγίες για την παροχή τους.

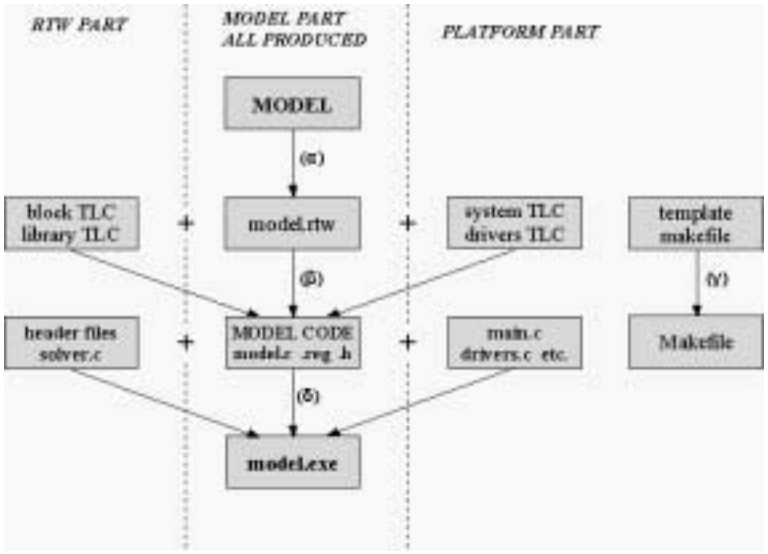
Στα παρακάτω με τον όρο *χρήστης*, εννοούμε αυτόν που χρησιμοποιεί το εργαλείο RTW για να δημιουργήσει μια εφαρμογή για μια πλατφόρμα που υποστηρίζεται ήδη από το πακέτο. Με τον όρο *προγραμματιστής*, εννοούμε αυτόν που αναπτύσσει μια νέα πλατφόρμα εκτέλεσης. Τέλος, όπου αναφέρεται ο όρος *εφαρμογή* εννοείται εφαρμογή

πραγματικού χρόνου.

4.1 Παραγωγή κώδικα

Η παραγωγή κώδικα από ένα μοντέλο Simulink ακολουθεί τα στάδια που είδαμε νωρίτερα στο Κεφάλαιο 2.

Ξεκινάει με τη φάση της ανάλυσης (Σχήμα 4.1(α)), στην οποία καθορίζεται η σειρά εκτέλεσης των blocks, που σύμφωνα με την ορολογία του Κεφαλαίου 2, είναι στατική και περιοδική. Δηλαδή, δεν αλλάζει κατά την εκτέλεση του μοντέλου και επαναλαμβάνεται περιοδικά. Σ'αυτή τη φάση, υπολογίζεται, επίσης, η μνήμη που είναι απαραίτητη για την εκτέλεση του συστήματος. Η πληροφορία αυτή, μαζί με τα υπόλοιπα χαρακτηριστικά του μοντέλου, αποθηκεύεται σε μια ενδιάμεση αναπαράσταση του συστήματος, που χρησιμοποιείται σε όλα τα επόμενα στάδια της διαδικασίας παραγωγής κώδικα.



Σχήμα 4.1: Τα στάδια της διαδικασίας παραγωγής κώδικα

(α) Ανάλυση του μοντέλου και παραγωγή μιας ενδιάμεσης αναπαράστασης σε γλώσσα ανεξάρτητη της τελικής, (β) σύνθεση κώδικα από τον Target Language Compiler (γ) δημιουργία του προσαρμοσμένου Makefile και (δ) δημιουργία του εκτελέσιμου

Ακολουθεί η φάση της σύνθεσης κώδικα (Σχήμα 4.1(β)), στην οποία συμμετέχει ο Target Language Compiler (TLC), ένα εργαλείο που συνοδεύει το RTW. Η γλώσσα Target Language είναι μια μεταφραζόμενη (interpreted) γλώσσα προγραμματισμού, που έχει σχεδιαστεί με αποκλειστικό σκοπό τη χρήση της στην σύνθεση κώδικα.

Στη φάση αυτή χρησιμοποιείται, εκτός από την ενδιάμεση αναπαράσταση του μοντέλου, μια πληθώρα άλλων αρχείων, που ονομάζονται target files και που σε γενικές γραμμές περιέχουν οδηγίες για τη μετάφραση του κάθε block αλλά και ολόκληρου του συστήματος στη γλώσσα υψηλού επιπέδου της επιλογής μας. Τα αρχεία αυτά κατατάσσονται στις εξής κατηγορίες:

- *System target file.* Το αρχείο αυτό είναι το σημείο έναρξης της μετάφρασης και είναι διαφορετικό για κάθε πλατφόρμα εκτέλεσης. Στο αρχείο αυτό ορίζονται οι γενικές παράμετροι της παραγωγής κώδικα, όπως είναι η γλώσσα (C ή Ada) και ο τύπος του παραγόμενου κώδικα (παράγραφος 4.1.1). Επίσης, εδώ ορίζονται οι σχετικές με την πλατφόρμα εκτέλεσης επιλογές που παρέχονται στο χρήστη, μέσα από το μενού επιλογών του Real-Time Workshop. Περισσότερα για τα περιεχόμενα αυτού του αρχείου θα δούμε στην παράγραφο 6.1.2.

- *Block target files.* Κάθε block του Simulink έχει ένα αντίστοιχο target file, που προσδιορίζει πως θα μεταφραστεί το συγκεκριμένο block σε τελικό κώδικα. Το αρχείο περιέχει δηλώσεις σε γλώσσα TLC, που κατευθύνουν την μετάφραση, και δηλώσεις σε γλώσα υψηλού επιπέδου, που αποτελούν τον κώδικα εκτέλεσης του block. Κάθε block target file είναι πλήρως παραμετρικό και υποστηρίζει όλες τις πιθανές παραμέτρους του block και της προσομοίωσης. Ένα απλό παράδειγμα τέτοιου αρχείου για το block *Quantizer* του Simulink, μπορεί να βρεθεί στο Παράρτημα Α.

- *Βιβλιοθήκη συναρτήσεων TLC*, που παρέχουν βοηθητικές λειτουργίες και καλούνται από τα υπόλοιπα target files κατά τη διάρκεια της μετάφρασης.

Η παρουσίαση του ακριβή τρόπου που γίνεται η σύνθεση κώδικα, δεν εμπίπτει στους σκοπούς αυτής της εργασίας. Αναλυτικές πληροφορίες μπορεί κανείς να βρει στα [31, 32]. Αφού ολοκληρωθεί η σύνθεση κώδικα, απομένει η μεταγλώττιση του (compiling) και η σύνδεση (linking) των διαφόρων τμημάτων καθώς και άλλων απαραίτητων βιβλιοθηκών μεταξύ τους (Σχήμα 4.1(δ)). Αυτό γίνεται με τη βοήθεια ενός Makefile, το οποίο παράγεται από ένα πρότυπο (template) Makefile (Σχήμα 4.1(γ)). Αυτό είναι απαραίτητο, προκειμένου να λαμβάνονται υπόψη οι επιλογές του χρήστη αλλά και να μεταγλωττίζονται και να συνδέονται κάθε φορά μόνο τα χρησιμοποιούμενα από το μοντέλο block και βιβλιοθήκες.

Το εκτελέσιμο, που παράγεται με αυτό τον τρόπο, μπορεί να εκτελεστεί άμεσα, χωρίς άλλη επέμβαση από την πλευρά του χρήστη, στην πλατφόρμα υλικού ή λογισμικού. Σε περίπτωση που παρέχεται σχετική επιλογή, αυτό είναι δυνατό να συμβεί αυτόματα. Εναλλακτικά, είναι δυνατό να ‘‘φορτωθεί’’ το εκτελέσιμο σε κάποιον εκσφαλματωτή (debugger) ή σε κάποιο περιβάλλον προγραμματισμού χαμηλού επιπέδου.

4.1.1 Τύποι παραγόμενου κώδικα

Η παραγωγή κώδικα, μπορεί να γίνει και με τους δύο τρόπους που είδαμε στο Κεφάλαιο 2 (subprogram threading και inlined threading). Το RTW συσχετίζει τους τρόπους αυτούς με δύο τυποποιήσεις παραγωγής κώδικα που ονομάζει *RealTime code format* και *Embedded code format* αντίστοιχα. Επειδή, οι δύο τυποποιήσεις χαρακτηρίζουν συνολικά τον τρόπο παραγωγής κώδικα, ο προγραμματιστής μιας νέας πλατφόρμας εκτέλεσης πρέπει εξ’αρχής να επιλέξει τη μια από τις δύο.

Ο βασικός κανόνας είναι ότι η πρώτη τυποποίηση είναι καταλληλότερη για την παραγωγή εφαρμογών rapid-prototyping ενώ η δεύτερη χρησιμοποιείται κυρίως για την παραγωγή τελικού κώδικα (production code). Μια διαφορά τους, που ενισχύει την παραπάνω διάκριση, βρίσκεται στη δομή δεδομένων που χρησιμοποιείται για την περιγραφή των βασικών χαρακτηριστικών του μοντέλου. Στην τυποποίηση RealTime η δομή αυτή περιέχει πεδία για όλες τις πιθανές ιδιότητες που μπορεί να έχει ένα μοντέλο με αποτέλεσμα να απαιτεί μεγάλο χώρο μνήμης για την αποθήκευση της. Από την άλλη, στην τυποποίηση Embedded περιέχει μόνο τα απαραίτητα πεδία.

Μια συνέπεια από τη χρήση inlined threading στην τυποποίηση Embedded, είναι η αδυναμία ενσωμάτωσης block που είναι γραμμένα σαν S-functions, αν δεν υπάρχει μια TLC αναπαράσταση τους (το *block target file* που αναφέρθηκε παραπάνω). Σε περίπτωση που υπάρχει, ο κώδικας που περιγράφεται στο TLC αρχείο ενσωματώνεται στον κώδικα του μοντέλου κανονικά. Το γεγονός ότι, δεν έχουν όλα τα S-function blocks, αναπαράσταση σε γλώσσα TLC, δημιουργεί περιορισμούς στα μοντέλα που μπορούν να χρησιμοποιηθούν με αυτή την τυποποίηση. Από την άλλη η τυποποίηση Embedded είναι μονόδρομος για πλατφόρμες επεξεργασίας σήματος με περιορισμένους πόρους, γι' αυτό και τελικά χρησιμοποιήθηκε.

4.2 Η δομή του κώδικα

Για τη δημιουργία της εκτελέσιμης εφαρμογής συνεισφέρουν τρία τμήματα κώδικα. Αυτά είναι (Σχήμα 4.2):

- Τα εξαρτώμενα από το μοντέλο τμήματα, τα οποία αναφέρονται και ως τμήματα της εφαρμογής.
- Τα εξαρτώμενα από την πλατφόρμα εκτέλεσης τμήματα.
- Τμήματα ανεξάρτητα τόσο του μοντέλου όσο και της πλατφόρμας εκτέλεσης (όπως οι αλγόριθμοι επίλυσης συνήθων διαφορικών εξισώσεων κλπ).

Τα εξαρτώμενα από το μοντέλο τμήματα παράγονται κατά τη διαδικασία της σύνθεσης, ενώ τα υπόλοιπα παραμένουν σταθερά και παρέχουν αυτό που ονομάζουμε *run-time interface* της εφαρμογής, δηλαδή το *πλαίσιο εκτέλεσής* της. Αυτό περιλαμβάνει ότι είναι απαραίτητο για την εκτέλεση της εφαρμογής στη συγκεκριμένη πλατφόρμα υλικού (π.χ. αρχικοποίηση υλικού, χειρισμός περιφερειακών συσκευών, χειρισμός διακοπών υλικού) και σε πραγματικό χρόνο (π.χ. ο χρονισμός της εκτέλεσης). Οι αλληλεξαρτήσεις των τμημάτων αυτών καθορίζονται σε κοινά αρχεία επικεφαλίδας (header files), γεγονός που επιτρέπει την ανεξάρτητη ανάπτυξη τους. Στις επόμενες παραγράφους θα δούμε αναλυτικότερα ποιες λειτουργίες εκτελεί το κάθε τμήμα κώδικα.

Σχήμα 4.2: Η γενική δομή του κώδικα που συνεισφέρει στη δημιουργία της εφαρμογής

4.2.1 Κώδικας εξαρτώμενος από την πλατφόρμα

Σε αυτό το τμήμα του κώδικα ανήκει η βασική συνάρτηση του προγράμματος (*main* function), που πρέπει να εκτελεί τις εξής λειτουργίες:

- Ενεργοποίηση και αρχικοποίηση των μονάδων υλικού που είναι απαραίτητα για την εκτέλεση του μοντέλου (παραδείγματα τέτοιων μονάδων είναι τα χρονόμετρα, οι σειριακές θύρες, τα κυκλώματα αναλογικής εισόδου και εξόδου).
- Εγκατάσταση ρουτινών χειρισμού των διακοπών υλικού (interrupt service routines, ISR) για εκτέλεση του μοντέλου σε πραγματικό χρόνο.
- Αρχικοποίηση της μνήμης που χρησιμοποιείται από το μοντέλο τόσο των δομών που αποθηκεύουν παραμέτρους (π.χ. χρόνοι δειγματοληψίας) όσο και των πινάκων δεδομένων.
- Έλεγχος για λάθη ή για συνθήκες τερματισμού του μοντέλου.

Μια εφαρμογή, που παράγεται με το RTW, μπορεί να είναι μια απλή προσομοίωση του μοντέλου (δηλαδή να εκτελείται σε μη πραγματικό χρόνο) ή να εκτελείται σε πραγματικό χρόνο. Το δεύτερο μπορεί να γίνει είτε με τη χρήση διακοπών υλικού (interrupts) είτε με τη χρήση ειδικών εντολών ενός λειτουργικού συστήματος πραγματικού χρόνου. Για την εκτέλεση της εφαρμογής σε πραγματικό χρόνο σε έναν επεξεργαστή ψηφιακού σήματος απαιτείται η χρήση διακοπών υλικού. Σ'αυτή την περίπτωση η συνάρτηση main() πρέπει να ενεργοποιήσει τις διακοπές υλικού και να εισέλθει σε έναν ατέρμονο βρόγχο, από όπου διακόπτεται είτε για την εκτέλεση του κώδικα του μοντέλου είτε για τερματισμό. Στο Σχήμα 4.3 φαίνεται σε ψευδοκώδικα η συνάρτηση main(). Εκτός από

```
main()
{
    Initialization
    Initialize and start hardware
    Enable interupts
    While (not Error)
        Background task
    EndWhile
    Disable interrupts
    Complete any background tasks
    Shutdown
}
```

Σχήμα 4.3: Η συνάρτηση main() σε ψευδοκώδικα

την βασική συνάρτηση, σε αυτή την κατηγορία εντάσσονται όλες οι ρουτίνες που σχετίζονται με τον χειρισμό του υλικού (π.χ. ρουτίνες για ανάγνωση από και εγγραφή σε συσκευές εισόδου/εξόδου και ρουτίνες για χειρισμό περιφερειακών συσκευών), καθώς και οδηγοί συσκευής για χρήση στην προσομοίωση και την εκτέλεση του μοντέλου (βλ. παράγραφο 4.4.1).

4.2.2 Κώδικας κοινός για όλα τα μοντέλα και πλατφόρμες

Ο κώδικας αυτός περιλαμβάνει τους αλγόριθμους εκτέλεσης του μοντέλου, που είναι παραμετρικοί ως προς τον τύπο του μοντέλου (singlerate, multirate, multitasking). Στο Σχήμα 4.4, φαίνονται σε ψευδοκώδικα οι αλγόριθμοι αυτοί για την περίπτωση singlerate και multirate μοντέλων. Η ρουτίνα rt_OneStep() καλείται από την ρουτίνα εξυπηρέτησης διακοπών υλικού που διακόπτει την main(). Αυτή, με τη σειρά της, καλεί τον κώδικα του μοντέλου που, στην περίπτωση multirate, είναι παραμετρικός ως προς τους χρόνους δειγματοληψίας.

4.2.3 Κώδικας εξαρτώμενος από το μοντέλο

Στην κατηγορία αυτή ανήκει ο κώδικας που εξαρτάται από το εκάστοτε μοντέλο, και παράγεται με τη διαδικασία της σύνθεσης που είδαμε στην παράγραφο 4.1. Περιλαμβάνει την συνάρτηση εκτέλεσης του μοντέλου και αρχεία επικεφαλίδας. Η συνάρτηση εκτέλεσης ενσωματώνει κώδικα για το κάθε block του μοντέλου με σειρά που καθορίζεται νωρίτερα στο στάδιο της ανάλυσης. Για την περίπτωση multirate μοντέλων περιλαμβάνει και συνθήκες ελέγχου της εκτέλεσης block ανάλογα με το ρυθμό δειγματοληψίας τους. Περισσότερα για τα ζητήματα εκτέλεσης των singlerate και multirate μοντέλων θα δούμε στην παράγραφο 4.3.1

Τα αρχεία επικεφαλίδας, που παράγονται, επιτρέπουν την ανταλλαγή πληροφορίας μεταξύ του κώδικα που εξαρτάται από το μοντέλο και των υπόλοιπων σταθερών

```
rt_OneStep()                                  /* SingleRate model */
{
    Check for interrupt overflow
    Enable "rt_OneStep" interrupt
    MODEL_step()                              /* Model Code */
}

rt_OneStep()                                  /* MultiRate model */
{
    Check for base-rate interrupt overflow
    Enable rt_OneStep() interrupt
    ModelStep(tid=0)                          /* Base-rate time step */
    For i=1:NumTasks                          /* Iterate over sub-rate tasks */
        Check for sub-rate interrupt overflow
        If (sub-rate task i is scheduled)
            ModelStep(tid=i)                  /* Sub-rate time step */
        EndIf
    EndFor
}
```

Σχήμα 4.4: Ψευδοκώδικας αλγόριθμου εκτέλεσης του μοντέλου για singlerate και multirate μοντέλα

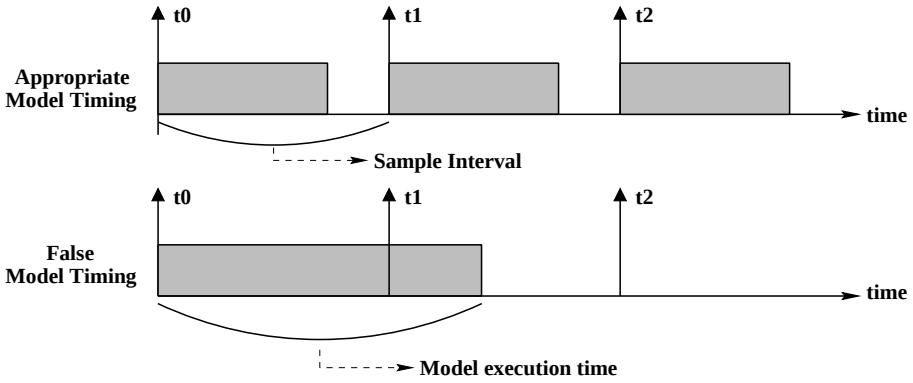
τμημάτων κώδικα. Η πληροφορία αυτή μπορεί να περιλαμβάνει παραμέτρους του συστήματος, δεδομένα που επεξεργάζεται το μοντέλο και τυποποιήσεις συναρτήσεων. Όλη η μνήμη που είναι απαραίτητη για την εκτέλεση του μοντέλου δηλώνεται στατικά. Επιπλέον, οι διάφορες δομές μνήμης ορίζονται μόνο εφόσον χρησιμοποιούνται και τα πεδία τους ποικίλουν. Τα δύο αυτά χαρακτηριστικά είναι συνέπεια της επιλογής της τυποποίησης Embedded.

4.3 Ζητήματα χρονισμού και μνήμης

4.3.1 Εκτέλεση του μοντέλου σε πραγματικό χρόνο

Όπως αναφέρθηκε προηγουμένως, από τη φάση της ανάλυσης του μοντέλου, προκύπτει μια στατική σειρά εκτέλεσης των υπολογισμών, που είναι περιοδική. Για την εκτέλεση μιας εφαρμογής σε πραγματικό χρόνο, απαιτείται η εκτέλεση κάθε επανάληψης να ολοκληρώνεται πριν παρέλθει το χρονικό διάστημα μιας περιόδου (πριν δηλαδή ξεκινήσει η εκτέλεση της επόμενης επανάληψης). Η απαίτηση αυτή, φαίνεται καθαρά στο Σχήμα 4.5.

