ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΦΥΣΙΚΗΣ
ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΦΥΣΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ ΜΕΤΑΠΤΥΧΙΑΚΟΥ “ΗΛΕΚΤΡΟΝΙΚΟΥ ΑΥΤΟΜΑΤΙΣΜΟΥ”

Σχεδίαση και ανάπτυξη προγραμμάτων για επικοινωνία και επεξεργασία δεδομένων, υλοποιημένα στην κάρτα ανάπτυξης Renesas SK-S2G7, με στόχο την εφαρμογή σε εργαστήρια μικροεπεξεργαστών

Κωνσταντίνος Σαπουντζής
Α.Μ. 7110132100216

Επιβλέπων:

Διονύσιος Ρεΐσης

Καθηγητής

ΑΘΗΝΑ
01/2024

Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε στα πλαίσια του διατμηματικού μεταπτυχιακού προγράμματος σπουδών του Τμήματος Φυσικής του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών, κατά το ακαδημαϊκό έτος 2023-2024, υπό την επίβλεψη του καθηγητή κ. Διονύση Ρεΐση.

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή κ. Διονύση Ρεΐση για την καθοδήγηση του και τη βοήθεια του στην ολοκλήρωση της διπλωματικής εργασίας. Ακόμη θα ήθελα να ευχαριστήσω τους φίλους και συγγενείς που με στήριξαν σε όλη τη διάρκεια των σπουδών μου.

Περίληψη

Η ακόλουθη διπλωματική εργασία θα εστιάσει στην ανάπτυξη προγραμμάτων με στόχο την εφαρμογή σε εργαστήρια μικροεπεξεργαστών. Οι εφαρμογές που θα αναπτυχθούν στοχεύουν στον μικροελεγκτή S7G2 της Renesas, ο οποίος αποτελείται από έναν αποδοτικό μικροεπεξεργαστή και ενσωματωμένες περιφερειακές συσκευές. Τα προγράμματα που θα αναπτυχθούν θα χρησιμοποιούν τις περιφερειακές συσκευές του μικροελεγκτή και η υλοποίηση τους θα γίνει στη κάρτα ανάπτυξης SK-S7G2.

Για την ανάπτυξη των εφαρμογών θα γίνει χρήση της διεπαφής προγραμματισμού εφαρμογών (API, application programming interface) που προσφέρει η Renesas καθώς και του στρώματος απόκρυψης υλικού (HAL, hardware abstraction layer). Ακόμη θα μελετηθεί το περιβάλλον ανάπτυξης και οι λειτουργίες του.

Όσον αφορά στις περιφερειακές συσκευές, ιδιαίτερη σημασία θα δοθεί στη μονάδα διαχείρισης των θυρών εισόδου-εξόδου, στη μονάδα διαχείρισης διακοπών (interrupts) και στη μονάδα χρονιστών-απαριθμητών GPT. Οι μονάδες αυτές αποτελούν πολύ σημαντικά εργαλεία για τον προγραμματισμό μικροελεγκτών και για την ανάπτυξη εφαρμογών σε ενσωματωμένα συστήματα.

Περιεχόμενα

Κεφάλαιο 1: Μικροεπεξεργαστές

1.1 Εισαγωγή στους μικροεπεξεργαστές

Ένας μικροεπεξεργαστής αποτελεί την κεντρική μονάδα επεξεργασίας (CPU) ενός ψηφιακού συστήματος, και περιλαμβάνει τις περισσότερες ή όλες τις λειτουργίες μιας κεντρικής μονάδας επεξεργασίας ενός ηλεκτρονικού υπολογιστή σε ένα ενιαίο ολοκληρωμένο κύκλωμα (IC). Λειτουργώντας ως ο εγκέφαλος ηλεκτρονικών συσκευών, ερμηνεύει και εκτελεί εντολές που αποθηκεύονται στη μνήμη, και πραγματοποιεί αριθμητικούς υπολογισμούς και λογικές διεργασίες. Ένας μικροεπεξεργαστής περιλαμβάνει μια μονάδα αριθμητικής και λογικής (ALU), μια μονάδα ελέγχου και καταχωρητές στη δομή του. Λειτουργεί υπό τον έλεγχο ενός σήματος ρολογιού, το οποίο συγχρονίζει την εκτέλεση των εντολών και τη ροή δεδομένων η οποία γίνεται μέσω διαύλων.

Η ευελιξία και οι υψηλές επιδόσεις των μικροεπεξεργαστών τους καθιστούν απαραίτητους σε αμέτρητες εφαρμογές του σύγχρονου υπολογιστικού κόσμου, από καθημερινές ηλεκτρονικές συσκευές μέχρι εξειδικευμένα βιομηχανικά και επιστημονικά συστήματα.

1.2 Εισαγωγή στους μικροελεγκτές

Στην οικογένεια των μικροεπεξεργαστών ανήκουν και οι μικροελεγκτές. Οι μικροελεγκτές μπορούν να οριστούν συνοπτικά, ως μικρά υπολογιστικά συστήματα. Πιο συγκεκριμένα, οι μικροελεγκτές είναι μικροεπεξεργαστές οι οποίοι διαθέτουν και ενσωματωμένες περιφερειακές συσκευές. Μερικές περιφερειακές συσκευές που συναντώνται στους μικροελεγκτές είναι οι μονάδες διακοπών (Interrupts), οι μετατροπείς αναλογικού σήματος σε ψηφιακό (ADC), οι χρονιστές-απαριθμητές (Timer/Counter), οι μονάδες επικοινωνίας (UART, I2C, κλπ) και πολλές άλλες.

Οι μικροελεγκτές διακρίνονται για την αυτονομία τους αφού δεν απαιτούν πολλά άλλα ολοκληρωμένα κυκλώματα για να λειτουργήσουν, το χαμηλό κόστος, το μικρό μέγεθος, τον μεγάλο αριθμό ακροδεκτών εισόδου-εξόδου και την ευκολία στην υλοποίηση εφαρμογών. Για τους παραπάνω λόγους, οι μικροελεγκτές χρησιμοποιούνται ευρέως στον κλάδο των ενσωματωμένων συστημάτων, όπως σε ηλεκτρικές και ηλεκτρονικές συσκευές, στην αυτοκινητοβιομηχανία, σε διαστημικές εφαρμογές καθώς και σε εφαρμογές που απαιτούν αυτοματοποίηση.

1.3 Βασικές περιφερειακές συσκευές των μικροελεγκτών

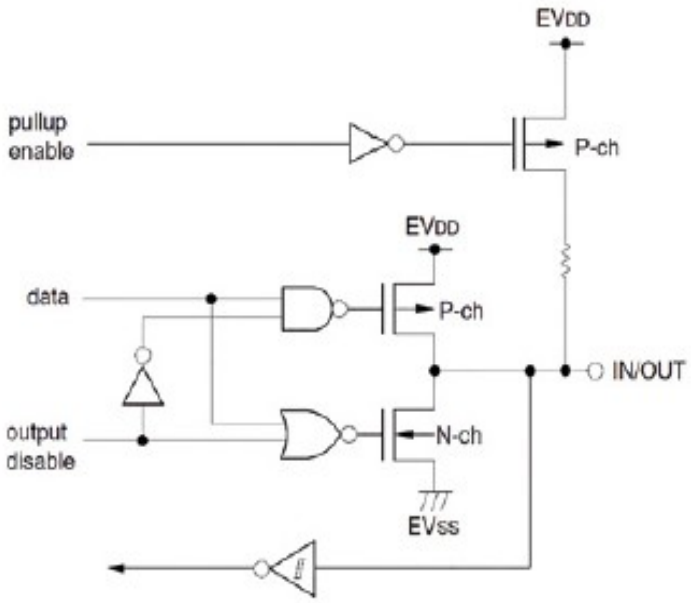
Σε αυτήν την υποενότητα θα γίνει αναφορά σε ορισμένες περιφερειακές μονάδες των μικροελεγκτών. Οι μονάδες αυτές συναντώνται στους περισσότερους, αν όχι σε όλους, τους μικροελεγκτές και είναι απαραίτητες για την λειτουργία τους και την ανάπτυξη εφαρμογών σε ενσωματωμένα συστήματα.

Θύρες εισόδου-εξόδου (I/O Ports):

Οι θύρες εισόδου-εξόδου διαχειρίζονται τα εισερχόμενα και εξερχόμενα ψηφιακά σήματα. Οι θύρες αυτές μπορεί να είναι γενικής χρήσης (GPIO), δηλαδή να μπορούν να δουλέψουν είτε ως θύρες εισόδου, είτε εξόδου ανάλογα με τις ανάγκες της εφαρμογής, ή εναλλακτικά να δεσμεύονται για χρήση σε κάποια συγκεκριμένη περιφερειακή συσκευή.