Για να ικανοποιείται αυτή η προϋπόθεση σε μια πλατφόρμα υλικού, απαιτείται η χρήση μιας ρουτίνας ISR, που θα καλεί τον κώδικα του μοντέλου (rt_OneStep στην περίπτωση μας) και θα ενεργοποιείται είτε από διακοπή υλικού είτε από διακοπή



Σχήμα 4.5: Χρονισμός της εκτέλεσης του μοντέλου

λογισμικού. Στην πρώτη περίπτωση απαιτείται η χρήση ενός μετρητή (timer) που θα παράγει διακοπές με βάση τον ρυθμό δειγματοληψίας του μοντέλου. Στην δεύτερη περίπτωση η διακοπή θα ενεργοποιείται από κάποιο γεγονός, όπως η ανάγνωση ενός δείγματος από την είσοδο για επεξεργασία.

Αν οι υπολογισμοί του μοντέλου δεν προλαβαίνουν να ολοκληρωθούν στο διάστημα μιας περιόδου, υπάρχει πρόβλημα να εκτελεστεί η εφαρμογή σε πραγματικό χρόνο και πρέπει να γίνουν βελτιώσεις. Η καλύτερη αξιοποίηση της μνήμης μπορεί να βοηθήσει (παράγραφος 4.3.3) ή η διερεύνηση των δυνατοτήτων του μεταγλωττιστή και του εργαλείου σύνδεσης (παράγραφος 5.3).

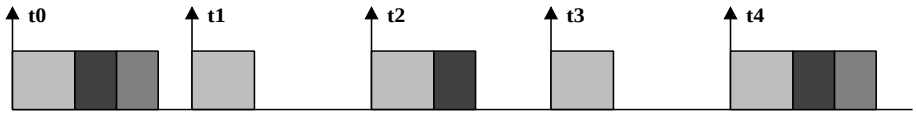
Επίσης η εκτέλεση των υπολογισμών σε πλαίσια (frames) δειγμάτων, αντί σε ένα μόνο δείγμα μοιράζει το κόστος από την ενεργοποίηση της ρουτίνας ISR και έτσι κατά περίπτωση μπορεί να επιτρέψει την εκτέλεση του μοντέλου σε πραγματικό χρόνο. Από την άλλη η χρήση πλαισίων μπορεί να φέρει το αντίθετο από το επιθυμητό αποτέλεσμα, γιατί η επεξεργασία μεγάλων πινάκων, από τη μία απαιτεί μεγάλο χώρο αποθήκευσης και από την άλλη αυξάνει το χρόνο επεξεργασίας ανά δείγμα.

4.3.2 Μοντέλα με πολλαπλούς χρόνους δειγματοληψίας

Τα παραπάνω αφορούν μοντέλα singlerate, όπου ο χρόνος εκτέλεσης μιας επανάληψης είναι σταθερός. Στα multirate μοντέλα όμως, υπάρχουν blocks που εκτελούνται πιο συχνά από άλλα. Έτσι ο χρόνος εκτέλεσης μιας επανάληψης δεν είναι πάντα ο ίδιος. Για να μπορεί να εκτελεστεί το μοντέλο σε πραγματικό χρόνο σε αυτή την περίπτωση δύο σχήματα είναι δυνατό να ακολουθηθούν. Το singletasking σχήμα και το multitasking. Το Real-Time Workshop υποστηρίζει τα δύο αυτά σχήματα μέσα από τις αντίστοιχες επιλογές προσομοίωσης της μεθόδου επίλυσης σταθερού βήματος (βλ. παράγραφο 3.3.1).

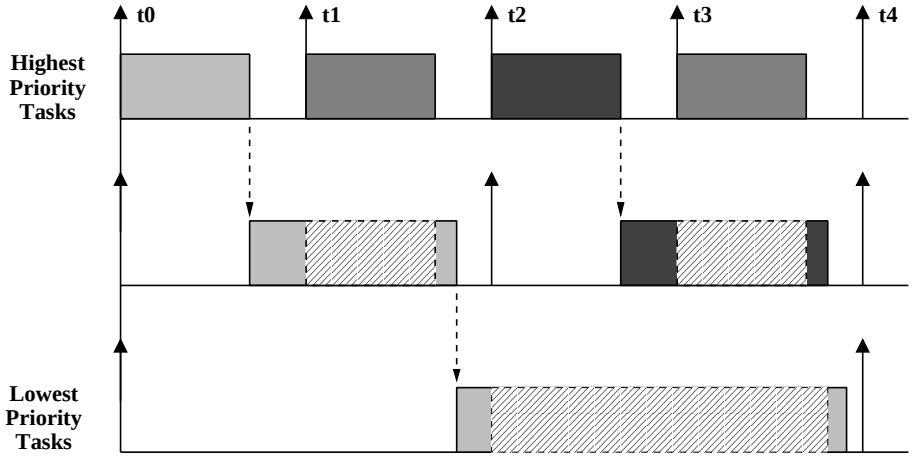
Το singletasking σχήμα, ορίζει ότι η περίοδος επανάληψης των υπολογισμών του μοντέλου είναι σταθερή και μεγαλύτερη ή ίση με το χρονικό διάστημα εκτέλεσης όλων

των blocks του μοντέλου (Σχήμα 4.6). Το σχήμα αυτό είναι εμφανώς λιγότερο αποδοτικό όσον αφορά την ταχύτητα εκτέλεσης και την αξιοποίηση του επεξεργαστή, αλλά σε ορισμένες περιπτώσεις επιτρέπει την απλοποίηση του μοντέλου.



Σχήμα 4.6: Singletasking εκτέλεση του μοντέλου

Το multitasking σχήμα, υλοποιεί ένα pseudomultitasking περιβάλλον. Σε ένα πραγματικό multitasking περιβάλλον τα blocks του μοντέλου ομαδοποιούνται με βάση το χρόνο δειγματοληψίας τους και διεργασίες (tasks) με διαφορετικές προτεραιότητες, αναλαμβάνουν την εκτέλεση κάθε ομάδας block. Blocks με μικρότερους χρόνους δειγματοληψίας εκτελούνται από διεργασίες με μεγαλύτερες προτεραιότητες και έτσι μπορούν να διακόψουν την εκτέλεση των πιο αργών blocks. Η συμπεριφορά αυτή προσομοιώνεται σε μια πλατφόρμα υλικού με τη βοήθεια αλληλοκαλυπτόμενων διακοπών υλικού (Σχήμα 4.7). Συγκεκριμένα, επιτρέπεται να συμβεί διακοπή υλικού ενώ εξυπηρετείται ακόμη η προηγούμενη, για να εκτελεστούν τμήματα του μοντέλου με μεγαλύτερη προτεραιότητα. Όταν ολοκληρωθεί η εκτέλεση τους, ο έλεγχος επιστρέφει στην ISR που είχε διακοπεί.



Σχήμα 4.7: Multitasking εκτέλεση του μοντέλου

4.3.3 Ζητήματα μνήμης

Ζητήματα μνήμης εγείρονται σε πλατφόρμες με περιορισμούς είτε στο μέγεθος της μνήμης είτε στο ρολόι του επεξεργαστή. Αφορούν, δε, από την μια την επάρκεια

του χώρου αποθήκευσης για τα δεδομένα και το πρόγραμμα και από την άλλη, την κατάλληλη κατανομή των δεδομένων αυτών ώστε να μειώνεται ο χρόνος εκτέλεσης του προγράμματος. Το δεύτερο, αναφέρεται κυρίως σε πλατφόρμες επεξεργασίας σήματος, όπου υπάρχουν πολλαπλές εσωτερικές τράπεζες μνήμης με δυνατότητα παράλληλης προσπέλασης και εξωτερική μνήμη με χαμηλότερες επιδόσεις.

Παρακάτω παρουσιάζονται κάποιες γενικές προσεγγίσεις που μπορεί να ακολουθήσει ο χρήστης για την καλύτερη αξιοποίηση της μνήμης. Επειδή το τελικό αποτέλεσμα εξαρτάται ισχυρά από ζητήματα του υλικού, τα θέμα αυτό θα θιγεί ξανά στο Κεφάλαιο 5.

4.3.3.1 Δυναμική και στατική δέσμευση μνήμης

Η μνήμη που απαιτείται για την εκτέλεση του μοντέλου είναι δυνατό να δεσμευτεί είτε δυναμικά είτε στατικά. Κάθε μια από τις δύο επιλογές έχει τα πλεονεκτήματα της. Η δυναμική δέσμευση είναι δυνατό να είναι περισσότερο αποδοτική σε ορισμένες περιπτώσεις, όπου η ίδια μνήμη μπορεί να αποδεσμεύεται και να ξαναχρησιμοποιείται σε άλλο σημείο του κώδικα. Επιπλέον με τη χρήση δεικτών (pointers) είναι δυνατό να αποφευχθούν άσκοπες αλλά χρονοβόρες αντιγραφές μνήμης. Από την άλλη, με την στατική δέσμευση γνωρίζει κανείς τη στιγμή της δημιουργίας του εκτελέσιμου πόση μνήμη απαιτείται, γεγονός που μπορεί να αποτρέψει λάθη κατά την εκτέλεση της εφαρμογής ειδικά σε πλατφόρμες υλικού με περιορισμένη μνήμη.

Σε περίπτωση επιλογής της *Embedded* τυποποίησης είναι υποχρεωτική η δέσμευση της μνήμης στατικά. Από την άλλη η τυποποίηση *RealTime* υποστηρίζει τόσο την στατική όσο και την δυναμική δέσμευση.

4.3.3.2 Τοπική και γενική δέσμευση μνήμης

Μέσα από το μενού επιλογών του RTW, δίνεται η επιλογή τοπικής (local) έναντι γενικής (global) δέσμευσης μνήμης. Στην πρώτη περίπτωση, ορισμένα blocks δεσμεύουν την μνήμη που χρειάζονται τοπικά, (μέσα στην συνάρτηση που καλεί τον κώδικα τους) αλλιώς όλη η απαιτούμενη μνήμη δεσμεύεται γενικά. Η τοπική δέσμευση μνήμης έχει το πλεονέκτημα ότι τα δεδομένα που χρειάζεται μια συνάρτηση βρίσκονται στη στοίβα και έτσι είναι μικρός ο χρόνος ανάκτησης τους (γιατί μπορούν να διευθυνσιοδοτηθούν άμεσα). Το πλεονέκτημα αυτό μπορεί να μην ισχύει, όταν τα τοπικά δεδομένα είναι πολλά (π.χ. μεγάλοι πίνακες). Επιπλέον όταν τα δεδομένα είναι σταθερά (π.χ. οι συντελεστές ενός φίλτρου) υπάρχει το κόστος της αντιγραφής τους στη στοίβα από μια άλλη θέση μνήμης κάθε φορά που η συνάρτηση καλείται. Χρειάζεται, επομένως, προσεκτικός χειρισμός της επιλογής αυτής για να μην υπάρχει μεγαλύτερη απώλεια από ότι κέρδος.

Είναι δυνατό τα αποτελέσματα της επιλογής αυτής να ελεγχθούν με τη βοήθεια των μεταβλητών MaxStackSize και MaxVariableStackSize του Target Language Compiler. Με την πρώτη περιορίζεται το συνολικό μέγεθος μνήμης που απαιτούν οι τοπικές

μεταβλητές όλων των συναρτήσεων του μοντέλου, ενώ με την δεύτερη περιορίζεται το μέγεθος της κάθε μιας τοπικής μεταβλητής. Οι μεταβλητές αυτές ορίζονται στο system target file.

4.3.3.3 Τύποι δεδομένων των σημάτων του μοντέλου

Το Simulink, εξ ορισμού χρησιμοποιεί τον τύπο double για τα σήματα ενός μοντέλου. Όταν είναι γνωστό εκ των προτέρων ότι το εύρος τιμών ενός σήματος δεν απαιτεί την ακρίβεια του τύπου αυτού είναι προτιμότερο να δηλωθεί ρητά ο κατάλληλος τύπος. Κατά περιπτώσεις, αυτή η μικρή βελτίωση του μοντέλου, μπορεί να μειώσει σημαντικά τις απαιτήσεις σε μνήμη δεδομένων (λιγότερα bytes για την αποθήκευση κάθε μεταβλητής) και προγράμματος (λιγότερες εντολές για την επεξεργασία κάθε μεταβλητής).

4.4 Συστατικά μιας πλατφόρμας εκτέλεσης

Από όσα αναφέρθηκαν παραπάνω είναι φανερό ότι για να αναπτύξει κανείς μια νέα πλατφόρμα εκτέλεσης θα πρέπει να παρέχει τα εξής:

- Ένα system target file, που θα ορίζει τις επιλογές RTW που υποστηρίζονται από την πλατφόρμα εκτέλεσης και θα παρέχει το γενικό πλαίσιο για την παραγωγή κώδικα.
- Ένα πρότυπο Makefile, προσαρμοσμένο στα εργαλεία μεταγλώττισης της πλατφόρμας.
- Ρουτίνες για αρχικοποίηση και χειρισμό του υλικού.
- Οδηγούς συσκευής για τη διάρκεια της προσομοίωσης (με τη μορφή S-function files) και για τη διάρκεια της εκτέλεσης (με τη μορφή inlined S-functions).
- Ρουτίνες χειρισμού των διακοπών υλικού για την εκτέλεση του μοντέλου και για άλλες χρήσεις (π.χ. ανάγνωση δειγμάτων από συσκευή εισόδου).
- Τη συνάρτηση main(), που συνδέει τα διάφορα τμήματα της εφαρμογής
- 'Αλλο κώδικα που είναι απαραίτητος για το χειρισμό της πλατφόρμας.

Επίσης είναι πιθανό να χρειαστεί να επέμβει σε κώδικα που παρέχει το RTW για να υποστηρίξει τις ιδιαιτερότητες της αρχιτεκτονικής του ή του μεταγλωττιστή του.

4.4.1 Δημιουργία οδηγών συσκευής

Οι οδηγοί συσκευής που συνοδεύουν μια πλατφόρμα εκτέλεσης, έχουν διπλή λειτουργία. Από τη μία, λειτουργούν ως τμήματα κώδικα που συνδέονται με τον υπόλοιπο κώδικα

του μοντέλου για την δημιουργία μιας εφαρμογής πραγματικού χρόνου. Από την άλλη, πρέπει να μπορούν να αλληλεπιδράσουν με το Simulink κατά τη διάρκεια της προσομοίωσης. Για να είναι και οι δύο λειτουργίες δυνατές, οι οδηγοί συσκευής πρέπει να γραφούν ακολουθώντας το S-function API και να ενσωματωθούν σε ένα block του Simulink.

Συνήθως, οι λειτουργίες που εκτελεί ένας οδηγός συσκευής κατά τη διάρκεια της προσομοίωσης δεν είναι ίδιες με αυτές που εκτελεί κατά την εκτέλεση της εφαρμογής (π.χ. συνήθως δεν επικοινωνεί με το υλικό κατά τη διάρκεια της προσομοίωσης). Αυτό μπορεί να αντιμετωπιστεί χρησιμοποιώντας συνθήκες ελέγχου που επιτρέπουν άλλα τμήματα του κώδικα να διαβάζονται κατά τη διάρκεια της προσομοίωσης και άλλα κατά την παραγωγή κώδικα.

Σε περίπτωση χρήσης, όμως, της τυποποίησης Embedded κάτι τέτοιο δεν είναι δυνατό. Όπως αναφέρθηκε και νωρίτερα, για να μπορεί να παραχθεί κώδικας από ένα S-function block πρέπει να υπάρχει η αντίστοιχη TLC αναπαράστασή του. Έτσι τελικά, ο οδηγός συσκευής έχει δύο υλοποιήσεις, μια για χρήση κατά τη διάρκεια της προσομοίωσης (S-function block) και μια για χρήση κατά την παραγωγή κώδικα (TLC block). Αυτές οι δύο υλοποιήσεις επικοινωνούν μέσω της ενδιάμεσης αναπαράστασης του μοντέλου. Αυτή η επικοινωνία είναι απαραίτητη για την μεταβίβαση των παραμέτρων του οδηγού συσκευής στην διαδικασία παραγωγής κώδικα. Συγκεκριμένα, κατά το στάδιο της ανάλυσης του μοντέλου, οι παράμετροι του S-function block αποθηκεύονται στην ενδιάμεση αναπαράσταση από όπου διαβάζονται κατά το στάδιο της παραγωγής κώδικα από το αντίστοιχο TLC block.

Κεφάλαιο 5

Η πλατφόρμα TI EVM C30

Η πλατφόρμα TMS320C30 Evaluation Module της Texas Instruments (για συντομία EVM C30) είναι ένας συνδυασμός εργαλείων υλικού και λογισμικού, που χρησιμοποιούνται στην ανάπτυξη εφαρμογών επεξεργασίας σήματος. Αποτελείται από την πλακέτα υλικού EVM C30, που συνδέεται σε προσωπικό υπολογιστή μέσω μιας εσωτερικής θύρας τύπου EISA, και από μια συλλογή εργαλείων ανάπτυξης λογισμικού που εκτελούνται στον υπολογιστή. Χρησιμοποιώντας τα εργαλεία αυτά, αναπτύσσεται το DSP πρόγραμμα, που στη συνέχεια μεταφέρεται στην πλακέτα για εκτέλεση, μέσω της θύρας επικοινωνίας.

Το βασικό κύκλωμα της πλακέτας είναι ο επεξεργαστής TMS320C30 της Texas Instruments, που διαθέτει διασυνδέσεις με τα υπόλοιπα συστατικά της πλακέτας. Αυτά περιλαμβάνουν ένα αναλογικό κύκλωμα εισόδου/εξόδου (Analog Interface Circuit, AIC), μια σειριακή πόρτα, ένα κύκλωμα εξομοίωσης και μνήμη. Στην παρούσα εργασία δεν χρησιμοποιήθηκαν η σειριακή πόρτα και το κύκλωμα εξομοίωσης.

Στις επόμενες παραγράφους, παρουσιάζονται τα βασικά στοιχεία της αρχιτεκτονικής του επεξεργαστή (κεντρική μονάδα επεξεργασίας, οργάνωση της μνήμης, διακοπές υλικού και περιφερειακές συσκευές). Στη συνέχεια, περιγράφεται η λειτουργία του αναλογικού κυκλώματος εισόδου/εξόδου. Τέλος, παρουσιάζονται τα εργαλεία ανάπτυξης λογισμικού που συνοδεύουν την πλακέτα (μεταγλωττιστής και εργαλείο σύνδεσης).

5.1 Ο επεξεργαστής TMS320C30

5.1.1 Κεντρική μονάδα επεξεργασίας

Η κεντρική μονάδα επεξεργασίας του TMS320C30, αποτελείται από τα παρακάτω τμήματα:

- Μια αριθμητική και λογική μονάδα (ALU) και έναν ολισθητή μεγέθους 32 bit. Η ALU, εκτελεί σε ένα κύκλο ρολογιού, πράξεις ανάμεσα σε ακέραιους αριθμούς μεγέθους 32 bit και σε αριθμούς κινητής υποδιαστολής μεγέθους 40 bit. Το αποτέλεσμα πάντα διατηρείται στα πλαίσια των 32 και των 40 bit αντίστοιχα. Επίσης υποστηρίζει μετατροπές, μέσα σε ένα κύκλο ρολογιού, μεταξύ των δύο τύπων αριθμών που προαναφέρθηκαν.

- Μια μονάδα πολλαπλασιασμού. Η μονάδα αυτή εκτελεί σε ένα κύκλο ρολογιού πολλαπλασιασμό με ακέραιους αριθμούς μεγέθους 24 bit και με αριθμούς κινητής υποδιαστολής μεγέθους 32 bit. Το αποτέλεσμα είναι ένας αριθμός μεγέθους 32 και 40 bit αντίστοιχα. Η μονάδα πολλαπλασιασμού δουλεύει παράλληλα με την ALU.