Οι θύρες εισόδου-εξόδου καθιστούν εφικτή την σύνδεση του μικροελεγκτή και του “εξωτερικού κόσμου”. Με αυτές ο μικροελεγκτής μπορεί να επικοινωνεί με άλλες συσκευές, να δέχεται δεδομένα από αισθητήρες, να τροφοδοτεί συσκευές όπως λυχνίες LED και μοτέρ και να εκτελεί πολλές ακόμη λειτουργίες.

Κάθε θύρα αποτελείται από ακροδέκτες (pins) και ανάλογα με το μέγεθος της θύρας (8-bit,16-bit κλπ) περιέχει τον αντίστοιχο αριθμό ακροδεκτών. Η πρόσβαση στις θύρες εισόδου-εξόδου, ώστε να χρησιμοποιηθούν σε κάποια εφαρμογή γίνεται μέσω των αντίστοιχων καταχωρητών. Χρησιμοποιώντας τους καταχωρητές αυτούς ο προγραμματιστής μπορεί να επιλέξει τη λειτουργία μιας θύρας (είσοδος ή έξοδος) και την τιμή που θα έχει ο κάθε ακροδέκτης (0 ή 1). Στην εικόνα 1.1 αποτυπώνεται ένα τυπικό σχεδιάγραμμα κυκλώματος για έναν ακροδέκτη εισόδου-εξόδου.



Εικόνα 1.1: Σχεδιάγραμμα κυκλώματος για έναν ακροδέκτη εισόδου-εξόδου

Πρωτόκολλα επικοινωνίας

Για την αποτελεσματική χρήση των μικροελεγκτών είναι απαραίτητη η επικοινωνία των ίδιων με άλλες συσκευές, οι οποίες μπορεί να είναι αισθητήρες, υπολογιστές ακόμη και άλλοι μικροελεγκτές. Όπως αναφέρθηκε προηγουμένως, η λήψη και αποστολή δεδομένων γίνεται μέσω των θυρών εισόδου-εξόδου, αλλά είναι αναγκαίο να υπάρχει και ένας κώδικας επικοινωνίας ώστε να έχει νόημα η αλληλουχία των bit που περνάει από τις θύρες αυτές. Ένας τέτοιος κώδικας ονομάζεται πρωτόκολλο επικοινωνίας και συνήθως γίνεται η διάκριση μεταξύ των σειριακών και παράλληλων πρωτοκόλλων ανάλογα με το πως γίνεται η μεταφορά δεδομένων. Η κωδικοποίηση και αποκωδικοποίηση της πληροφορίας σε κάποιο πρωτόκολλο επικοινωνίας είναι πολύπλοκη διαδικασία και έτσι, οι περισσότεροι μικροελεγκτές έχουν περιφερειακές μονάδες που συγκεκριμένα αναλαμβάνουν αυτή την διαδικασία. Έτσι ο επεξεργαστής καθίσταται ελεύθερος να εκτελεί άλλες λειτουργίες.

Χρονιστές – Απαριθμητές (Timers/Counters)

Ο χρονιστής-απαριθμητής (timer/counter) είναι μια περιφερειακή συσκευή ειδικού σκοπού η οποία χρησιμοποιείται για την μέτρηση χρονικής διάρκειας και γεγονότων. Οι χρονιστές-απαριθμητές είναι ικανοί να μετράνε γεγονότα που συμβαίνουν εξωτερικά του μικροελεγκτή, να μετράνε χρονικές περιόδους, ακόμα και να παράγουν ψηφιακά σήματα (PWM).

Οι χρονιστές-απαριθμητές είναι απαραίτητοι σε ενσωματωμένα συστήματα πραγματικού χρόνου, όπου ο ακριβής συγχρονισμός γεγονότων ή διεργασιών είναι άκρως σημαντικός.

Μετατροπείς αναλογικού σήματος σε ψηφιακό (ADC)

Πολλές φορές οι πληροφορίες που δέχεται ο μικροελεγκτής από εξωτερικές συσκευές, όπως για παράδειγμα από αισθητήρες, βρίσκονται σε αναλογική μορφή. Ο μικροελεγκτής όμως, μπορεί να κατανοήσει και να διαχειριστεί μόνο ψηφιακά σήματα και κατά συνέπεια είναι αναγκαία η μετατροπή των αναλογικών σημάτων. Η περιφερειακή συσκευή που αναλαμβάνει αυτή τη μετατροπή είναι ο μετατροπέας αναλογικού σήματος σε ψηφιακό (ADC) ενώ πολλοί μικροελεγκτές διαθέτουν και συσκευή που εκτελεί την ανάποδη διαδικασία (DAC).

Οι μετατροπείς αναλογικού σήματος σε ψηφιακό είναι απραίτητοι σε ενσωματωμένα συστήματα που διαθέτουν αναλογικούς αισθητήρες, που επεξεργάζονται ήχο και που διαθέτουν επικοινωνία σε αναλογική μορφή (π.χ. modems).

Κεφάλαιο 2: Το οικοσύστημα για την ανάπτυξη και υλοποίηση εφαρμογών

Στην ενότητα αυτή θα μελετηθεί το οικοσύστημα μέσα στο οποίο θα γίνει η ανάπτυξη των εφαρμογών και η υλοποίηση τους. Πιο συγκεκριμένα, θα γίνει περιγραφή του μικροελεγκτή S7G2 καθώς και της κάρτας ανάπτυξης SK-S7G2 που τον περιέχει, και στη συνέχεια, θα γίνει συνοπτική παρουσίαση των προγραμμάτων και εργαλείων σε επίπεδο λογισμικού που θα χρησιμοποιηθούν για την ανάπτυξη των εφαρμογών.

2.1 Ο Μικροελεγκτής S7G2

Ο μικροελεγκτής S7G2 αποτελεί ένα από τα μοντέλα της σειράς S7 της οικογένειας μικροελεγκτών Synergy και είναι σχεδιασμένος ώστε να έχει υψηλές επιδόσεις και αυξημένη ικανότητα για συνδεσιμότητα.

2.1.1 Ο Επεξεργαστής Cortex-M4

Ο “εγκέφαλος” του μικροελεγκτή είναι ο επεξεργαστής Cortex-M4, ο οποίος μεταξύ άλλων συνδυάζει υψηλές επιδόσεις, μικρή κατανάλωση ενέργειας, ευκολία στη χρήση και χαμηλό κόστος. Ο Cortex-M4 είναι ένας αποδοτικός επεξεργαστής με συχνότητα λειτουργίας έως και 240 Mhz και κατανάλωση ρεύματος συνήθως μικρότερη από 200 μA/MHz (εξαρτάται από τον μικροελεγκτή, μερικοί καταναλώνουν και κάτω από 100 μA/MHz) .

Η αρχιτεκτονική των 32-bit που διαθέτει επιτρέπει την πρόσβαση σε έως και 4 GB μνήμης και προαιρετικά μπορεί να διαθέτει μονάδα προστασία μνήμης (MPU). Η μονάδα προστασίας μνήμης χωρίζει τη μνήμη σε περιοχές και ορίζει ποιος επιτρέπεται να έχει πρόσβαση σε αυτές, εμποδίζοντας έτσι την παραποίηση των δεδομένων από κακόβουλα λογισμικά. Μια ακόμη χρήσιμη λειτουργία της μονάδας αυτής είναι η ικανότητα της να διαμορφώνει περιοχές μνήμης, έτσι ώστε να μπορούν μόνο να διαβαστούν, αποτρέποντας έτσι την τυχαία διαγραφή δεδομένων. Η μονάδα προστασίας μνήμης χρησιμοποιείται συνήθως σε συνδυασμό με κάποιο λειτουργικό σύστημα.

Μερικά ακόμη αξιοσημείωτα χαρακτηριστικά του επεξεργαστή Cortex-M4 είναι η μονάδα κινητής υποδιαστολής (FPU), η μονάδα ψηφιακής επεξεργασίας σήματος (DSP) που επιτρέπει σε πολλαπλούς υπολογισμούς δεδομένων να εκτελούνται παράλληλα διευκολύνοντας έτσι εφαρμογές που απαιτούν επεξεργασία ήχου ή εικόνας καθώς και διάφορα άλλα χαρακτηριστικά που βοηθούν στην αποσφαλμάτωση (debugging) των

προγραμμάτων . Τέλος, ο Cortex-M4 διαθέτει μονάδα διαχείρισης διακοπών, η οποία θα μελετηθεί σε επόμενη ενότητα.

2.1.2 Χαρακτηριστικά του μικροελεγκτή S7G2

Για να γίνουν κατανοητές οι δυνατότητες του μικροελεγκτή και οι πιθανές εφαρμογές που μπορούν να αναπτυχθούν σε αυτόν, θα γίνει μια σύντομη αναφορά σε μερικά από τα κυριότερα χαρακτηριστικά του.