- Δύο βοηθητικές αριθμητικές μονάδες (ARAUs) για τον υπολογισμό δύο διευθύνσεων σε κάθε κύκλο ρολογιού. Οι μονάδες αυτές δουλεύουν παράλληλα με τις δύο προηγούμενες. Χρησιμοποιούνται για να υποστηρίζουν τους διάφορους τρόπους διευθυνσιοδότησης του επεξεργαστή, όπως χρήση displacement, index registers, bit-reversed και circular διευθυνσιοδότηση.

- Τέσσερις εσωτερικούς διαδρόμους, που επιτρέπουν τη χρήση δύο τελεστών από τη μνήμη και δύο από τους καταχωρητές για την εκτέλεση παράλληλων πράξεων πρόσθεσης/αφαίρεσης και πολλαπλασιασμού σε κάθε κύκλο ρολογιού.

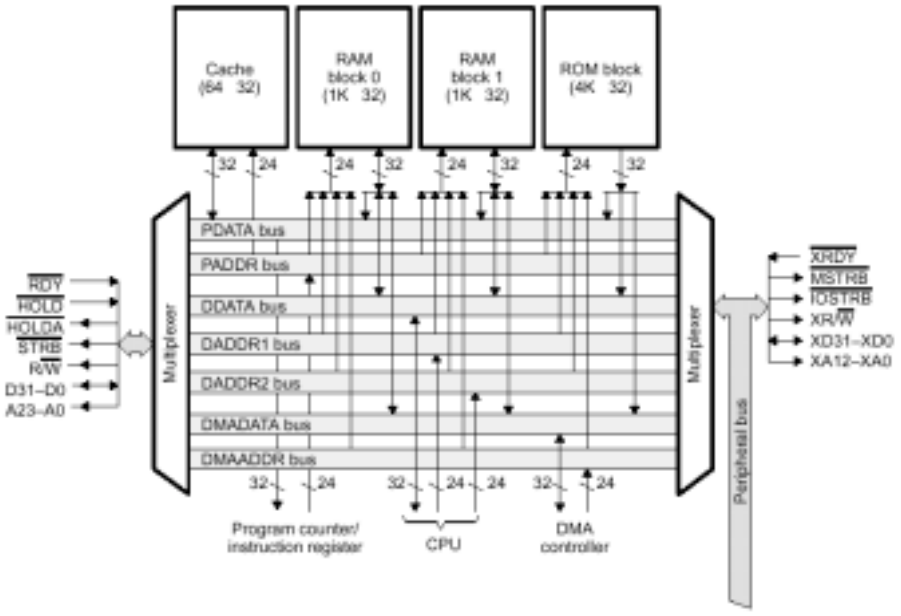
- Μνήμη καταχωρητών (register file), που περιλαμβάνει οκτώ καταχωρητές εκτεταμένης ακρίβειας, που χρησιμοποιούνται σε αριθμητικές πράξεις, οκτώ βοηθητικούς καταχωρητές και δύο καταχωρητές δείκτες, που χρησιμοποιούνται από τις ARAUs στο υπολογισμό διευθύνσεων, και άλλους έντεκα καταχωρητές που εκτελούν συγκεκριμένες λειτουργίες. Οι τελευταίοι περιλαμβάνουν τον καταχωρητή κατάστασης (status register), τον μετρητή προγράμματος (program counter), τον καταχωρητή εντολών (instruction register) κ.ά..

5.1.2 Οργάνωση της μνήμης

Όπως αναφέρθηκε στο Κεφάλαιο 2, είναι συνηθισμένο χαρακτηριστικό των επεξεργαστών ψηφιακού σήματος να χρησιμοποιούνται πολλαπλές τράπεζες μνήμης σε συνδυασμό με παράλληλους διαδρόμους για την αύξηση του εύρους μνήμης. Αντίστοιχη οργάνωση ακολουθεί και η μνήμη του επεξεργαστή TMS320C30. Όπως φαίνεται και στο Σχήμα 5.1, αποτελείται από δύο blocks μνήμης RAM μεγέθους 1K λέξεων το καθένα, ένα block μνήμης ROM μεγέθους 4K λέξεων και cache εντολών μεγέθους 64 λέξεων. Καθένα από αυτά τα blocks μνήμης υποστηρίζουν δύο προσπελάσεις από την CPU σε ένα κύκλο ρολογιού[1].

[1]Οι δύο προσπελάσεις μνήμης δεν χρειάζεται να υποστηρίζονται από την cache, γιατί σε έναν κύκλο ρολογιού γίνεται μόνο μια προσπέλαση εντολής

Επίσης, μέσω των δύο εξωτερικών διαδρόμων, υπάρχει η δυνατότητα προσπέλασης εξωτερικής μνήμης μεγέθους έως 16M λέξεων. Η πλακέτα EVM C30, διαθέτει ένα block μνήμης SRAM μεγέθους 16K λέξεων, που συνδέεται με τον επεξεργαστή μέσω του κύριου εξωτερικού διαδρόμου (primary bus). Η μνήμη αυτή είναι zero-wait state, που σημαίνει ότι μπορεί να προσπελαστεί σε έναν κύκλο ρολογιού.



Σχήμα 5.1: Η οργάνωση της μνήμης στον επεξεργαστή TMS320C30

Η μνήμη που προαναφέρθηκε (εσωτερική και εξωτερική), μπορεί να οργανωθεί σε διευθύνσεις με δύο τρόπους, ανάλογα αν ο επεξεργαστής έχει οριστεί σαν μικροεπεξεργαστής ή σαν μικροϋπολογιστής. Ο ορισμός αυτός γίνεται σε επίπεδο υλικού, και στην πλακέτα EVM C30 έχει επιλεγεί ο ορισμός του μικροεπεξεργαστή. Συνέπεια αυτής της επιλογής είναι η αδυναμία χρήσης της μνήμης ROM, που δεν αντιστοιχίζεται σε κάποιο χώρο διευθύνσεων.

Οι δυνατότητα προσπέλασης πολλαπλών θέσεων μνήμης σε ένα κύκλο ρολογιού, σε συνδυασμό με την ύπαρξη πολλαπλών ανεξάρτητων διαδρόμων προσπέλασης, επιτρέπει την εκτέλεση σε ένα κύκλο ρολογιού εντολών που απαιτούν έως και 3 προσπελάσεις μνήμης. Για να μπορεί να αξιοποιηθεί αυτή η δυνατότητα στο μέγιστο, πρέπει τα δεδομένα και το πρόγραμμα να είναι κατανεμημένα κατάλληλα στα blocks μνήμης. Αν δεν ισχύει αυτό, τότε από τη μία αυξάνεται ο χρόνος εκτέλεσης της εντολής και από την άλλη υποχρησιμοποιείται η μνήμη.

Υπάρχουν συγκεκριμένοι συνδυασμοί προσπέλασης δεδομένων και προγράμματος που μπορούν να αξιοποιήσουν το εύρος μνήμης επιτυγχάνοντας χρόνο εκτέλεσης των εντολών ενός κύκλου ρολογιού. Οι συνδυασμοί αυτοί, που περιλαμβάνουν την προσπέλαση μιας εντολής και ενός ή δύο δεδομένων, φαίνονται στον Πίνακα 5.1. Στην

Προσπελάσεις εξωτερικής μνήμης	Προσπελάσεις εσωτερικής μνήμης
1	1
---	2 από το ίδιο η διαφορετικό block μνήμης
1	2 από το ίδιο η διαφορετικό block μνήμης
---	2 από το ίδιο και 1 από διαφορετικό block μνήμης
---	3 από διαφορετικά blocks

Πίνακας 5.1: Συνδυασμοί προσπέλασης μνήμης για μέγιστη απόδοση σε χρόνο εκτέλεσης και αξιοποίηση μνήμης

εσωτερική μνήμη θεωρούμε ότι ανήκει και η cache.

5.1.3 Διακοπές υλικού

Ο επεξεργαστής TMS320C30, υποστηρίζει τέσσερις εξωτερικές διακοπές υλικού και αρκετές εσωτερικές, που μπορεί να παράγονται από τον ελεγκτή DMA, τα χρονόμετρα και τις σειριακές πόρτες. Κάθε διακοπή, έχει μια σειρά προτεραιότητας, που επιτρέπει την αντιμετώπιση συγκρούσεων με έναν προκαθορισμένο τρόπο, σε περίπτωση ταυτόχρονης παραγωγής διακοπών.

Ο πίνακας διανυσμάτων διακοπών (interrupt vector table), καταλαμβάνει τις πρώτες 40 θέσεις (λέξεις) μνήμης και περιέχει διευθύνσεις ρουτινών εξυπηρέτησης διακοπών (ISRs). Η κάθε θέση μνήμης αντιστοιχεί σε μια συγκεκριμένη διακοπή, Η σύμβαση αυτή είναι ευθύνη του προγραμματιστή να τηρείται. Οι διακοπές υλικού που υποστηρίζονται από τον TMS320C30, οι θέσεις τους στον πίνακα διανυσμάτων διακοπών και οι προτεραιότητες τους φαίνονται στον Πίνακα 5.2.

Με τις διακοπές υλικού, σχετίζονται δύο καταχωρητές του επεξεργαστή, ο *καταχωρητής ενεργοποίησης διακοπών* IE (interrupt enable) και ο *καταχωρητής σημαίας διακοπών* IF (interrupt flag), και το πεδίο *GIE* (global interrupts enable bit) του καταχωρητή κατάστασης.

Οι καταχωρητές αυτοί χρησιμοποιούνται για τον έλεγχο των διακοπών. Συγκεκριμένα το πεδίο GIE, χρησιμοποιείται για την συνολική ενεργοποίηση των διακοπών του επεξεργαστή ενώ επιμέρους διακοπές ενεργοποιούνται από τον καταχωρητή IE. Για την ενεργοποίηση μιας διακοπής απαιτείται να τεθούν στην τιμή 1, τόσο το πεδίο GIE όσο και το κατάλληλο πεδίο του καταχωρητή IE. Ο καταχωρητής IF, χρησιμοποιείται για να δοθεί το έναυσμα για μια διακοπή. Όταν κάποιο πεδίο του καταχωρητή αυτού τίθεται στην τιμή 1, ο επεξεργαστής αναγνωρίζει την παραγωγή μιας διακοπής και ξεκινάει τις προετοιμασίες εκτέλεσης της ρουτίνας εξυπηρέτησης της αντίστοιχης διακοπής. Ο καταχωρητής αυτός, είναι προσπελάσιμος μέσω λογισμικού (επιτρέπεται

Διακοπή	Θέση Διανύσματος	Προτεραιότητα	Λειτουργία
RESET	0h	0	Εξωτερικό σήμα reset
INT0	1h	1	Εξωτερική διακοπή INT0
INT1	2h	2	Εξωτερική διακοπή INT1
INT2	3h	3	Εξωτερική διακοπή INT2
INT3	4h	4	Εξωτερική διακοπή INT3
XINT0	5h	5	Διακοπή μετάδοσης της σειριακής πόρτας 0.
RINT0	6h	6	Διακοπή λήψης της σειριακής πόρτας 0.
XINT1	7h	7	Διακοπή μετάδοσης της σειριακής πόρτας 1.
RINT1	8h	8	Διακοπή λήψης της σειριακής πόρτας 1.
TINT0	9h	9	Διακοπή του χρονόμετρου 0.
TINT1	Ah	10	Διακοπή του χρονόμετρου 1.
DINT0	Bh	11	Διακοπή του ελεγκτή DMA.

Πίνακας 5.2: Προτεραιότητες και λειτουργία διακοπών υλικού

δηλαδή να αλλάξει από τον προγραμματιστή) γεγονός που επιτρέπει την ενεργοποίηση μιας διακοπής μέσω λογισμικού. Στο Κεφάλαιο 6 θα δούμε μια εφαρμογή αυτής της δυνατότητας.

Το πεδίο GIE, χρησιμοποιείται από τον επεξεργαστή σαν μέθοδος αποτροπής των αλληλοκαλυπτόμενων διακοπών. Για το σκοπό αυτό, η πρώτη εργασία που εκτελείται μόλις διαπιστωθεί ότι έχει συμβεί μια διακοπή, είναι ο μηδενισμός του. Έτσι, αν είναι επιθυμητή η διακοπή της ρουτίνας εξυπηρέτησης, πρέπει να τεθεί αυτό το πεδίο στην τιμή 1 μέσα στην ρουτίνα.

Αρκετά άλλα θέματα που σχετίζονται με τη λειτουργία των διακοπών, όπως η καθυστέρηση αναγνώρισης μιας διακοπής, ο χρόνος που μεσολαβεί από την αναγνώρισή της έως την έναρξη της εκτέλεσης της ρουτίνας εξυπηρέτησης και το κόστος κλήσης μιας ISR (αποθήκευση και ανάκτηση καταχωρητών), μπορούν να βρεθούν στο [34, Κεφάλαιο 7]

5.1.3.1 Χρήση διακοπών υλικού με τη C

Μια ρουτίνα εξυπηρέτησης διακοπών μπορεί να είναι γραμμένη σε assembly ή σε C. Όταν γράφεται σε C, πρέπει να ακολουθεί συγκεκριμένη ονοματολογία, ώστε να αναγνωρίζεται ως ISR από τον μεταγλωττιστή και έτσι να ακολουθούνται διαφορετικοί κανόνες αποθήκευσης και ανάκτησης καταχωρητών κατά την κλήση της. Το όνομα της

ρουτίνας πρέπει να είναι της μορφής *c_intnn*, όπου *nn* είναι ένας αριθμός μεταξύ 00 και 99. Για βελτίωση της αναγνωσιμότητας του κώδικα είναι δυνατό να δοθεί ένα συνώνυμο στο c_intnn με τη βοήθεια μακροεντολών. Το όνομα της ρουτίνας δεν καθορίζει την λειτουργία της (αν δηλαδή είναι διακοπή χρονόμετρου ή διακοπή σειριακής πόρτας κλπ). Από τη στιγμή που θα δημιουργηθεί η ISR, πρέπει να δηλωθεί η διεύθυνση της κατάλληλα στον πίνακα διανυσμάτων διακοπών. Η δήλωση αυτή γίνεται σε assembly κώδικα.

Δύο κανόνες έχουν εφαρμογή στις ISRs. Πρώτον, δεν δέχονται καμία παράμετρο και δεύτερον, δεν επιστρέφουν καμία παράμετρο. 'Αρα μια ISR επικοινωνεί με το υπόλοιπο πρόγραμμα μόνο μέσω γενικών μεταβλητών. 'Αλλος ένας κανόνας, που δεν είναι υποχρεωτικός αλλά είναι καλό να τηρείται, αφορά την κλήση άλλων συναρτήσεων μέσα από ISR. Επειδή ο μεταγλωττιστής δεν έχει τρόπο να γνωρίζει ποιους καταχωρητές χρησιμοποιεί αυτή η συνάρτηση, αποθηκεύει και ανακτά όλους τους καταχωρητές. Έτσι είναι καλό να μην καλούνται άλλες συναρτήσεις μέσα από μια ISR.

5.1.4 Περιφερειακές συσκευές

Ο επεξεργαστής TMS320C30, έχει τέσσερις περιφερειακές συσκευές. Δύο χρονόμετρα (timers) και δύο σειριακές πόρτες (serial ports). Η λειτουργία τους ελέγχεται μέσω memory-mapped καταχωρητών, που είναι προσπελάσιμοι από ένα αποκλειστικό διάδρομο. Αυτό επιτρέπει την προσπέλαση τους ταυτόχρονα με άλλες θέσεις μνήμης.

Το γεγονός ότι οι καταχωρητές ελέγχου είναι προσπελάσιμοι μέσω της μνήμης, μπορεί να αξιοποιηθεί για τη δημιουργία μιας C βιβλιοθήκης, που θα επιτρέπει τον άνετο χειρισμό τους μέσα από τη C.

Αναλυτική παρουσίαση της λειτουργίας των περιφερειακών συσκευών και των αντίστοιχων καταχωρητών ελέγχου μπορεί να βρεθεί στο [34, Κεφάλαιο 12]. Παρακάτω παρουσιάζονται τα βασικότερα χαρακτηριστικά τους.

5.1.4.1 Χρονόμετρα

Τα χρονόμετρα μπορούν να χρησιμοποιηθούν είτε με εσωτερικό ρολόι (του επεξεργαστή) είτε με εξωτερικό, για να ενεργοποιήσουν λειτουργίες ανά τακτά χρονικά διαστήματα ή για να μετρήσουν εξωτερικά γεγονότα. Για παράδειγμα, ένα χρονόμετρο μπορεί να χρησιμοποιηθεί για να ενεργοποιήσει μια εξωτερική λειτουργία μετατροπής αναλογικού σε ψηφιακό, που θέλουμε να εκτελεστεί με συχνότητα μικρότερη από αυτήν του επεξεργαστή. Για να χρησιμοποιηθεί ένα χρονόμετρο πρέπει να διαμορφωθεί κατάλληλα ο σχετικός καταχωρητής ελέγχου.

Ένα χρονόμετρο, παίρνει σαν είσοδο τους παλμούς του ρολογιού του επεξεργαστή ή ενός εξωτερικού ρολογιού και βγάζει στην έξοδο παλμούς με συχνότητα που είναι υποπολλαπλάσια από αυτή της εισόδου. Το μέγεθος της υποδιαίρεσης καθορίζεται σε έναν από τους καταχωρητές ελέγχου του χρονόμετρου, που ονομάζεται καταχωρητής περιόδου (period register).

Η μέγιστη συχνότητα, που παράγει το χρονόμετρο, όταν χρησιμοποιείται εσωτερικό ρολόι, είναι το 1/4 της συχνότητας ρολογιού του επεξεργαστή. Έτσι η μέγιστη συχνότητα εξόδου στην περίπτωση μας είναι τα 7.5MHz.

Κάθε χρονόμετρο συνδέεται με μια διακοπή υλικού και μπορεί να χρησιμοποιηθεί για την εκτέλεση μιας λειτουργίας σε τακτά χρονικά διαστήματα.

5.1.4.2 Σειριακές πόρτες

Οι σειριακές πόρτες του TMS320C30, υποστηρίζουν την ταυτόχρονη μετάδοση και προς τις δύο κατευθύνσεις (bidirectional), λέξεων δεδομένων των 8, 16, 24 και 32 bit. Ο χρονισμός τους μπορεί να παράγεται είτε εσωτερικά από χρονόμετρα που διαθέτουν οι ίδιες, είτε εξωτερικά. Η χρήση τους, απαιτεί τη διαμόρφωση τριών καταχωρητών ελέγχου με τις κατάλληλες ρυθμίσεις.

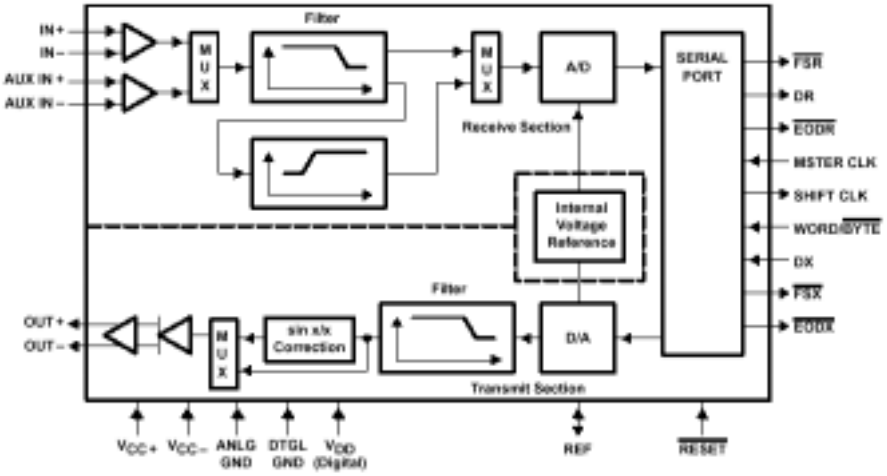
Κάθε σειριακή πόρτα συνδέεται με δύο διακοπές υλικού. Μία για τη μετάδοση δεδομένων και μια για την λήψη. Οι διακοπές αυτές ενεργοποιούνται από τις πράξεις της μετάδοσης δεδομένων και της εγγραφής δεδομένων στους αντίστοιχους καταχωρητές της σειριακής (data transmit register και data receive register). Στην περίπτωση της λήψης δεδομένων, απαιτείται η ανάγνωση του καταχωρητή λήψης εγκαίρως. Διαφορετικά, τα νέα δεδομένα, που φτάνουν στη σειριακή, αγνοούνται.

5.2 Το αναλογικό κύκλωμα εισόδου/εξόδου

Το αναλογικό κύκλωμα εισόδου/εξόδου TLC32044 [36], που έχει χρησιμοποιηθεί στην πλακέτα EVM C30, είναι ένα πλήρες σύστημα μετατροπής αναλογικού σε ψηφιακό και αντίστροφα. Οι συχνότητες δειγματοληψίας που υποστηρίζει (7.200Hz έως 19.200Hz), το κάνουν κατάλληλο για χρήση σε εφαρμογές επεξεργασίας φωνής (π.χ. modems, σύνθεση φωνής, αναγνώριση ομιλίας). Ενσωματώνει:

- ένα αναλογικό ζωνοπερατό φίλτρο εισόδου για την αντιμετώπιση του φαινομένου των ψευδώνυμων συχνοτήτων, τεχνολογίας switched capacitor, που αποτελείται από ένα υψιπερατό φίλτρο με συχνότητα μετάβασης τα 150Hz και ένα χαμηλοπερατό φίλτρο με συχνότητα αποκοπής τα 3.600Hz,
- έναν μετατροπέα αναλογικού σε ψηφιακό με ανάλυση 14 bit,
- σειριακή πόρτα για επικοινωνία με τον επεξεργαστή,
- έναν μετατροπέα ψηφιακού σε αναλογικό με ανάλυση 14 bit και
- ένα χαμηλοπερατό φίλτρο αποκατάστασης της εξόδου ακολουθούμενο από ένα φίλτρο διόρθωσης.

Στο Σχήμα 5.2 φαίνεται το χονδρικό σχηματικό διάγραμμα του κυκλώματος.



Σχήμα 5.2: Το χονδρικό διάγραμμα του αναλογικού κυκλώματος εισόδου/εξόδου

Κάποια από τα χαρακτηριστικά του AIC, μπορούν να προγραμματιστούν από τον επεξεργαστή με τη βοήθεια ενός απλού πρωτόκολλου επικοινωνίας. Τα χαρακτηριστικά αυτά είναι:

- Η συχνότητα δειγματοληψίας, που μπορεί να είναι 7.200Hz, 8.000Hz, 9.600Hz, 14.400Hz και 19.200Hz.
- Η δυνατότητα ασύγχρονης ή σύγχρονης λειτουργίας των κυκλωμάτων λήψης και αποστολής. Στην ασύγχρονη λειτουργία, οι μονάδες μετατροπής αναλογικό σε ψηφιακό και αντίστροφα μπορούν να λειτουργούν με διαφορετικό ρυθμό δειγματοληψίας.
- Η χρήση του υψιπερατού φίλτρου εισόδου.
- Η χρήση του φίλτρου διόρθωσης της εξόδου.
- Ο βαθμός ενίσχυσης της αναλογικής εισόδου, που εξαρτάται από το εύρος της τάσης εισόδου.
- Η λειτουργία loop back, κατά την οποία η έξοδος του κυκλώματος συνδέεται εσωτερικά στην είσοδο. Με κατάλληλο πρόγραμμα ελέγχου που τρέχει στον επεξεργαστή μπορεί να γίνει έλεγχος της ακρίβειας μετατροπής αναλογικού σε ψηφιακό και αντίστροφα.

Όλες οι λειτουργίες του AIC, οδηγούνται από ένα σήμα εισόδου, το MSTR_CLK, που πρέπει να παρέχεται εξωτερικά. Στην περίπτωση μας παράγεται από ένα από τα χρονόμετρα του επεξεργαστή. Έτσι οι λειτουργίες του AIC, είναι συγχρονισμένες με το ρολόι του επεξεργαστή. Από το σήμα αυτό παράγονται τα σήματα που οδηγούν τα φίλτρα εισόδου (στα 288kHz), τις μονάδες μετατροπής και τις μεταφορές δεδομένων από

και προς το AIC. Η συχνότητα του MSTR_CLK δεν πρέπει να ξεπερνάει τα 10MHz, που είναι η μέγιστη συχνότητα λειτουργίας του AIC. Από την άλλη, η ελάχιστη συχνότητα λειτουργίας του κυκλώματος είναι τα 75KHz, που ορίζεται από την ανάγκη ανανέωσης εσωτερικών σημάτων του κυκλώματος, που είναι αποθηκευμένα σε πυκνωτές.

Το AIC, επικοινωνεί με τον επεξεργαστή μέσω της σειριακής πόρτας, που συνδέεται με μια από τις δύο σειριακές πόρτες του επεξεργαστή. Η επικοινωνία αυτή ενεργοποιείται από τη μεριά του επεξεργαστή, που γράφει για πρώτη φορά στον καταχωρητή μετάδοσης της σειριακής του. Στη συνέχεια, ακολουθούν ορισμένες εντολές διαμόρφωσης της λειτουργίας του AIC (έως τέσσερις) και ξεκινάει η ανταλλαγή δεδομένων.

Οι ρουτίνες ανταλλαγής δεδομένων με το AIC, συνδέονται με τις αντίστοιχες διακοπές υλικού της σειριακής πόρτας (η ρουτίνα ανάγνωσης με τη διακοπή λήψης και η ρουτίνα εγγραφής με τη διακοπή μετάδοσης). Όταν οι δύο μονάδες μετατροπής του AIC δουλεύουν συγχρονισμένα, υπάρχει μόνο μια ρουτίνα που συνδέεται με την διακοπή μετάδοσης.

5.3 Τα εργαλεία παραγωγής κώδικα

Τα βασικά εργαλεία λογισμικού, που συνοδεύουν την πλακέτα EVM C30, περιλαμβάνουν έναν assembler, έναν μεταγλωττιστή της γλώσσας C και ένα εργαλείο σύνδεσης. Αρκετά άλλα βοηθητικά προγράμματα παρέχονται, όπως το πρόγραμμα δημιουργίας βιβλιοθηκών. Επίσης, παρέχεται ένας εκσφαλματωτής (debugger), που χρησιμοποιεί το κύκλωμα εξομοίωσης της πλακέτας EVM C30, για τον έλεγχο της εκτέλεσης του προγράμματος στον επεξεργαστή.

5.3.1 Μεταγλωττιστής

Ο μεταγλωττιστής της γλώσσας C, που παρέχεται από την Texas Instruments για την οικογένεια επεξεργαστών C3x, είναι από τους πιο αποδοτικούς μεταγλωττιστές επεξεργαστών ψηφιακού σήματος [19]. Αντιμετωπίζει με επιτυχία κάποια από τα προβλήματα που έχουν οι μεταγλωττιστές συγγενικών επεξεργαστών, όπως είναι οι C2x, C5x, C54x [37]. Σε αυτό βοηθάει και η αποτελεσματική υποστήριξη διαφόρων μεθόδων διευθυνσιοδότησης από τον επεξεργαστή C3x.

Ο μεταγλωττιστής είναι πλήρως συμβατός με το πρότυπο ANSI C. Υποστηρίζει την αλληλεπίδραση κώδικα C και assembly με τρεις τρόπους. Με κλήση συναρτήσεων assembly μέσα από κώδικα C, κλήση συναρτήσεων C μέσα από κώδικα assembly και ενσωμάτωση μεμονωμένων δηλώσεων assembly μέσα στον κώδικα C, με χρήση της εντολής *asm()*. Όλοι οι ακέραιοι τύποι δεδομένων παριστάνονται με δυαδικές τιμές μεγέθους 32 bit. Έτσι όλοι οι ακέραιοι τύποι και οι τύποι χαρακτήρων (char, short, int, long και οι αντίστοιχοι unsigned) είναι ισοδύναμοι. Όσον αφορά τους αριθμούς κινητής υποδιαστολής, οι τύποι float και double είναι ισοδύναμοι (32 bit), ενώ ο τύπος long double είναι εκτεταμένης ακρίβειας (40 bit).

Ο επεξεργαστής C30, έχει την ιδιαιτερότητα να διευθυνσιοδοτεί λέξεις μνήμης μεγέθους 32 bit. Επιμέρους οκτάδες bit από αυτή τη λέξη δεν είναι προσπελάσιμες ανεξάρτητα. Έτσι για τον C30 το μέγεθος του byte είναι μια λέξη δηλαδή 32 bit. Έτσι, ο τελεστής sizeof όταν ρωτείται για το μέγεθος των τύπων δεδομένων που καταλαμβάνουν 32 bit, απαντάει 1, ενώ δίνει το μέγεθος του long double ίσο με 2. Το αποτέλεσμα αυτό δεν είναι αναμενόμενο και μερικές φορές μπορεί να δημιουργεί προβλήματα συμβατότητας του κώδικα.

Η παραγωγή κώδικα γίνεται σε τρεις φάσεις. Τη φάση της ανάλυσης (parsing), τη φάση της βελτιστοποίησης (optimising) και τη φάση της παραγωγής κώδικα. Για την επιτυχή μεταγλώττιση του κώδικα είναι απαραίτητη η χρήση μιας βιβλιοθήκης που υλοποιεί το πρότυπο ANSI C και παρέχει την συνάρτηση εκκίνησης του επεξεργαστή.

Ανάμεσα στα ορίσματα που δέχεται ο μεταγλωττιστής, μεγαλύτερο ενδιαφέρουν έχουν οι λεγόμενες run-time επιλογές, που καθορίζουν το γενικό πλαίσιο μεταγλώττισης του κώδικα. Πριν την μεταγλώττιση ενός C προγράμματος, είναι υποχρεωτική η επιλογή ενός συνόλου run-time ρυθμίσεων, που καθορίζουν και την κατάλληλη run-time βιβλιοθήκη. Αναλυτική περιγραφή των επιλογών αυτών μπορεί να βρεθεί στο [38].

'Αλλες επιλογές του μεταγλωττιστή αφορούν τη φάση της βελτιστοποίησης. Σε αυτό το στάδιο, ο μεταγλωττιστής εκτελεί βελτιώσεις γενικές (ανεξάρτητες της αρχιτεκτονικής του επεξεργαστή) και βελτιώσεις σχετικές με τον επεξεργαστή. Τα επίπεδα βελτιστοποίησης που παρέχονται είναι πέντε. Το επίπεδο 0 (ή επίπεδο καταχωρητή), το επίπεδο 1 (ή τοπικό επίπεδο), το επίπεδο 2 (ή γενικό επίπεδο), το επίπεδο 3 (ή επίπεδο αρχείου) και το επίπεδο 4 (ή επίπεδο προγράμματος). Το προτεινόμενο επίπεδο βελτιστοποίησης είναι το 4, όπου όλος ο κώδικας του προγράμματος αντιμετωπίζεται ενιαία και είναι εμφανείς οι όλες οι αλληλεξαρτήσεις. Οι επιλογές debugging και βελτιστοποίηση δεν συμβαδίζουν, γιατί η χρήση της πρώτης απενεργοποιεί τη δεύτερη. Στο [38, Appendix A] μπορεί να βρεθεί μια περιγραφή των λειτουργιών, που εκτελούνται κατά τη φάση της βελτιστοποίησης.

5.3.2 Εργαλείο σύνδεσης

Η μεταγλώττιση του κώδικα παράγει ένα αρχείο object, που είναι relocatable. Δηλαδή δεν χρησιμοποιεί απόλυτες διευθύνσεις και τα τμήματα του μπορούν να τοποθετηθούν οπουδήποτε στη μνήμη. Την εργασία αυτή αναλαμβάνει το εργαλείο σύνδεσης, σύμφωνα με οδηγίες που του δίνει ο χρήστης μέσα από ένα αρχείο εντολών.

Το αρχείο εντολών μπορεί να περιέχει όλα τα ορίσματα που παίρνει το εργαλείο σύνδεσης στη γραμμή εντολών (ονόματα object αρχείων για σύνδεση, ονόματα βιβλιοθηκών, μονοπάτια αναζήτησης βιβλιοθηκών, ορίσματα κλπ). Επιπλέον περιέχει δύο ντιρεκτίβες, την *memory*{} και την *section*{}. Η πρώτη, χρησιμοποιείται για τον ορισμό ενός χάρτη μνήμης ενώ η δεύτερη, για την κατανομή των διαφόρων τμημάτων του προγράμματος στις διαφορετικές τράπεζες μνήμης. Για να είναι αυτό δυνατό, τμήματα του προγράμματος που είναι επιθυμητό να καταλαμβάνουν διαφορετικές θέσεις

μνήμης, πρέπει να μεταγλωττιστούν σε διαφορετικά αρχεία object. Για παράδειγμα, κώδικας αρχικοποίησης και κώδικας που καταλαμβάνει μεγάλο ποσοστό του χρόνου εκτέλεσης, μπορούν να μεταγλωττιστούν σε διαφορετικά αρχεία και να τοποθετηθούν, ο πρώτος, σε εξωτερική μνήμη και ο δεύτερος σε εσωτερική.

Περισσότερα για την σύνταξη του αρχείου εντολών του εργαλείου σύνδεσης, και για τη χρήση των ντιρεκτίβων memory{} και section{}, μπορούν να βρεθούν στο [39].

Κεφάλαιο 6

Υλοποίηση

Στην παρούσα εργασία υλοποιήθηκε μια νέα πλατφόρμα εκτέλεσης για χρήση με το Real-Time Workshop. Με τη βοήθεια της πλατφόρμας αυτής μπορούμε να παράγουμε μια αυτόνομη εφαρμογή πραγματικού χρόνου για τον επεξεργαστή TMS320C30 της Texas Instruments από την περιγραφή ενός αλγόριθμου ή συστήματος επεξεργασίας σήματος στο Simulink. Για το σκοπό αυτό υποστηρίζεται η ενσωμάτωση οδηγών συσκευής για επικοινωνία με αναλογική είσοδο/έξοδο. Τα κυκλώματα αναλογικής εισόδου/εξόδου, που υποστηρίζονται, είναι τα TLC32040 και TLC32044.

Στις επόμενες παραγράφους παρουσιάζεται η υλοποίηση μας. Αρχικά θα δούμε τα διάφορα τμήματα κώδικα, που αναπτύχθηκαν για τη δημιουργία της πλατφόρμας. Στη συνέχεια, θα δούμε τις δυνατότητές της (μοντέλα που υποστηρίζονται) και τους περιορισμούς που εισάγονται από επιλογές της υλοποίησης και του υλικού. Τέλος θα παρουσιάσουμε ένα παράδειγμα χρήσης της πλατφόρμας μας. Θα ξεκινήσουμε από την ανάλυση ενός απλού μοντέλου στο Simulink, και θα δούμε τον κώδικα που παράγεται από αυτό. Θα σχολιάσουμε τις αδυναμίες και τα θετικά στοιχεία του συστήματος παραγωγής κώδικα. Τέλος, θα δούμε τον τρόπο που εκτελείται το μοντέλο αυτό στην πλατφόρμα μας (χρονισμός της εκτέλεσης).

6.1 Ο κώδικας της πλατφόρμας

6.1.1 Η βασική συνάρτηση main()

Η συνάρτηση main(), που υλοποιήσαμε, ακολουθεί το γενικό πλαίσιο που είδαμε στην παράγραφο 4.2.

Οι αρχικοποιήσεις που εκτελούνται περιλαμβάνουν:

- Την αρχικοποίηση του μοντέλου (δηλαδή της μνήμης και των δομών δεδομένων που χρησιμοποιούνται).
- Την αρχικοποίηση του επεξεργαστή (αρχικοποίηση του διαδρόμου, καθαρισμός της cache και των απαραίτητων καταχωρητών ή πεδίων τους)
- Την αρχικοποίηση και ενεργοποίηση του μετρητή που χρησιμοποιείται κατά περίπτωση για τον έλεγχο της εκτέλεσης του μοντέλου (βλ. παράγραφο 6.1.4).
- Την ενεργοποίηση των απαραίτητων διακοπών υλικού.

Στη συνέχεια η συνάρτηση main(), εκτελεί μια ατέρμονη επανάληψη, που μπορεί να διακοπεί μόνο από μια συνθήκη σφάλματος εκτέλεσης. Το σώμα της επανάληψης αυτής είναι κενό ή μπορεί να περιέχει κάποιες εντολές που εκτελούνται στο παρασκήνιο, όταν ο επεξεργαστής δεν είναι απασχολημένος να εκτελεί το μοντέλο. Ο κώδικας του μοντέλου δεν εκτελείται από την main() αλλά από μια ρουτίνα χειρισμού διακοπών υλικού, όπως θα δούμε στην παράγραφο 6.1.4. Η εκτέλεση του, δε, ξεκινάει με την ενεργοποίηση των αντίστοιχων διακοπών.

Συνθήκες σφάλματος της εκτέλεσης του μοντέλου, είναι η μη εκτέλεση του σε πραγματικό χρόνο, η λήψη μιας μη αναμενόμενης διακοπής υλικού ή η αδυναμία δέσμευσης μνήμης (για δυναμική δέσμευση).

6.1.2 Το αρχείο system target

Το αρχείο system target μπορεί να χρησιμοποιηθεί για να οριστούν παράμετροι, που εμφανίζονται στο παράθυρο διαλόγου Real-Time Workshop, σχετικές με την πλατφόρμα εκτέλεσης και γενικές μεταβλητές, που χρησιμοποιούνται από την διαδικασία παραγωγής κώδικα. Επίσης μπορεί να χρησιμοποιηθεί για να παραχθούν διάφορα αρχεία ή τμήματα κώδικα που εξυπηρετούν την δημιουργία της εφαρμογής.

Η δήλωση των παραμέτρων περιλαμβάνει εκτός από την πληροφορία που παρουσιάζεται στο παράθυρο διαλόγου, ονόματα μεταβλητών για την αποθήκευση κάθε παραμέτρου. Οι μεταβλητές αυτές μπορούν να χρησιμοποιηθούν από την διαδικασία παραγωγής κώδικα ή/και την μεταγλώττιση και προωθούνται ανάλογα από την διαδικασία MAKE_RTW. Οι παράμετροι που υποστηρίζει η πλατφόρμα μας θα παρουσιαστούν στην παράγραφο 6.3.

Όσον αφορά τις μεταβλητές που δηλώνονται σε αυτό το αρχείο, άλλες είναι υποχρεωτικό να παρέχονται από κάθε πλατφόρμα εκτέλεσης ενώ άλλες είναι προαιρετικές. Οι υποχρεωτικές είναι οι εξής:

- *CodeFormat* : Επιλέγει την τυποποίηση που θα χρησιμοποιηθεί. Στην περίπτωση μας έχει την τιμή *Embedded-C*.
- *TargetType* : Παίρνει την τιμή *RT* ή *NRT* όταν ο παραγόμενος κώδικας εκτελείται σε πραγματικό χρόνο ή σε χρόνο προσομοίωσης αντίστοιχα.

- *Language* : Μπορεί να πάρει την τιμή *C* ή *Ada* ανάλογα με τη γλώσσα στην οποία θα παραχθεί ο κώδικας.

Από τις προαιρετικές, χρήσιμες για την πλατφόρμα μας ήταν οι εξής:

- *DSP32* : Έχει την τιμή 1 αν η πλατφόρμα εκτέλεσης είναι επεξεργαστής ψηφιακού σήματος.

- *MaxStackSize* : Ελέγχει το συνολικό μέγεθος της στοίβας που χρησιμοποιείται από όλες τις συναρτήσεις του παραγόμενου κώδικα. Στην περίπτωση που δεν καλούνται συναρτήσεις στον κώδικα κάποιου block, η μόνη συνάρτηση που ελέγχεται είναι η MODEL_step(). Έχει την τιμή *255* εξαιτίας του περιορισμού που υπάρχει στη χρήση του indirect addressing mode.