Αρχικά, στο κομμάτι της μνήμης διαθέτει 4 MB μνήμης τύπου FLASH για αποθήκευση του λογισμικού και 64 KB μνήμης ίδιου τύπου για μόνιμη αποθήκευση δεδομένων. Διαθέτει ακόμη 640 KB μνήμη τυχαίας προσπέλασης SRAM για αποθήκευση δεδομένων κατά την εκτέλεση προγραμμάτων. Τέλος, διαθέτει μονάδα άμεσης προσπέλασης μνήμης (DMAC, direct memory access controller) 8 καναλιών η οποία αναλαμβάνει τη μεταφορά δεδομένων χωρίς να επιβαρύνεται ο επεξεργαστής.

Μεγάλη ποικιλία παρατηρείται στα ρολόγια τα οποία επιλέγονται ανάλογα με τις απαιτήσεις της κάθε εφαρμογής σε ταχύτητα και κατανάλωση ενέργειας. Για τη μέτρηση γεγονότων και χρονικής διάρκειας ο μικροελεγκτής διαθέτει γενικής χρήσης χρονιστές-απαριθμητές των 32-bit με 14 κανάλια, οι οποίοι μπορούν και να παράγουν σήματα. Χρήσιμο είναι και το ρολόι πραγματικού χρόνου που συναντάται μεταξύ των περιφερειακών συσκευών του μικροελεγκτή.

Όσον αφορά στο κομμάτι της επικοινωνίας, ο μικροελεγκτής υποστηρίζει τα πιο δημοφιλή πρωτόκολλα επικοινωνίας όπως σύγχρονα και ασύγχρονα σειριακά (UART,IIC,SPI,QSPI), το πρωτόκολλο CAN (controller area network) το οποίο είναι χρήσιμο σε εφαρμογές με υψηλό ηλεκτρομαγνητικό θόρυβο, μονάδα για επικοινωνία μέσω του πρωτοκόλλου USB καθώς και μονάδα ETHERNET ώστε να είναι δυνατή η ανάπτυξη εφαρμογών δικτύου.

Για την σύνδεση με τον αναλογικό κόσμο ο μικροελεγκτής διαθέτει μετατροπείς αναλογικού σήματος σε ψηφιακό των 12-bit και μετατροπέα ψηφιακού σήματος σε αναλογικό με ακρίβεια 12-bit και ενισχυτή σήματος εξόδου. Ο μικροελεγκτής S7G2 περιέχει ακόμη αισθητήρα θερμοκρασίας και υψηλής ταχύτητας αναλογικό συγκριτή.

Στο κομμάτι των γραφικών συμπεριλαμβάνεται η μονάδα διαχείρισης οθονών LCD, η οποία υποστηρίζει διάφορους τύπους δεδομένων και πάνελ οθονών καθώς και μονάδες για παραγωγή και επεξεργασία εικόνας.

Κλείνοντας αυτή την υποενότητα, μια ακόμη πολύ χρήσιμη μονάδα είναι αυτή του διαχειριστή σύνδεσης γεγονότων (ELC, event link controller), ο οποίος παρακολουθεί τις αιτήσεις για διακοπή που παράγουν οι διάφορες περιφερειακές συσκευές και τις χρησιμοποιεί ως εναρκτήρια σήματα για άλλες περιφερειακές συσκευές. Με αυτό το τρόπο επιτυγχάνεται η σύνδεση μεταξύ συσκευών χωρίς να επιβαρύνεται ο επεξεργαστής.

2.2 Η κάρτα ανάπτυξης SK-S7G2

2.2.1 Εισαγωγικές έννοιες στις κάρτες ανάπτυξης

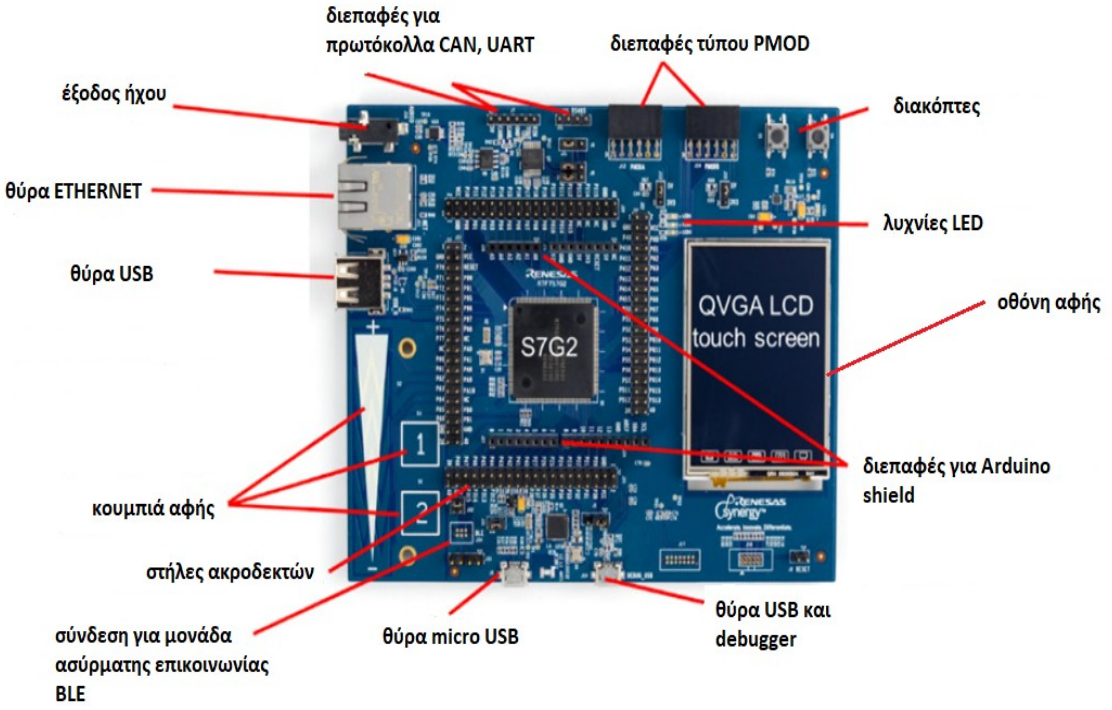
Ο κάρτες ανάπτυξης είναι ένα σύνολο ολοκληρωμένων κυκλωμάτων, ηλεκτρικών στοιχείων και ηλεκτρονικών συσκευών κατασκευασμένο πάνω σε μια ηλεκτρονική κάρτα. Οι κάρτες ανάπτυξης συγκεντρώνονται σε ένα κυρίως εξάρτημα, όπως ένας μικροελεγκτής, και τα κυκλώματα ή οι ηλεκτρονικές συσκευές που διαθέτουν υπάρχουν για να υποστηρίζουν τη λειτουργία και χρήση του κυρίως εξαρτήματος. Το υλικό (hardware), δηλαδή το παραπάνω σύνολο κυκλωμάτων και συσκευών που διαθέτει μια κάρτα εξαρτάται από τα χαρακτηριστικά του μικροελεγκτή καθώς και το είδος των εφαρμογών για τις οποίες είναι σχεδιασμένη. Μερικά παραδείγματα υλικού μιας κάρτας είναι η τροφοδοσία του μικροελεγκτή, οι διάφορες θύρες και διεπαφές για την πρόσβαση στις περιφερειακές συσκευές του μικροελεγκτή, το σύνολο των ακροδεκτών προς σύνδεση με εξωτερικές συσκευές καθώς και οι διάφορες διεπαφές για αλληλεπίδραση του χρήστη με τον μικροελεγκτή. Οι κάρτες ανάπτυξης κάνουν εύκολη και γρήγορη τη χρήση ενός μικροελεγκτή και χρησιμοποιούνται για εκμάθηση νέων μικροελεγκτών και ταχύτερη ανάπτυξη εφαρμογών σε αυτούς.

2.2.2 Η κάρτα ανάπτυξης SK-S7G2

Η κάρτα ανάπτυξης SK-S7G2 περιέχει και είναι σχεδιασμένη γύρω από τον μικροελεγκτή Renesas Synergy S7G2 και αποτελεί την κάρτα στην οποία θα γίνει η υλοποίηση των εφαρμογών. Τα χαρακτηριστικά και το υλικό (hardware) που διαθέτει η κάρτα συνοψίζονται παρακάτω:

- Μικροελεγκτής S7G2 με 176 ακροδέκτες
- Οθόνη αφής LCD 240x320 πίξελ
- Διεπαφές για σειριακά πρωτόκολλα επικοινωνίας
- Διεπαφή για πρωτόκολλο CAN και USB
- Διεπαφή για πρωτόκολλο ETHERNET
- Μνήμη 8 MB
- Έξοδο ήχου και ενισχυτή
- 3 λυχνίες LED
- 2 διακόπτες συνδεδεμένους με ακροδέκτες διακοπών (interrupt)