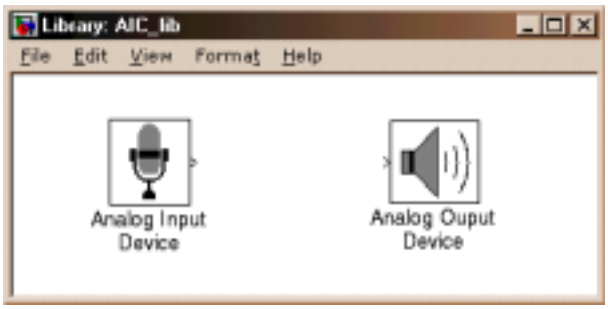
- *MaxStackVariableSize* : Ελέγχει το μέγιστο μέγεθος της κάθε μεταβλητής που δεσμεύει χώρο στη στοίβα. Αν η μεταβλητή απαιτεί περισσότερο χώροι, τότε δεσμεύεται γενικά. Έχει την τιμή *64*, ώστε να επιτρέπεται η τοπική δέσμευση έως και 4 μεταβλητών μέγιστου μεγέθους, νούμερο που από παρατηρήσεις είδαμε ότι επαρκεί.

Στην υλοποίηση μας, χρησιμοποιήσαμε το αρχείο αυτό για να παράγουμε κώδικα που εξαρτάται από την πλατφόρμα, για λόγους βελτιστοποίησης του τελικού εκτελέσιμου. Συγκεκριμένα, παράγουμε τον assembly κώδικα που ορίζει τις ρουτίνες χειρισμού των διακοπών υλικού με βάση το μοντέλο μας. Έτσι, αν, για παράδειγμα, δεν χρησιμοποιηθεί το αναλογικό κύκλωμα εισόδου/εξόδου, δεν θα οριστούν οι αντίστοιχες ρουτίνες ανάγνωσης και εγγραφής. Επίσης, παράγουμε το αρχείο εντολών, που παίρνει είσοδο το εργαλείο σύνδεσης (βλ. παράγραφο 5.3). Με αυτό τον τρόπο, έχουμε τη δυνατότητα να κατανείμουμε τα διάφορα τμήματα του κώδικα σε διαφορετικά τμήματα της μνήμες. Έτσι, ο κώδικας αρχικοποίησης πηγαίνει στην εξωτερική μνήμη ενώ ο κώδικας του μοντέλου που εκτελείται επαναληπτικά πηγαίνει στην εσωτερική μνήμη.

6.1.3 Οδηγοί συσκευών

Η πλατφόρμα μας, παρέχει δύο οδηγούς συσκευής για την επικοινωνία με το αναλογικό κύκλωμα εισόδου/εξόδου. Ο ένας χρησιμοποιείται για την ανάγνωση δειγμάτων από το κύκλωμα ενώ ο άλλος για την εγγραφή δειγμάτων σε αυτό. Ο κάθε οδηγός συσκευής περιλαμβάνει ένα Simulink block υλοποιημένο σε S-function, το σχετικό παράθυρο διαλόγου για εισαγωγή παραμέτρων, την TLC αναπαράσταση του block και μια ρουτίνα χειρισμού των διακοπών υλικού.

Η χρήση των οδηγών συσκευής είναι προαιρετική για την δημιουργία εκτελέσιμου για την πλατφόρμα μας. Ωστόσο, πρέπει να χρησιμοποιηθούν αν είναι επιθυμητή η επικοινωνία με το αναλογικό κύκλωμα εισόδου/εξόδου. Στο Σχήμα 6.1 φαίνονται τα δύο blocks εισόδου/εξόδου.



Σχήμα 6.1: Η βιβλιοθήκη αναλογικής εισόδου/εξόδου

6.1.3.1 Παράμετροι οδηγών συσκευής

Ο οδηγός συσκευής αναλογικής εισόδου υποστηρίζει τις εξής παραμέτρους (Σχήμα 6.2):

- *Ρυθμός δειγματοληψίας εισόδου.* Η παράμετρος αυτή καθορίζει τον ρυθμό με τον οποίο διαβάζονται δείγματα από την αναλογική είσοδο. Η τιμές που μπορεί να πάρει είναι συγκεκριμένες και καθορίζονται από το αναλογικό κύκλωμα.

- *Εύρος τιμών τάσης εισόδου.* Η παράμετρος αυτή προσδιορίζει το εύρος τιμών της τάσης εισόδου, που εξαρτάται από τη συσκευή εισόδου που χρησιμοποιούμε (π.χ. μικρόφωνο). Χρησιμοποιείται για να προσαρμόσει κατάλληλα την ενίσχυση της εισόδου πριν την δειγματοληψία, ώστε να αποφύγουμε φαινόμενα ψαλιδισμού ή απώλειας ακρίβειας.

- *Χρήση υψιπερατού φίλτρου.* Το αναλογικό κύκλωμα παρέχει την επιλογή της χρήσης ή όχι ενός αναλογικού υψιπερατού φίλτρου με συχνότητα αποκοπής τα 150 Hz.

- *Κέρδος εισόδου.* Η παράμετρος αυτή χρησιμοποιείται για την ενίσχυση ή την αποδυνάμωση της έντασης της δειγματοληπτημένης εισόδου κατά περιπτώσεις. Η χρήση της είναι αποδοτικότερη έναντι της χρήσης κάποιου *Gain* block μετά το block της εισόδου.

- *Αριθμός δειγμάτων ανά πλαίσιο.* Η παράμετρος αυτή επιτρέπει την ομαδοποίηση δειγμάτων εισόδου σε ένα πλαίσιο για περαιτέρω επεξεργασία. Και πάλι η χρήση της είναι αποδοτικότερη έναντι της χρήσης ενός *Buffer* block μετά το block της εισόδου.

- *Λειτουργία loop back.* Το αναλογικό κύκλωμα παρέχει μια λειτουργία loopback, κατά την οποία η έξοδος συνδέεται εσωτερικά με την είσοδο. Η λειτουργία αυτή επιτρέπει τον έλεγχο της ακρίβειας του αναλογικού κυκλώματος.

Οι παράμετροι της αναλογικής εξόδου (Σχήμα 6.3) είναι παρόμοιες, με τη διαφορά ότι υποστηρίζεται η χρήση ενός φίλτρου διόρθωσης αντί για το υψιπερατό φίλτρο που

Σχήμα 6.2: Το παράθυρο διαλόγου του block αναλογικής εισόδου

είδαμε στην είσοδο. Η παράμετρος του αριθμού δειγμάτων ανά πλαίσιο απουσιάζει γιατί κληρονομείται από το σήμα εισόδου. Η τιμή της πρέπει να είναι η ίδια με αυτή της αναλογικής εισόδου. Η παράμετρος του ρυθμού δειγματοληψίας ορίζει τον ρυθμό με τον οποίο γράφονται δείγματα στην έξοδο, και πρέπει να είναι ίση με την αντίστοιχη της εισόδου, στην παρούσα υλοποίηση.

Σχήμα 6.3: Το παράθυρο διαλόγου του block αναλογικής εξόδου

Η τιμή της παραμέτρου που ορίζει το μέγεθος του πλαισίου επεξεργασίας, σε συνδυασμό με τον ρυθμό δειγματοληψίας της εισόδου, καθορίζει τον ρυθμό εκτέλεσης του κώδικα του μοντέλου (ρυθμός επεξεργασίας). Όταν η παράμετρος αυτή έχει την τιμή 1, ο ρυθμός επεξεργασίας ταυτίζεται με τον ρυθμό δειγματοληψίας. Διαφορετικά ισούται με τον ρυθμό δειγματοληψίας δια την τιμή της παραμέτρου.

Το μέγεθος του πλαισίου επεξεργασίας, επηρεάζει τόσο το μέγεθος του κώδικα όσο και την δυνατότητα εκτέλεσης του σε πραγματικό χρόνο. Η επιλογή της τιμής του

εξαρτάται από την πολυπλοκότητα του μοντέλου. Π.χ. σε ένα μοντέλο με πολλά επίπεδα υπολογισμών είναι προτιμότερη η επιλογή ενός μικρού μεγέθους πλαισίου.

6.1.3.2 TLC αναπαράσταση οδηγών συσκευής

Η TLC αναπαράσταση των blocks εισόδου/εξόδου, που χρησιμοποιείται για την παραγωγή κώδικα σχετικού με το αναλογικό κύκλωμα, περιλαμβάνει λειτουργίες που εκτελούνται κατά την αρχικοποίηση του μοντέλου (κώδικας αρχικοποίησης) και λειτουργίες που εκτελούνται σε κάθε επανάληψη της εκτέλεσής του (κώδικας εκτέλεσης).

Ο κώδικας αρχικοποίησης, που παράγεται, φροντίζει για την αρχικοποίηση του αναλογικού κυκλώματος και της μνήμης που χρησιμοποιείται για την αποθήκευση της εισόδου/εξόδου. Η αρχικοποίηση του κυκλώματος γίνεται με βάση τις παραμέτρους των blocks εισόδου/εξόδου και περιλαμβάνει την ενεργοποίηση του μετρητή που είναι απαραίτητος για τη λειτουργία του κυκλώματος, της σειριακής πόρτας μέσω της οποίας επικοινωνεί το κύκλωμα με τον επεξεργαστή και την ενεργοποίηση των κατάλληλων διακοπών υλικού.

Ο κώδικας εκτέλεσης, που παράγεται από το block εισόδου, φροντίζει για τη μεταβίβαση των δειγμάτων της εισόδου, στο υπόλοιπο του μοντέλου για επεξεργασία. Αντίστοιχα, ο κώδικας που παράγεται από το block εξόδου, μεταβιβάζει τα επεξεργασμένα δείγματα στην έξοδο. Ο όρος μεταβίβαση δεν περιλαμβάνει την ίδια την πράξη της ανάγνωσης ή εγγραφής δειγμάτων από/προς το αναλογικό κύκλωμα, που είναι ευθύνη μιας ρουτίνας ISR. Αφορά μόνο την αντιγραφή μνήμης που χρησιμοποιείται από τη ρουτίνα ISR, σε μνήμη που χρησιμοποιείται από το μοντέλο και αντίστροφα. Η αντιγραφή αυτή επιβάλλεται από την ανάγκη δέσμευσης της μνήμης του μοντέλου στατικά (λόγω χρήσης της Embedded τυποποίησης) και από την ανάγκη ανάγνωσης των δειγμάτων ενός νέου πλαισίου πριν ολοκληρωθεί η επεξεργασία του προηγούμενου.

Η μνήμη, που χρησιμοποιείται από την ISR ρουτίνα εισόδου/εξόδου, μπορεί να δεσμευτεί ανεξάρτητα από την υπόλοιπη μνήμη του μοντέλου είτε στατικά είτε δυναμικά. Στη υλοποίηση μας επιλέχθηκε η στατική δέσμευση, ώστε να είναι γνωστό το συνολικό μέγεθος της απαιτούμενης μνήμης πριν την εκτέλεση του μοντέλου. Αποφεύγονται έτσι συνθήκες σφάλματος που οφείλονται στην μη επάρκεια της μνήμης ή του μεγέθους του σωρού (heap).

Η ρουτίνα ISR, που είναι υπεύθυνη για την επικοινωνία με το αναλογικό κύκλωμα περιγράφεται στην επόμενη παράγραφο.

6.1.4 Ρουτίνες χειρισμού διακοπών υλικού

Η πλατφόρμα μας, χρειάστηκε δύο ρουτίνες χειρισμού διακοπών υλικού (ISRs). Η πρώτη, εκτελεί την επικοινωνία με το αναλογικό κύκλωμα εισόδου/εξόδου (ISR εισόδου/εξόδου) και η δεύτερη επεξεργάζεται τα δείγματα αυτά σύμφωνα με το εκάστοτε μοντέλο (ISR επεξεργασίας).

Είδαμε ήδη στο Κεφάλαιο 5, πως εκτελείται η επικοινωνία του AIC με τον επεξεργαστή. Είδαμε, επίσης, ότι η ISR εισόδου/εξόδου συνδέεται με την διακοπή μετάδοσης της σειριακής πόρτας.

Όταν ένα πλαίσιο εισόδου έχει διαβαστεί, ενεργοποιείται η ISR ρουτίνα επεξεργασίας. Η εκτέλεση της ρουτίνας αυτής, πρέπει να γίνεται με ρυθμό που ισούται με τον ρυθμό δειγματοληψίας δια τον αριθμό των δειγμάτων ανά πλαίσιο. Η ενεργοποίηση της, μπορεί να γίνει με δύο τρόπους. Σύμφωνα με τον πρώτο τρόπο, ενεργοποιείται ανεξάρτητα από την ISR ρουτίνα εισόδου/εξόδου, με τη βοήθεια ενός χρονόμετρου του επεξεργαστή, που έχει ρυθμιστεί κατάλληλα. Το μειονέκτημα της μεθόδου αυτής είναι ότι είναι δύσκολο να επιτευχθεί απόλυτος συγχρονισμός των δύο ρουτινών ISR, που είναι απαραίτητος για την εκτέλεση του μοντέλου σε πραγματικό χρόνο. Έτσι, συχνά υπάρχει σφάλμα εκτέλεσης σε πραγματικό χρόνο σε πολύπλοκα multi rate μοντέλα.

Ένας άλλος τρόπος ενεργοποίησης της ρουτίνας αυτής, χωρίς προβλήματα συγχρονισμού, είναι η κλήση της από την ρουτίνα ISR εισόδου/εξόδου όταν έχει διαβαστεί ένα πλαίσιο εισόδου. Η μέθοδος αυτή εκτελεί το μοντέλο σε πραγματικό χρόνο, γιατί η περίοδος επανάληψης της εκτέλεσης του μοντέλου είναι ακριβώς πολλαπλάσια της περιόδου δειγματοληψίας. Η ρουτίνα επεξεργασίας συσχετίζεται και σε αυτή την περίπτωση με την διακοπή του χρονόμετρου, που όμως παράγεται από το λογισμικό. Το ίδιο το χρονόμετρο απενεργοποιείται.

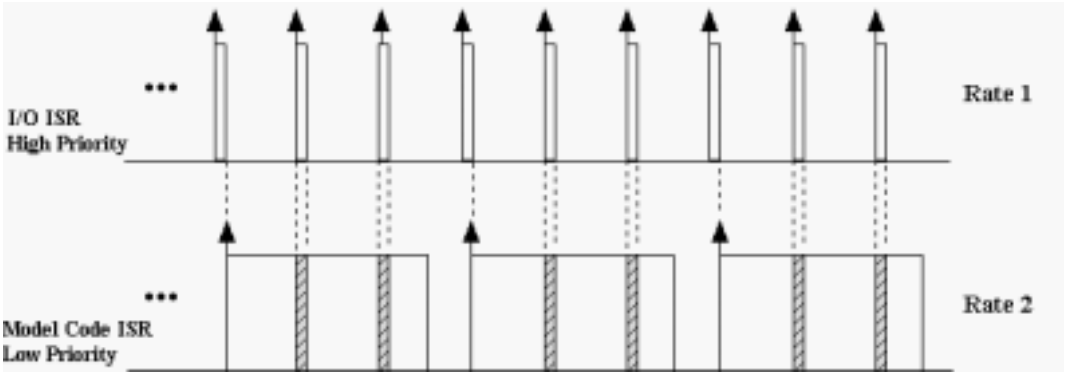
Το μειονέκτημα της μεθόδου αυτής είναι ότι δεν μπορεί να χρησιμοποιηθεί για multi rate μοντέλα, όπου ο χρόνος δειγματοληψίας της αναλογικής εισόδου/εξόδου δεν είναι ο θεμελιώδης του συστήματος. Από την άλλη, έχει σταθερή συμπεριφορά γιατί δεν απαιτεί εξωτερικό συγχρονισμό των δύο ρουτινών, γι' αυτό και τελικά επιλέχθηκε η χρήση της έναντι της πρώτης μεθόδου.

Η ρουτίνα επεξεργασίας, εκτελείται από μια ISR που έχει χαμηλότερη προτεραιότητα από την ISR της ρουτίνας εισόδου/εξόδου (βλ. Πίνακα 5.2). Αυτό είναι απαραίτητο για να μπορεί να διακόπτεται από την ISR εισόδου/εξόδου κάθε φορά που υπάρχει ένα δείγμα για ανάγνωση/εγγραφή. Μόλις ολοκληρωθεί η είσοδος/έξοδος ο έλεγχος επιστρέφει στην ISR επεξεργασίας. Η εκτέλεση αυτή στο χρόνο φαίνεται στο Σχήμα 6.4. Σε περίπτωση που το μοντέλο είναι multitasking, ισχύουν επιπλέον και όσα αναφέρθηκαν στην παράγραφο 4.3.2.

Η ενεργοποίηση των διακοπών υλικού γίνεται για τις μεν διακοπές της σειριακής πόρτας στην ρουτίνα αρχικοποίησης του AIC, ενώ για τις δε διακοπές του μετρητή στην main().

6.1.4.1 Μοντέλα χωρίς αναλογική είσοδο/έξοδο

Σε μοντέλα χωρίς αναλογική είσοδο/έξοδο, ο κώδικας που αφορά τη λειτουργία του AIC δεν συνδέεται με τον υπόλοιπο κώδικα του μοντέλου. Σε αυτή την περίπτωση, έχουμε μόνο μια ISR ρουτίνα (αυτή που εκτελεί τον κώδικα του μοντέλου) και η οποία καλείται με τη βοήθεια ενός χρονόμετρου. Το χρονόμετρο ρυθμίζεται στη συνάρτηση



Σχήμα 6.4: Η εκτέλεση των ISR ρουτινών εισόδου/εξόδου και επεξεργασίας

Στο σχήμα φαίνεται ο χρονισμός εκτέλεσης των ρουτίνων ISR. Το μέγεθος πλαισίου είναι 3 και η ρουτίνα επεξεργασίας ενεργοποιείται στο τέλος κάθε τρίτης ρουτίνας εισόδου/εξόδου. Ο χρόνος εκτέλεσης της είναι μεγαλύτερος από την περίοδο δειγματοληψίας και έτσι διακόπτεται για την εκτέλεση I/O και επανέρχεται.

main(), όπου και ενεργοποιείται η σχετική διακοπή υλικού.

Σε αυτά τα μοντέλα, αν η συχνότητα δειγματοληψίας που ορίζεται από τον χρήστη, δεν είναι ακριβές υποπολλαπλάσιο της συχνότητας των 7.5MHz[1], αυτή στρογγυλοποιείται στην κοντινότερη συχνότητα, που πληρεί αυτή την προϋπόθεση.

6.1.5 Επεμβάσεις στο Real-Time Workshop

Για να είναι δυνατή η παραγωγή σωστού κώδικα για τον επεξεργαστή TMS320C30, χρειάστηκε να γίνουν ορισμένες επεμβάσεις σε αρχεία του πυρήνα του Real-Time Workshop. Οι επεμβάσεις αυτές αφορούσαν την υποστήριξη των τύπων δεδομένων του συγκεκριμένου επεξεργαστή και της ιδιαίτερης συμπεριφοράς του τελεστή sizeof (βλ. παράγραφο 5.3).

'Αλλη μια επέμβαση που έγινε, αφορά τον τύπο δεδομένων που χρησιμοποιείται εξ ορισμού κατά την παραγωγή κώδικα για τις δηλώσεις μεταβλητών. Το Real-Time Workshop χρησιμοποιεί τον τύπο δεδομένων με τη μεγαλύτερη ακρίβεια (long double για την πλατφόρμα μας). Η επιλογή αυτή δεν είναι αποδεκτή για λόγους που παρουσιάζονται στη συνέχεια και γι'αυτό αντικαταστάθηκε από τον τύπο float. Η αλλαγή αυτή κρίθηκε απαραίτητη γιατί ο τύπος long double, καταλαμβάνει στην αρχιτεκτονική του επεξεργαστή μας διπλάσιο χώρο μνήμης από ότι οι τύποι float και double. Επιπλέον, η χρήση του επιβαρύνει σημαντικά τον χρόνο εκτέλεσης υπολογισμών καθώς δεν υπάρχουν ενσωματωμένες (build-in) εντολές για την εκτέλεση πολλαπλασιασμών εκτεταμένης ακρίβειας. Οι ρουτίνες που χρησιμοποιούνται για αυτό το σκοπό έχουν χρόνο εκτέλεσης πολύ μεγαλύτερο του ενός κύκλου ρολογιού. Τέλος, και ο χειρισμός αυτού του τύπου δεδομένων είναι δαπανηρός, καθώς απαιτούνται δύο προσπελάσεις μνήμης για την ανάγνωση και εγγραφή του. Έτσι, πρέπει να

[1]Η μέγιστη συχνότητα που παράγει το χρονόμετρο

χρησιμοποιείται προσεκτικά και μόνο όταν η ακρίβεια που παρέχει είναι απαραίτητη.