- 3 κουμπιά αφής που μπορούν να πυροδοτήσουν διακοπές (interrupt)
- Διεπαφή JTAG και μονάδα αποσφαλμάτωσης (debugger)



Εικόνα 2.1 : Κάρτα ανάπτυξης SK-S7G2

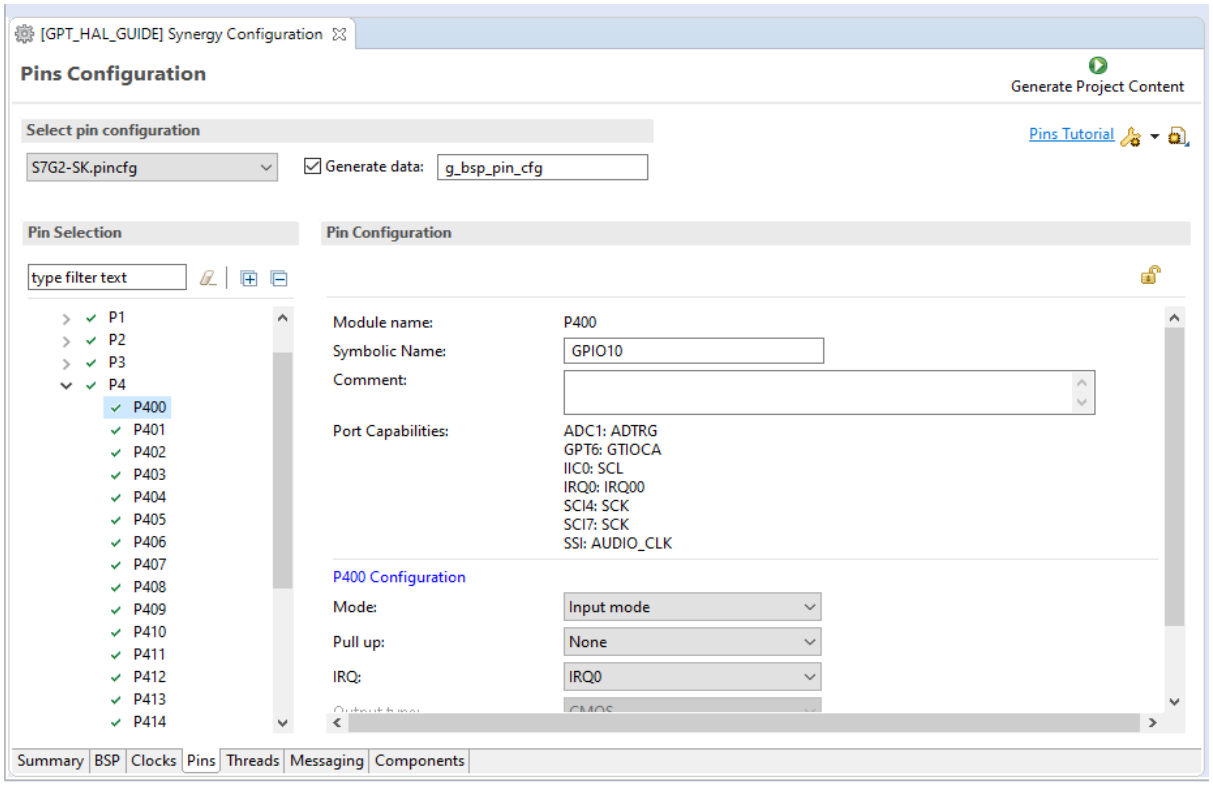
2.3 Το περιβάλλον ανάπτυξης

Για την ανάπτυξη και υλοποίηση εφαρμογών χρειάζεται ένα σύνολο από εργαλεία, τα οποία επιτρέπουν την ανάπτυξη κώδικα, τον προγραμματισμό του μικροελεγκτή και τον έλεγχο της ορθής λειτουργίας του προγράμματος. Για την επίτευξη των παραπάνω χρησιμοποιείται ένα ολοκληρωμένο περιβάλλον ανάπτυξης και συγκεκριμένα το e²studio, το οποίο περιέχει όλα τα απαραίτητα εργαλεία για τον προγραμματισμό του μικροελεγκτή S7G2.