6.2 Δυνατότητες και περιορισμοί

Τα blocks, που μπορούν να χρησιμοποιηθούν για τη δημιουργία μοντέλων, με στόχο την εκτέλεση τους στην πλατφόρμα μας, προέρχονται από τη βασική βιβλιοθήκη του Simulink και τη βιβλιοθήκη Ψηφιακής Επεξεργασίας Σήματος (DSP). Υποστηρίζονται όλα τα blocks εκτός από τις παρακάτω εξαιρέσεις:

- Συνεχή blocks. Τα blocks αυτά δεν χρησιμοποιούνται στην μοντελοποίηση διακριτών συστημάτων (όπως είναι τα συστήματα ψηφιακής επεξεργασίας σήματος) και έτσι δεν υποστηρίζονται από την διαδικασία παραγωγής κώδικα. Το γεγονός αυτό δεν είναι περιοριστικό καθώς οι λειτουργίες τους παρέχονται από ισοδύναμα διακριτά blocks.

- S-function blocks χωρίς TLC αναπαράσταση. Στο θέμα αυτό έχουμε ήδη αναφερθεί στην παράγραφο 4.1.1. Ο περιορισμός αυτός δεν είναι κρίσιμος, γιατί μόνο μια μικρή μειοψηφία blocks από την βιβλιοθήκη DSP δεν πληρούν αυτή την προϋπόθεση. Παραδείγματα είναι το block που εκτελεί αντίστροφο μετασχηματισμό Fourier, το block Levinson-Doublin κλπ. Εξάλλου, σε νεώτερες εκδόσεις της βιβλιοθήκης αναμένεται κάθε S-function block να έχει ισοδύναμη TLC αναπαράσταση.

- Blocks της βασικής βιβλιοθήκης του Simulink, που χρησιμοποιούν απόλυτο χρόνο για να υπολογίσουν τις εξόδους τους. Τα blocks αυτά δεν υποστηρίζονται από μοντέλα που είναι σχεδιασμένα για επ' άπειρον εκτέλεση γιατί οδηγούν σε σφάλμα υπερχείλισης (overflow) μετρητή. Παραδείγματα, είναι τα *Step*, *Ramp*, *SineWave*, *Chirp Signal*, *Pulse Generator* κλπ. Μια πλήρης λίστα μπορεί να βρεθεί στο [31, Appendix A]. Κάποια από τα blocks αυτά παρέχονται από την βιβλιοθήκη DSP, σε υλοποιήσεις που δεν έχουν τον παραπάνω περιορισμό (π.χ. *Ramp*, *SineWave*, *Chirp*).

- Blocks που καλούν MATLABκώδικα όπως είναι τα *MATLAB Fcn blocks* και *MATLAB S-functions.*

Εκτός από τα παραπάνω blocks, στο μοντέλο μπορούν να ενσωματωθούν και ρουτίνες του χρήστη σαν S-function blocks, αφού προσαρμοστούν στο S-function API και γραφεί η TLC αναπαράσταση τους. 'Αλλος ένας τρόπος με τον οποίο ο χρήστης μπορεί να προσθέσει τμήματα κώδικα στο παραγόμενο μοντέλο, είναι με τη χρήση των ειδικών blocks του Real-Time Workshop *Code Modules.* Η σωστή χρήση τους, προϋποθέτει κατανόηση των τμημάτων κώδικα που παράγονται και των αλληλεξαρτήσεων τους.

Για τη σωστή εκτέλεση του μοντέλου σε πραγματικό χρόνο, είναι απαραίτητη η χρήση της μεθόδου επίλυσης σταθερού βήματος. Η επιλογή της παραμέτρου singletasking/multitasking, μπορεί να γίνει από το χρήστη ανάλογα με τον τύπο του

μοντέλου του (singlerate, multirate) και ανάλογα με τον τρόπο που επιθυμεί να γίνει η εκτέλεση του (παράγραφος 4.3.2). Υπενθυμίζουμε εδώ, ότι η επιλογή multitasking σε ένα singlerate μοντέλο δεν είναι αποδεκτή.

Τα μοντέλα που υποστηρίζονται είναι τα singlerate και από τα multirate αυτά στα οποία ο χρόνος δειγματοληψίας του αναλογικού κυκλώματος είναι ο θεμελιώδης του συστήματος (παράγραφος 6.1.4). 'Αρα, δεν υποστηρίζονται multirate μοντέλα, όπου ο ρυθμός δειγματοληψίας του AIC είτε δεν είναι ο μικρότερος είτε είναι ο μικρότερος αλλά δεν είναι θεμελιώδης. Η δυνατότητα εκτέλεσης ενός συγκεκριμένου μοντέλου που εμπίπτει σε μια από τις δύο πρώτες κατηγορίες, εξαρτάται επιπλέον από την πολυπλοκότητα των υπολογισμών.

6.3 Παράμετροι παραγωγής κώδικα

Η υλοποίηση μας, προσφέρει κάποιες επιλογές ελέγχου της διαδικασίας παραγωγής κώδικα σχετικές με την πλατφόρμα υλοποίησης. Αυτές χωρίζονται σε δύο κατηγορίες: επιλογές μεταγλώττισης και επιλογές σύνδεσης και εκτέλεσης.

6.3.1 Επιλογές μεταγλώττισης

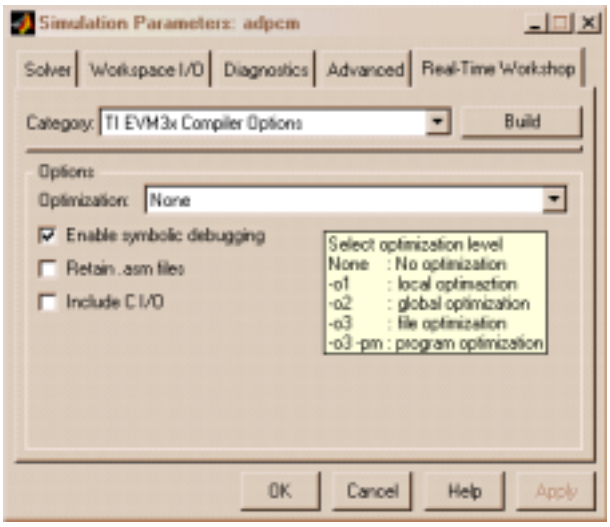
Στο Σχήμα 6.5 φαίνεται το παράθυρο διαλόγου με τις επιλογές μεταγλώττισης. Αυτές περιλαμβάνουν:

- Την επιλογή επιπέδου βελτιστοποίησης. Πιθανές επιλογές είναι τα επίπεδα 0 έως 3. Το τελευταίο επίπεδο (βελτιστοποίηση προγράμματος) δεν υποστηρίζεται προς το παρόν. Όταν επιλέγεται κάποιο επίπεδο βελτιστοποίησης διαφορετικό του 0, πρέπει να απενεργοποιείται η επιλογή debugging (βλ. και παράγραφο 5.3).

- Την ενεργοποίηση της δυνατότητας debugging. Με την επιλογή αυτή ο assembly κώδικας που παράγεται διατηρεί επιπλέον πληροφορία (όπως το symbol table) που επιτρέπει την εκτέλεση του μέσα από ένα πρόγραμμα εκσφαλμάτωσης.

- Την διατήρηση του παραγόμενου assembly κώδικα. Φυσιολογικά, μετά τη δημιουργία των αρχείων object, τα αρχεία assembly κώδικα απομακρύνονται. Η διατήρηση τους μπορεί να είναι χρήσιμη για την μελέτη του παραγόμενου κώδικα.

- Την χρήση C ρουτινών εισόδου/εξόδου. Η run-time βιβλιοθήκη του μεταγλωττιστή περιλαμβάνει όλες τις ορισμένες από το πρότυπο ANSI C ρουτίνες I/O. Οι ρουτίνες αυτές αλληλεπιδρούν με ένα τερματικό στον προσωπικό υπολογιστή, που φιλοξενεί την πλακέτα EVM C30, και έτσι μπορούν να χρησιμοποιηθούν για τον έλεγχο της εκτέλεσης του μοντέλου. Ωστόσο, η χρήση τους διπλασιάζει σχεδόν το μέγεθος του παραγόμενου κώδικα και πολλές φορές δημιουργεί προβλήματα εκτέλεσης του μοντέλου. Για το λόγο αυτό, η σχετική επιλογή πρέπει να χρησιμοποιείται προσεκτικά.



Σχήμα 6.5: Το παράθυρο διαλόγου με τις επιλογές μεταγλώττισης

6.3.2 Επιλογές σύνδεσης και εκτέλεσης

Στο Σχήμα 6.6 φαίνεται το παράθυρο διαλόγου με τις επιλογές σύνδεσης και εκτέλεσης. Αυτές περιλαμβάνουν:

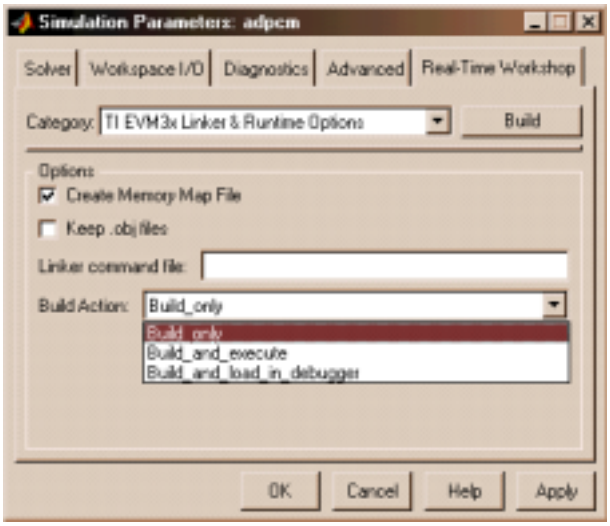
- Τη δυνατότητα δημιουργίας ενός αρχείου αποτύπωσης του χάρτη μνήμης (memory map file). Το αρχείο αυτό περιέχει πληροφορία σχετικά με την τοποθέτηση στη μνήμη των διαφόρων τμημάτων του προγράμματος (δεδομένα και πρόγραμμα ανά αρχείο object). Μπορεί να χρησιμοποιηθεί για την εξαγωγή συμπερασμάτων σχετικών με το μέγεθος του παραγόμενου κώδικα και τη δυνατότητα κατανομής του στη μνήμη με ένα πιο βέλτιστο τρόπο.

- Τη δυνατότητα διατήρησης των αρχείων object, που φυσιολογικά απομακρύνονται μετά τη δημιουργία του εκτελέσιμου. Η επιλογή αυτή, μπορεί να χρησιμοποιηθεί για τη μείωση του χρόνου μεταγλώττισης του μοντέλου.

- Τον ορισμό ενός αρχείου εντολών για το εργαλείο σύνδεσης. Με τη βοήθεια της επιλογής αυτής ο χρήστης μπορεί να ορίσει ένα χάρτη μνήμης διαφορετικό από αυτόν που παρέχεται. Η επιλογή αυτή είναι χρήσιμη, όταν παράγεται κώδικας για μια πλατφόρμα διαφορετική από την EVM C30 στο μέγεθος της μνήμης. Επίσης μπορεί να χρησιμοποιηθεί για την προσαρμογή του παρεχόμενου αρχείου εντολών στο συγκεκριμένο μοντέλο. Σε συνδυασμό με την επιλογή αυτή παρέχεται στο χρήστη ένα πρότυπο αρχείο εντολών, που περιέχει τα απαραίτητα ορίσματα, για τη σύνδεση του προγράμματος.

- Την επιλογή του σταδίου στο οποίο σταματάει η διαδικασία παραγωγής κώδικα και που μπορεί να είναι η δημιουργία του εκτελέσιμου, το ‘‘φόρτωμα’’ του στην

πλακέτα EVM C30 για εκτέλεση ή το ‘‘φόρτωμα’’ του στον εκσφαλματωτή για εκτέλεση.



Σχήμα 6.6: Το παράθυρο διαλόγου με τις επιλογές σύνδεσης και εκτέλεσης

6.4 Παραδείγματα

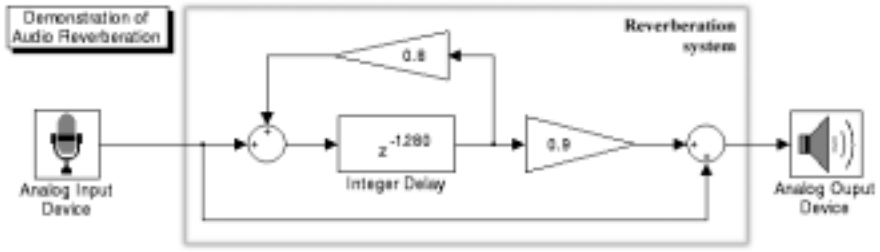
Το εργαλείο που αναπτύξαμε, μπορεί να χρησιμοποιηθεί για την υλοποίηση συστημάτων από διάφορες περιοχές εφαρμογών. Στα παραδείγματα περιλαμβάνονται συστήματα προσθήκης ηχητικών εφέ (π.χ. reverberation και flanging), FIR και IIR φίλτρα, μετασχηματισμοί (fourier, discrete cosine), συστήματα delta modulation κλπ. Τα συστήματα που αναφέρθηκαν εδώ είναι γενικά μικρής πολυπλοκότητας. Παρόλα αυτά, η ποικιλία των συστημάτων που μπορούν να υλοποιηθούν είναι πολύ ευρύτερη και περιορίζεται μόνο από την πολυπλοκότητα και τα χαρακτηριστικά εισόδου / εξόδου του συστήματος.

Παρακάτω ακολουθούν δύο παραδείγματα χρήσης του εργαλείου μας. Στο πρώτο παρουσιάζεται η υλοποίηση ενός συστήματος που εισάγει ηχώ σε ένα ακουστικό σήμα, από την κατηγορία συστημάτων προσθήκης ακουστικών εφέ, ενώ στο δεύτερο παρουσιάζεται ένα τηλεπικοινωνιακό σύστημα και συγκεκριμένα ένα σύστημα που υλοποιεί διαμόρφωση και αποδιαμόρφωση τύπου adaptive delta pulse code (ADPCM).

6.4.1 Σύστημα εισαγωγής ηχούς

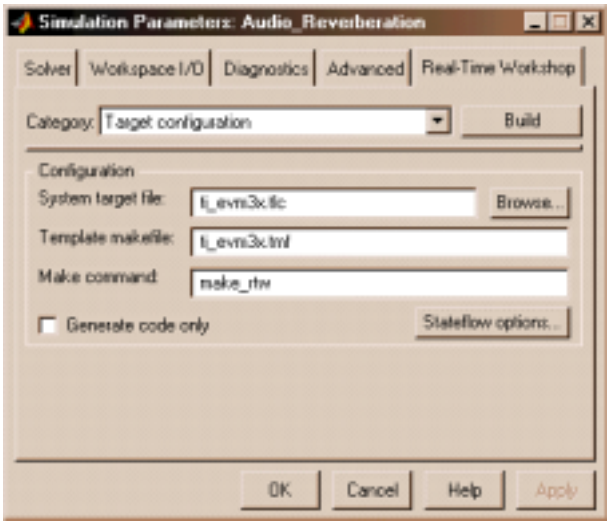
Στόχος του παραδείγματος αυτού είναι η αναπαραγωγή του σήματος εισόδου στην έξοδο μαζί με ηχώ. Ο αλγόριθμος εισαγωγής ηχούς, πρέπει πρώτα να σχεδιαστεί και επαληθευτεί στο Simulink. Για τη σχεδίαση, μπορούν να χρησιμοποιηθούν blocks

από τη βασική βιβλιοθήκη του Simulink και από το DSP blockset. Για τη σωστή μοντελοποίηση ενός συστήματος επεξεργασίας σήματος στο Simulink, πρέπει κανείς να συμβουλευθεί τα [28, 29]. Στον αλγόριθμο εισαγωγής ηχούς, προσθέτουμε στη συνέχεια τα blocks αναλογικής εισόδου / εξόδου και διαμορφώνουμε κατάλληλα τις παραμέτρους του συστήματος. Οι επιλογές μας περιλαμβάνουν τον ρυθμό δειγματοληψίας, το μέγεθος της καθυστέρησης σε δείγματα και το μέγεθος πλαισίου. Στο Σχήμα 6.7, βλέπουμε το μοντέλο του συστήματος στο Simulink.



Σχήμα 6.7: Μοντέλο συστήματος εισαγωγής ηχούς στο Simulink

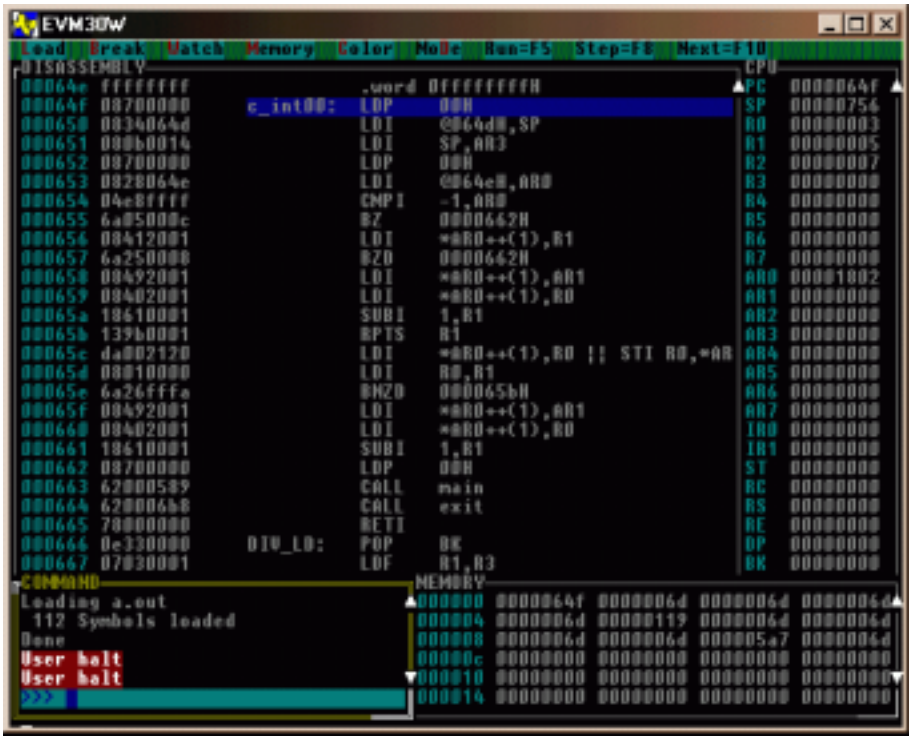
Όταν είμαστε έτοιμοι να παράγουμε κώδικα χαμηλού επιπέδου για το σύστημα που σχεδιάσαμε (αφού δηλαδή έχουμε επαληθεύσει με προσωμοίωση την ορθότητα του), επιλέγουμε από το κύριο μενού επιλογών του RTW (Σχήμα 6.8), την πλατφόρμα TI EVM C30. Ένα νέο μενού επιλογών, (που είδαμε αναλυτικά στο Κεφάλαιο 6), σχετικό με την πλατφόρμα αυτή, γίνεται τώρα διαθέσιμο. Θέτουμε τις νέες παραμέτρους ανάλογα με τις ανάγκες μας ή αφήνουμε τις αρχικές τιμές.



Σχήμα 6.8: Το βασικό μενού επιλογών του Real-Time Worskshop

Πατώντας τώρα το κουμπί ‘‘build’’, που βρίσκεται στο κύριο μενού επιλογών του

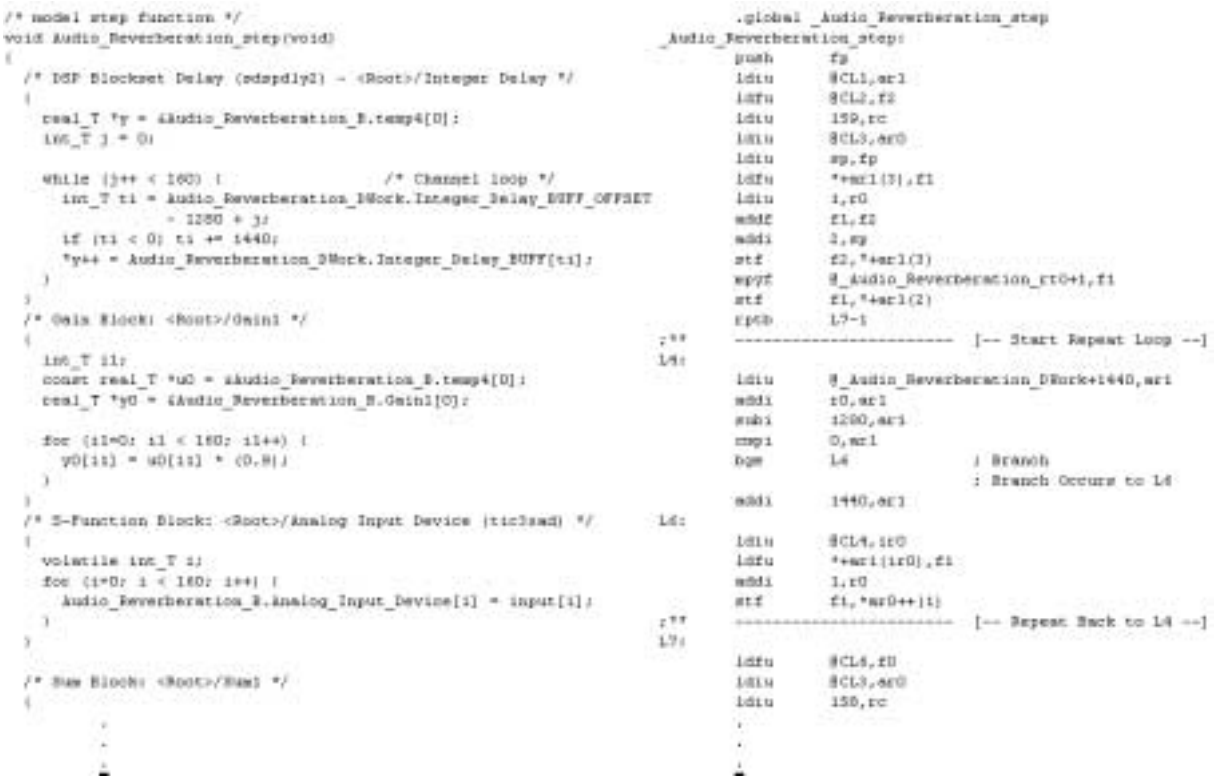
RTW, υλοποιούμε το σύστημα μας. Έχουμε την επιλογή να σταματήσουμε στο στάδιο της σύνθεσης λογισμικού ή να συνεχίσουμε με την παραγωγή κώδικα. Το εκτελέσιμο που παράγεται είναι μια ολοκληρωμένη εφαρμογή που μπορεί να εκτελεστεί άμεσα στην πλατφόρμα μας. Μπορούμε να ζητήσουμε από τη διαδικασία make να ‘‘φορτώσει’’ αυτόματα την εφαρμογή στην πλακέτα για εκτέλεση σε πραγματικό χρόνο ή στον εκσφαλματωτή για εκτέλεση βήμα-προς-βήμα (Σχήμα 6.9).



Σχήμα 6.9: ’Αποψη από το περιβάλλον του εκσφαλματωτή στον οποίο έχουμε ‘‘φορτώσει’’ το εκτελέσιμο για εκτέλεση βήμα-προς-βήμα

Βάζοντας ένα ηχητικό σήμα στην είσοδο, παίρνουμε το ίδιο στην έξοδο μαζί με ηχώ. Μπορούμε, αν θέλουμε, να αλλάξουμε τις παραμέτρους του συστήματος και να επαναλάβουμε την παραπάνω διαδικασία. Έτσι, με μικρή προσπάθεια και σε σύντομο χρονικό διάστημα, μπορούμε να παρατηρήσουμε την επίδραση που έχουν στην απόδοση του συστήματος οι διάφορες παράμετροι του (π.χ. το μέγεθος της καθυστέρησης, ο βαθμός αποδυνάμωσης προηγούμενων δειγμάτων κλπ). Ο χρόνος που ιρειαστήκαμε για την υλοποίηση του συστήματος αυτού (λίγα λεπτά) ήταν πολύ μικρός συγκρινόμενος με τις αρκετές ώρες προγραμματισμού σε γλώσσα χαμηλού επιπέδου, ακόμα και για ένα τόσο απλό σύστημα. Το μέγεθος του κώδικα (170 λέξεις για τον αλγόριθμο εισαγωγής ηχούς) κρίνεται ικανοποιητικό σε σχέση με το μέγεθος κώδικα γραμμένου από προγραμματιστή. Συνολικά απαιτήθηκαν περίπου 2K λέξεις μνήμης για τον κώδικα και περίπου 3K λέξεις για τα δεδομένα. Οι υψηλές απαιτήσεις σε

μνήμη δεδομένων οφείλονται στη χρήση πλαισίων μεγέθους 160 δειγμάτων και στην καθυστέρηση μεγέθους 8 πλαισίων. Στο Σχήμα 6.10 βλέπουμε τμήματα του C και assembly κώδικα που παρήγαγε το εργαλείο μας για το σύστημα εισαγωγής ηχούς.



```
/* model step function */
void Audio_Reverberation_step(void)
{
  /* DSP Blockset Delay (sdspdly2) - <Root>/Integer Delay */
  {
    real_T *y = &Audio_Reverberation_B.temp4[0];
    int_T j = 0;

    while (j++ < 160) {                 /* Channel loop */
      int_T ti = Audio_Reverberation_DWork.Integer_Delay_BUFF_OFFSET
                 - 1280 + j;
      if (ti < 0) ti += 1440;
      *y++ = Audio_Reverberation_DWork.Integer_Delay_BUFF[ti];
    }
  }
  /* Gain Block: <Root>/Gain1 */
  {
    int_T i1;
    const real_T *u0 = &Audio_Reverberation_B.temp4[0];
    real_T *y0 = &Audio_Reverberation_B.Gain1[0];

    for (i1=0; i1 < 160; i1++) {
      y0[i1] = u0[i1] * (0.8);
    }
  }
  /* S-Function Block: <Root>/Analog Input Device (tic3sad) */
  {
    volatile int_T i;
    for (i=0; i < 160; i++) {
      Audio_Reverberation_B.Analog_Input_Device[i] = input[i];
    }
  }

  /* Sum Block: <Root>/Sum1 */
  {
        .
        .
        .
```

```
        .global _Audio_Reverberation_step
_Audio_Reverberation_step:
        push    fp
        ldiu    @CL1,ar1
        ldfu    @CL2,f2
        ldiu    159,rc
        ldiu    @CL3,ar0
        ldiu    sp,fp
        ldfu    *+ar1(3),f1
        ldiu    1,r0
        mpyf    f1,f2
        addi    3,sp
        stf     f2,*+ar1(3)
        mpyf    @_Audio_Reverberation_rt0+1,f1
        stf     f1,*+ar1(2)
        rptb    L7-1
;**     ------------------------  [-- Start Repeat Loop --]
L4:
        ldiu    @_Audio_Reverberation_DWork+1440,ar1
        addi    r0,ar1
        subi    1280,ar1
        cmpi    0,ar1
        bge     L6              ; Branch
                                ; Branch Occurs to L6
        addi    1440,ar1
L6:
        ldiu    @CL4,ir0
        ldfu    *+ar1(ir0),f1
        addi    1,r0
        stf     f1,*ar0++(1)
;**     ------------------------  [-- Repeat Back to L4 --]
L7:
        ldfu    @CL6,f0
        ldiu    @CL3,ar0
        ldiu    158,rc
        .
        .
        .
```

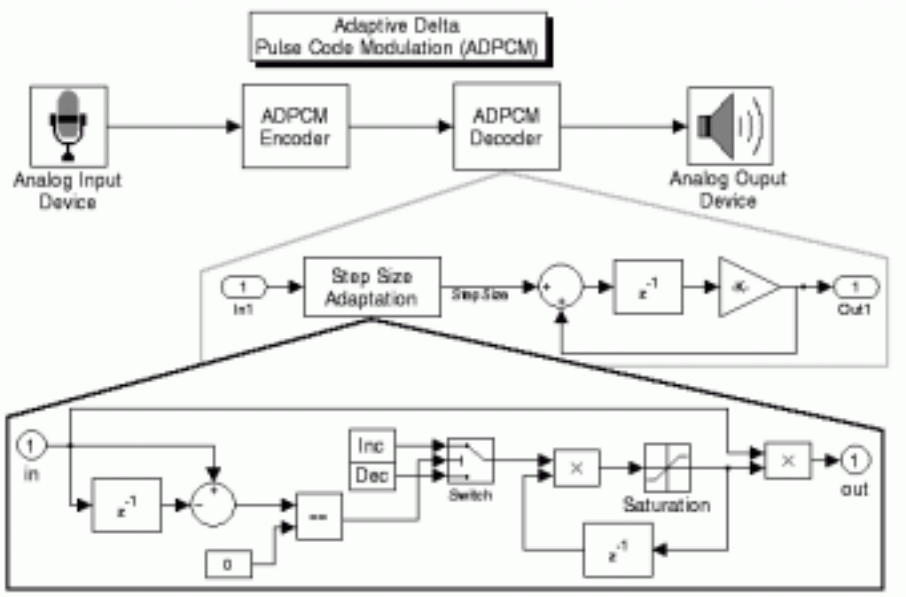
Σχήμα 6.10: Τμήμα του C και assembly κώδικα που παρήγαγε το εργαλείο μας για το σύστημα εισαγωγής ηχούς

6.4.2 Σύστημα διαμόρφωσης/αποδιαμόρφωσης ADPCM

Στο παράδειγμα αυτό υλοποιούμε ένα πιο πολύπλοκο σύστημα ADPCM. Για τις ανάγκες του παραδείγματος υλοποιούμε τόσο τον διαμορφωτή όσο και τον αποδιαμορφωτή. Έτσι το σήμα εξόδου πρέπει να είναι το ίδιο με αυτό της εισόδου με εξαίρεση κάποια απώλεια ποιότητας, που οφείλεται στον κβαντισμό του αναλογικού σήματος. Παρόλα αυτά, σε ένα εμπορικό σύστημα είναι πιθανό να υλοποιείται μόνο ο διαμορφωτής ή μόνο ο αποδιαμορφωτής.

Αντίστοιχα με το προηγούμενο παράδειγμα, σχεδιάζουμε και επαληθεύουμε το σύστημα μας στο Simulink. Στη συνέχεια, προσθέτουμε την αναλογική είσοδο / έξοδο και διαμορφώνουμε κατάλληλα τις παραμέτρους του συστήματος. Στο Σχήμα 6.11, βλέπουμε το μοντέλο του συστήματος στο Simulink.

Η παραγωγή κώδικα και εκτέλεση της εφαρμογής γίνεται με τον ίδιο τρόπο, όπως παραπάνω. Το μέγεθος του κώδικα χαμηλού επιπέδου για τον αλγόριθμο διαμόρφωσης



Σχήμα 6.11: Μοντέλο συστήματος διαμόρφωσης/αποδιαμόρφωσης ADPCM στο Simulink

και αποδιαμόρφωσης ήταν 350 λέξεις. Συνολικά για τον κώδικα χρειάστηκαν 2K λέξεις ενώ για τα δεδομένα 512 λέξεις.

Κεφάλαιο 7

Επίλογος

7.1 Συμπεράσματα

Στην εργασία αυτή παρουσιάσαμε ένα εργαλείο προγραμματισμού και παραγωγής κώδικα για τον επεξεργαστή TMS320C30 της TI. Περιγράψαμε τη μέθοδο ανάπτυξης του και καταδείξαμε τον τρόπο που αυτή μπορεί να εφαρμοστεί για μια διαφορετική πλατφόρμα εκτέλεσης.

Η χρήση του εργαλείου αυτού απλοποιεί τον προγραμματισμό των επεξεργαστών ψηφιακού σήματος και ελαχιστοποιεί το χρόνο υλοποίησης ενός νέου συστήματος. Το περιβάλλον σχεδίασης με χρήση σχηματικών διαγραμμάτων προσφέρει τα πλεονεκτήματα της διαισθητικής σχεδίασης, της αυξημένης αναγνωσιμότητας του συστήματος αλλά και της υψηλής μεταφερσιμότητας των υποσυστημάτων του. Η αυτοματοποίηση της διαδικασίας παραγωγής κώδικα, που παρέχει μια αυτόνομη ολοκληρωμένη εφαρμογή επεξεργασίας σήματος, επιτρέπει στον προγραμματιστή να επικεντρωθεί στη σχεδίαση, επαλήθευση και αξιολόγηση του συστήματος παρά στις λεπτομέρειες του προγραμματισμού σε γλώσσα χαμηλού επιπέδου.

Το μειονέκτημα της παραγωγής κώδικα λιγότερο αποδοτικού από τον αντίστοιχο γραμμένο από προγραμματιστή, δεν είναι απαγορευτικό για το είδος των εφαρμογών που υποστηρίζονται από τα χαρακτηριστικά του πλατφόρμας υλικού μας (σύνθεση και αναγνώριση φωνής, modems κλπ).

Τα χαρακτηριστικά του αυτά, το καθιστούν ιδιαίτερα κατάλληλο για χρήση σε εκπαιδευτικές διαδικασίες. Σε εργαστήρια επεξεργασίας σήματος, μπορεί να χρησιμοποιηθεί για την μελέτη της συμπεριφοράς διαφόρων συστημάτων. Οι φοιτητές, που θα το χρησιμοποιήσουν, δεν χρειάζεται να γνωρίζουν λεπτομέρειες της αρχιτεκτονικής του επεξεργαστή ψηφιακού σήματος, ούτε το σύνολο εντολών του. Επιπλέον, αποκτούν

εξοικείωση με τη φιλοσοφία των σύγχρονων εργαλείων σχεδίασης DSP συστημάτων, που χρησιμοποιούνται ευρύτατα από τις εταιρίες ανάπτυξης λογισμικού για DSP.

7.2 Βελτιώσεις - Επεκτάσεις

Ένα σύνολο δυνατοτήτων μπορούν να διερευνηθούν για την βελτίωση της αποτελεσματικότητας του εργαλείου, όσον αφορά την ποιότητα του παραγόμενου κώδικα.

Ένα πρώτο παράδειγμα, είναι η ενσωμάτωση του κώδικα των ISR εισόδου/εξόδου στην ISR επεξεργασίας, όταν το μέγεθος του πλαισίου είναι 1. Η αλλαγή αυτή θα επέτρεπε την κλήση μιας μόνο ISR ανά δείγμα αντί για δύο που καλούνται τώρα. Με δεδομένο το υψηλό κόστος από την κλήση μιας ISR, μπορεί να υπάρχει σημαντικό κέρδος κατά περιπτώσεις.

'Αλλη μια δυνατότητα, είναι η προσαρμογή του κώδικα επιλεγμένων block από τη βιβλιοθήκη του Simulink ή του DSP blockset (π.χ. φίλτρα) με γνώμονα την αποδοτικότερη εκτέλεση τους στην συγκεκριμένη πλατφόρμα. Για παράδειγμα, η τοπική δέσμευση μνήμης για πίνακες μεγάλου μεγέθους και σταθερής τιμής (όπως οι συντελεστές των φίλτρων) προκαλεί χρονοβόρες αντιγραφές κάθε φορά που καλείται η συνάρτηση και είναι επιθυμητό να αποφεύγεται.

Επίσης, η δυνατότητα του μεταγλωττιστή για function inlining, δηλαδή αντικατάσταση της εντολής κλήσης μιας συνάρτησης από το σώμα της συνάρτησης, μπορεί να βελτιώσει την απόδοση του παραγόμενου κώδικα. Από την άλλη, η χρήση της μπορεί να προκαλέσει σημαντική αύξηση του μεγέθους του κώδικα, γι'αυτό πρέπει να χρησιμοποιείται επιλεκτικά για συχνά καλούμενες συναρτήσεις μικρού μεγέθους.

Τέλος, μπορεί να διερευνηθεί η εφαρμογή διαφόρων τεχνικών βελτιστοποίησης που αναφέρθηκαν στο Κεφάλαιο 2 και σχετίζονται με την καλύτερη κατανομή των δεδομένων στη μνήμη και τη μείωση του μεγέθους του παραγόμενου κώδικα.

Εκτός από τις βελτιώσεις αυτές, οι δυνατότητες του εργαλείου που αναπτύξαμε μπορούν να επεκταθούν με αρκετούς τρόπους. Παρακάτω αναφέρονται ορισμένοι από αυτούς.

Παραγωγή κώδικα για τον C31: Ο επεξεργαστής C31, είναι ένα νεότερο μέλος της οικογένειας επεξεργαστών C3x. Στην ουσία πρόκειται για μια οικονομικότερη έκδοση του C30, με πολύ παρόμοια αρχιτεκτονική. Η υποστήριξη της παραγωγής κώδικα για τον C31, θα επιτρέψει την αξιοποίηση της πλακέτας TMS320C31 DSP Starter Kit.

Ασύγχρονη λειτουργία του αναλογικού κυκλώματος: Χρήσιμη θα ήταν, η υποστήριξη της δυνατότητας λειτουργίας σε διαφορετική συχνότητα δειγματοληψίας των δύο μονάδων μετατροπής του AIC. Αυτή η δυνατότητα, θα διεύρυνε το εύρος συστημάτων που μπορούν να υλοποιηθούν από το εργαλείο μας.

External Mode: Με τον όρο *external mode*, αναφέρεται η δυνατότητα επικοινωνίας του προσωπικού υπολογιστή με τον επεξεργαστή για την αλλαγή παραμέτρων

της εφαρμογής ενώ αυτή εκτελείται σε πραγματικό χρόνο. Η επικοινωνία αυτή βασίζεται σε ένα μοντέλο πελάτη/εξυπηρετητή και απαιτεί την ύπαρξη ενός πρωτοκόλλου επικοινωνίας. Από την πλευρά της πλακέτας EVM C30, μπορεί να υποστηριχθεί από το κύκλωμα εξομοίωσης σε συνδυασμό με τη δυνατότητα διακοπής του επεξεργαστή από υψηλής προτεραιότητας εξωτερικές διακοπές. Από τη μεριά του προσωπικού υπολογιστή στην επικοινωνία συμμετέχει το Simulink, γεγονός που επιτρέπει την αλλαγή παραμέτρων απευθείας στο σχηματικό διάγραμμα του μοντέλου που εκτελείται στον επεξεργαστή. Η υποστήριξη του external mode, θα επιτρέψει την πολύ γρήγορη προσαρμογή της εφαρμογής στις επιθυμητές ρυθμίσεις χωρίς την ανάγκη επανάληψης της διαδικασίας παραγωγής κώδικα.

Χρήση εξωτερικής σειριακής πόρτας: Η δεύτερη, σειριακή πόρτα του επεξεργαστή C30, συνδέεται σε εξωτερική σειριακή πόρτα της πλακέτας EVM C30. Ένα νέο block οδηγού συσκευής μπορεί να προστεθεί στην βιβλιοθήκη της πλατφόρμας μας, που θα υποστηρίζει τη χρήση αυτής της σειριακής πόρτας είτε για την ανταλλαγή ψηφιακών δεδομένων με τον επεξεργαστή (μέσω της σειριακής του υπολογιστή) είτε για την αποστολή ψηφιακών δεδομένων από τον επεξεργαστή σε μια άλλη συσκευή.

Παράρτημα Α

Ένα block target file για το block *quantizer*

Παρακάτω βλέπουμε ενδεικτικά ένα block target file, που χρησιμοποιείται στην παραγωγή κώδικα για το block quantizer του Simulink. Οι γραμμές που ξεκινούν με το χαρακτήρα % είναι γραμμένες στη γλώσσα TLC και αποτελούν οδηγίες για την παραγωγή κώδικα με βάση τις παραμέτρους του block και του μοντέλου. Οι υπόλοιπες γραμμές περιέχουν τον κώδικα που παράγεται σε γλώσσα C.

```
%% $RCSfile: quantize.ttlc,v $
%% File : quantize.tlc generated from quantize.ttlc revsion 1.5
%% $Date: 2000/03/08 15:33:03 $
%%
%% Copyright 1994-2000 The MathWorks, Inc.
%%
%% Abstract: Quantizer block target file.

%implements Quantizer "C"

%% Function: Outputs ================================================
%% Abstract:
%%      Y = Quantization * floor(fabs(U/Quantization)) + 0.5) * Sign(U)
%%
%function Outputs(block, system) Output
  /* %<Type> Block: %<Name> */
  %assign uIsComplex = LibBlockInputSignalIsComplex(0)
  %assign rollVars = ["U","Y","P"]
  %roll sigIdx = RollRegions, lcv = RollThreshold, block, "Roller", rollVars
    %if (uIsComplex)
      %assign ure = LibBlockInputSignal (0, "", lcv, "%<tRealPart>%<sigIdx>")
```

```
        %assign uim = LibBlockInputSignal (0, "", lcv, "%<tImagPart>%<sigIdx>")
        %assign yre = LibBlockOutputSignal(0, "", lcv, "%<tRealPart>%<sigIdx>")
        %assign yim = LibBlockOutputSignal(0, "", lcv, "%<tImagPart>%<sigIdx>")
      %else
        %assign ure = LibBlockInputSignal(0, "", lcv, sigIdx)
        %assign yre = LibBlockOutputSignal(0, "", lcv, sigIdx)
      %endif
      %%
      %assign q = LibBlockParameter(QuantizationInterval, "", lcv, sigIdx)
      %assign outpDType = LibBlockOutputSignalDataTypeId(0)
      %assign outpDTname = LibGetDataTypeNameFromId(outpDType)
      %switch outpDType
        %case tSS_DOUBLE
          %<yre> = %<q> * floor(fabs(%<ure>/(%<q>)) + 0.5) *
                (%<ure> >= 0 ? 1.0 : -1.0);
          %if (uIsComplex)
            %<yim> = %<q> * floor(fabs(%<uim>/(%<q>)) + 0.5) *
                  (%<uim> >= 0 ? 1.0 : -1.0);
          %endif
          %break
        %case tSS_SINGLE
          %<yre> = %<q> * (%<outpDTname>) (floor(fabs(%<ure>/(%<q>)) + 0.5) *
                (%<ure> >= 0 ? 1.0 : -1.0));
          %if (uIsComplex)
                %<yim> = %<q> * (%<outpDTname>) (floor(fabs(%<uim>/(%<q>)) + 0.5) *
                  (%<uim> >= 0 ? 1.0 : -1.0));
          %endif
          %break
        %default
          %assign errTxt = "Unsupported/Unhandled complex datatype: %<outDType>"
          %<LibBlockReportFatalError(block, errTxt)>
          %break
      %endswitch
    %endroll
%endfunction

%% [EOF] quantize.tlc
```

Ευρετήριο Α

Ελληνοαγγλικοί Όροι

Α

ανάλυση	resolution
αναγνωσιμότητα	readability
αναλογικό κύκλωμα εισόδου/εξόδου	analog interface circuit
αξιολόγηση	evaluation
αριθμητική και λογική μονάδα	ALU
αρχείο επικεφαλίδας	header file
αρχικοποίηση	initialization
αρχιτεκτονική	architecture

Β

βήμα της προσομοίωσης	simulation step
βρόγχος	loop

Γ

γενική δέσμευση μνήμης	global memory allocation

Δ

δέσμευση μνήμης	memory allocation

δείγμα	sample
δείκτης	pointer
διάρκεια ζωής	life cycle
διακοπή υλικού	hardware interrupt
διακριτό	discrete
διανυσματικός	vectorized
διανυσματικότητα	vectorization
διεπαφή εφαρμογής	application interface
διεργασία	task
δομή	structure
δυναμικά συνδεόμενη βιβλιοθήκη	dynamically linked library
δυναμική δέσμευση μνήμης	dynamic memory allocation
δυνατότητα επαναχρησιμοποίησης	reusability

E

εκσφαλματωτής	debugger
εκτέλεση σε πραγματικό χρόνο	real-time execution
εκτέλεση	execution
ενσωματωμένος	build-in
ενσωματωμένο σύστημα	embedded system
εντολή ελέγχου ροής	control flow statement
εντολή	instruction
επίπεδος κώδικας	inlined code
επίπεδο βελτιστοποίησης	optimization level
επαλήθευση	verification
επαναληπτική σχεδίαση και προσομοίωση	rapid prototyping
επεξεργασία ανά δείγμα	sample-based processing
επεξεργασία ανά πλαίσιο	frame-based processing
επεξεργαστές ψηφιακού σήματος	digital signal processors
επεξεργαστές ψηφιακού σήματος	DSP
επιτακτική γλώσσα προγραμματισμού	imperative programming language
εργαλείο σύνδεσης	linker
εφαρμογή	application
εύρος μνήμης	memory bandwidth

Z

ζωνοπερατό φίλτρο	bandpass filter

Θ

θεμελιώδης χρόνος δειγματοληψίας	fundamental sample time

Κ

κατάσταση	state
καταχωρητής ενεργοποίησης διακοπών	interrupt enable register
καταχωρητής κατάστασης	status register
καταχωρητής σημαίας διακοπών	interrupt flag register
κινητής υποδιαστολής	floating-point
κόστος κλήσης συνάρτησης	context-switch overhead

Λ

λειτουργικές μονάδες	functional units

Μ

μέθοδος διευθυνσιοδότησης	addressing mode
μέθοδος επίλυσης μεταβλητού βήματος	variable-step solver
μέθοδος επίλυσης σταθερού βήματος	fixed-step solver
μέθοδος επίλυσης	solver
μεταβλητή	variable
μεταγλωττιστής	compiler
μεταφερσιμότητα	modularity
μεταφραστής	interpreter
μνήμη καταχωρητών	register file
μονάδα πολλαπλασιασμού	multiplier
μοντέλο υπολογισμού ροής δεδομένων	data flow computational model
μοντελοποίηση	modeling

Ο

οδηγός συσκευής	device driver
ολισθητής	shifter

Π

πίνακας διανυσμάτων διακοπών	interrupt vector table
παράγοντας εξισσορόπησης	trade-off
παράθυρο διαλόγου	dialog box
παράμετρος	parameter

παραγωγή κώδικα	code production
περίοδος δειγματοληψίας	sample period
περιοδικό	periodic
περιφερειακές συσκευές	peripherals
πλαίσιο εκτέλεσης	run-time interface
πλαίσιο	frame
πλατφόρμα εκτέλεσης	target platform
προσδιορισμός σειράς εκτέλεσης	scheduling
προσομοίωση	simulation

P

ρουτίνα εξυπηρέτησης διακοπής	interrupt service routine

Σ

σήμα	signal
σειριακή θύρα	serial port
στατική δέσμευση μνήμης	static memory allocation
στιγμή δειγματοληψίας	sample time hit
στοίβα	stack
στυγμιότυπο	instance
συνάρτηση ενημέρωσης	update function
συνάρτηση εξόδου	output function
συνάρτηση παραγώγισης	derivative function
συνήθης διαφορική εξίσωση	ordinary differential equation
συναρτήσεις συστήματος	system functions
συνεπή μοντέλα συστημάτων	consistent system models
συνεχές	continuous
συντάκτης γραφικού περιβάλλοντος	graphical editor
συντάκτης	editor
συντελεστής φίλτρου	filter coefficient
σχηματικό διάγραμμα	block diagram
σωρός	heap
σύζευξη φάσεων	phase coupling
σύνθεση λογισμικού	software synthesis

T

τμηματοποίηση	segmentation
τοπική δέσμευση μνήμης	local memory allocation

τράπεζα μνήμης	memory bank
τυποποίηση συνάρτησης	function prototype
τύπος δεδομένων	data type

Υ

υλικό	hardware
υπερχείλιση	overflow
υψιπερατό φίλτρο	highpass filter

Φ

φίλτρο διόρθωσης	correction filter

Χ

χάρτης μνήμης	memory map
χαμηλοπερατό φίλτρο	lowpass filter
χρονισμός	timing
χρονόμετρο	timer
χρόνος ανάπτυξης νέου προϊόντος	time-to-market
χρόνος δειγματοληψίας	sample time

Ευρετήριο Β

Αγγλοελληνικοί Όροι

A

addressing mode	μέθοδος διευθυνσιοδότησης
ALU	αριθμητική και λογική μονάδα
analog interface circuit	αναλογικό κύκλωμα εισόδου/εξόδου
application interface	διεπαφή εφαρμογής
application	εφαρμογή
architecture	αρχιτεκτονική

B

bandpass filter	ζωνοπερατό φίλτρο
block diagram	σχηματικό διάγραμμα
build-in	ενσωματωμένος

C

code production	παραγωγή κώδικα
compiler	μεταγλωττιστής
consistent system models	συνεπή μοντέλα συστημάτων
context-switch overhead	κόστος κλήσης συνάρτησης
continuous	συνεχές
control flow statement	εντολή ελέγχου ροής

correction filter	φίλτρο διόρθωσης

D

data flow computational model	μοντέλο υπολογισμού ροής δεδομένων
data type	τύπος δεδομένων
debugger	εκσφαλματωτής
derivative function	συνάρτηση παραγώγισης
device driver	οδηγός συσκευής
dialog box	παράθυρο διαλόγου
digital signal processors	επεξεργαστές ψηφιακού σήματος
discrete	διακριτό
DSP	επεξεργαστές ψηφιακού σήματος
dynamic memory allocation	δυναμική δέσμευση μνήμης
dynamically linked library	δυναμικά συνδεόμενη βιβλιοθήκη

E

editor	συντάκτης
embedded system	ενσωματωμένο σύστημα
evaluation	αξιολόγηση
execution	εκτέλεση

F

filter coefficient	συντελεστής φίλτρου
fixed-step solver	μέθοδος επίλυσης σταθερού βήματος
floating-point	κινητής υποδιαστολής
frame-based processing	επεξεργασία ανά πλαίσιο
frame	πλαίσιο
function prototype	τυποποίηση συνάρτησης
functional units	λειτουργικές μονάδες
fundamental sample time	θεμελιώδης χρόνος δειγματοληψίας

G

global memory allocation	γενική δέσμευση μνήμης
graphical editor	συντάκτης γραφικού περιβάλλοντος

H

hardware interrupt	διακοπή υλικού
hardware	υλικό
header file	αρχείο επικεφαλίδας
heap	σωρός
highpass filter	υψιπερατό φίλτρο

I

imperative programming language	επιτακτική γλώσσα προγραμματισμού
initialization	αρχικοποίηση
inlined code	επίπεδος κώδικας
instance	στυγμιότυπο
instruction	εντολή
interpreter	μεταφραστής
interrupt enable register	καταχωρητής ενεργοποίησης διακοπών
interrupt flag register	καταχωρητής σημαίας διακοπών
interrupt service routine	ρουτίνα εξυπηρέτησης διακοπής
interrupt vector table	πίνακας διανυσμάτων διακοπών

L

life cycle	διάρκεια ζωής
linker	εργαλείο σύνδεσης
local memory allocation	τοπική δέσμευση μνήμης
loop	βρόγχος
lowpass filter	χαμηλοπερατό φίλτρο

M

memory allocation	δέσμευση μνήμης
memory bandwidth	εύρος μνήμης
memory bank	τράπεζα μνήμης
memory map	χάρτης μνήμης
modeling	μοντελοποίηση
modularity	μεταφερσιμότητα
multiplier	μονάδα πολλαπλασιασμού

O

optimization level	επίπεδο βελτιστοποίησης
ordinary differential equation	συνήθης διαφορική εξίσωση

output function	συνάρτηση εξόδου
overflow	υπερχείλιση

P

parameter	παράμετρος
periodic	περιοδικό
peripherals	περιφερειακές συσκευές
phase coupling	σύζευξη φάσεων
pointer	δείκτης

R

rapid prototyping	επαναληπτική σχεδίαση και προσομοίωση
readability	αναγνωσιμότητα
real-time execution	εκτέλεση σε πραγματικό χρόνο
register file	μνήμη καταχωρητών
resolution	ανάλυση
reusability	δυνατότητα επαναχρησιμοποίησης
run-time interface	πλαίσιο εκτέλεσης

S

sample period	περίοδος δειγματοληψίας
sample time hit	στιγμή δειγματοληψίας
sample time	χρόνος δειγματοληψίας
sample-based processing	επεξεργασία ανά δείγμα
sample	δείγμα
scheduling	προσδιορισμός σειράς εκτέλεσης
segmentation	τμηματοποίηση
serial port	σειριακή θύρα
shifter	ολισθητής
signal	σήμα
simulation step	βήμα της προσομοίωσης
simulation	προσομοίωση
software synthesis	σύνθεση λογισμικού
solver	μέθοδος επίλυσης
stack	στοίβα
state	κατάσταση
static memory allocation	στατική δέσμευση μνήμης
status register	καταχωρητής κατάστασης

structure	δομή
system functions	συναρτήσεις συστήματος

T

target platform	πλατφόρμα εκτέλεσης
task	διεργασία
time-to-market	χρόνος ανάπτυξης νέου προϊόντος
timer	χρονόμετρο
timing	χρονισμός
trade-off	παράγοντας εξισσορόπησης

U

update function	συνάρτηση ενημέρωσης

V

variable-step solver	μέθοδος επίλυσης μεταβλητού βήματος
variable	μεταβλητή
vectorization	διανυσματικότητα
vectorized	διανυσματικός
verification	επαλήθευση

Βιβλιογραφία

[1] Jeff Bier. Choosing High-Level Tools for DSP Design. DSPx Exhibition and Symbosium, March 1996. Berkeley Design Technology, Inc.

[2] Rulph Chassaing. *Digital Signal Processing with C and the TMS320C30.* John Wiley & Sons, 1992.

[3] Steven A. Tretter. *Communications System Design Using DSP Algorithms with Laboratory Experiments for the TMS320C30.* Plenum Press, New York, 1995.

[4] Henrik V. Sorensen and Jianping Chen. *Digital Signal Processing Laboratory Using the TMS320C30.* Prentice Hall, 1997.

[5] W. B. Ackerman. Data Flow Languages. *IEEE Computer Magazine*, 15(2):15--24, February 1982.

[6] Edward A. Lee and David G. Messerschmitt. Synchronous Data Flow. *Proceeding of the IEEE*, 75(9):1137--1344, September 1987.

[7] E. A. Lee, W. H. Ho, E. Goei, J. Bier, and S. S. Bhattacharyya. Gabriel: A design environment for DSP. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(11):1751--1762, November 1989.

[8] D. R. O'Hallaron. The ASSIGN parallel program generator. Technical report, School of Computer Science, Carnegie Mellon University, May 1991.

[9] Edward A. Lee. Overview of the Ptolemy Project. Technical Memorandum UCB/ERL M01/11, Dept. of EECS, University of California, Berkeley, March 2001.

[10] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete. Cyclo-static dataflow. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, page 32553258, May 1995.

[11] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete. Cyclo-static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397--408, February 1996.

[12] S. Ritz, M. Pankert, and H. Meyr. Optimum vectorization of scalable synchronous dataflow graphs. In *Proceedings of the International Conference on Application Specific Array Processors*, October 1993.

[13] Edward A. Lee. Representing and exploiting data parallelism using multidimensional dataflow diagrams. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, page 453456, April 1993.

[14] P. K. Murthy and E. A. Lee. An extension of multidimensional synchronous dataflow to handle arbitrary sampling lattices. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 3306--3309, May 1996.

[15] G. R. Gao, R. Govindarajan, and P. Panangaden. Well-behaved programs for dsp computation. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, March 1992.

[16] J. T. Buck and E. A. Lee. Scheduling dynamic dataflow graphs using the token flow model. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, April 1993.

[17] Shuvra S. Bhattacharyya, Rainer Leupers, and Peter Marwedel. Software Synthesis and Code Generation for Signal Processing Systems. Technical Report UMIACS-TR-99-57, Institute for Advanced Computer Studies, University of Maryland, September 1999.

[18] V. Zivojnovic, H. Schraut, M. Willems, and H. Meyr. DSPs, GPPs, and multimedia applications an evaulation using DSPstone. In *Proceedings of the International Conference on Signal Processing Applications and Technology*, November 1995.

[19] M. Levy. C compilers for DSPs flex their muscles. *EDN Access*, 12, June 1997.

[20] Edward. A. Lee. Programmable DSP Architectures - Part I. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 5(4), October 1988.

[21] Edward. A. Lee. Programmable DSP Architectures - Part II. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 6(1), January 1989.

[22] Mentor Graphics Corporation. *DSP Architect DFL User's and Reference Manual*, 1993. Version 8.2.6.

[23] Benjamin Krepp. DSP-Oriented Extensions to ANSI C. In *Proceedings of the International Conference on Signal Processing Applications and Technology*, pages 695--702, October 1994. Vol 1.

[24] Eyal Ben-Avraham. Mid-Level C-like Programming Language and Compiler for a Dedicated DSP Vector Processor. In *Proceedings of the International Conference on Signal Processing Applications and Technology*, pages 671--676, October 1994. Vol 1.

[25] A. Sudarsanam and S. Malik. Memory Bank and Register Allocation in Software Synthesis for ASIPs. In *Int. Conf. on Computer-Aided Design (ICCAD)*, pages 388--392, 1995.

[26] S. Davidson, D. Landskov, B.D. Shriver, and P.W. Mallett. Some Experiments in Local Microcode Compaction for Horizontal Machines. *IEEE Trans. on Computers*, 30(7):460--477, 1981.

[27] A. Timmer, M. Strik, J. van Meerbergen, and J. Jess. Conflict Modelling and Instruction Scheduling in Code Generation for In-House DSP Cores. In *32nd Design Automation Conference (DAC)*, pages 593--598, 1995.

[28] The Mathworks Inc. *Using Simulink*, 4th edition, November 2000.

[29] The Mathworks Inc. *DSP Blockset, User's Guide*, 4th edition, November 2000.

[30] The Mathworks Inc. *Writing S-Functions*, 4th edition, November 2000.

[31] The Mathworks Inc. *Real-Time Workshop, User's Guide*, 4th edition, September 2000.

[32] The Mathworks Inc. *Target Language Compiler, Reference Guide*, 4th edition, September 2000.

[33] Texas Instruments Inc. *TMS320C30 Evaluation Module, Technical Reference*, revision b edition, October 1990. literature number SPRU069.

[34] Texas Instruments Inc. *TMS320C3x, User's Guide*, revision l edition, July 1997. literature number SPRU031E.

[35] Leor Brenman. Setting Up TMS320 DSP Interrupts In C. Application report, Texas Instruments Inc., March 1995. literature number SPRA036.

[36] Texas Instruments Inc. *TLC32044C Analog Interface Circuit, User's Guide*, May 1995. literature number SPRU017F.

[37] Karen Baldwin and Rosemarie Piedra. Generating Efficient Code with TMS320 DSPs: Style Guidelines. Application report, Texas Instruments Inc., July 1997. literature number SPRA366.

[38] Texas Instruments Inc. *TMS320C3x/C4x Optimizing C Compiler, User's Guide*, March 1997. literature number SPRU034G.

[39] Texas Instruments Inc. *TMS320C3x/C4x Assembly Language Tools, User's Guide*, revision c edition, March 1997. literature number SPRU035C.