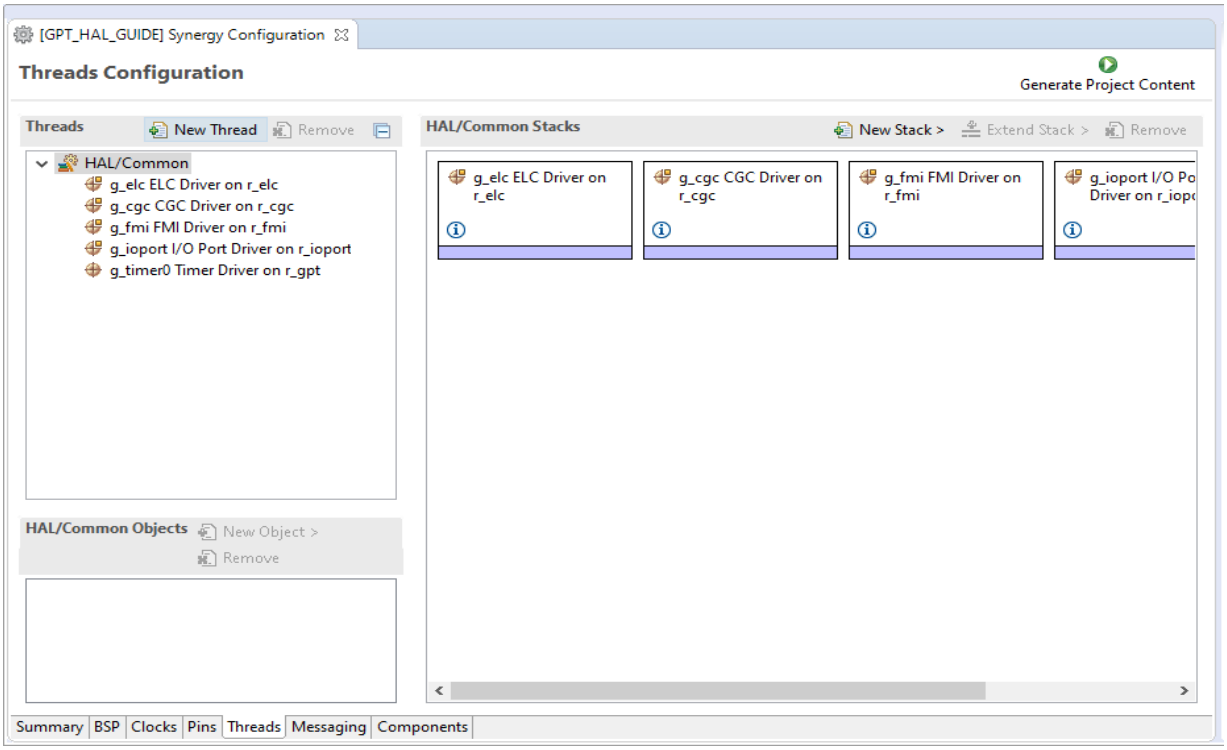
Για τον προγραμματισμό του μικροελεγκτή θα γίνει χρήση της γλώσσας προγραμματισμού C, η οποία είναι μια γλώσσα υψηλού επιπέδου και μπορεί εύκολα να κατανοηθεί και να χρησιμοποιηθεί από ανθρώπους. Ο μικροελεγκτής, όμως, δουλεύει μόνο με γλώσσα μηχανής, δηλαδή ακολουθίες από 0 και 1, έτσι είναι απαραίτητο να υπάρχει ένα εργαλείο το οποίο θα μεταφράζει το πρόγραμμα που είναι γραμμένο στη γλώσσα C σε γλώσσα μηχανής. Το εργαλείο που αναλαμβάνει την δουλεία αυτή είναι ο μεταγλωττιστής.

Αναγκαίο είναι επίσης το εργαλείο που ονομάζεται linker και αναλαμβάνει την σύνθεση πολλαπλών πηγαίων αρχείων σε ένα εκτελέσιμο αρχείο. Τέλος, κανένα σχεδόν πρόγραμμα δεν είναι ελεύθερο από σφάλματα και έτσι είναι απαραίτητο να υπάρχουν εργαλεία τα οποία θα επιτρέπουν τον εντοπισμό πιθανών σφαλμάτων. Σε συνδυασμό με το μεταγλωττιστή, ο οποίος έχει τη δυνατότητα να ελέγχει για πιθανά σφάλματα στο συντακτικό του προγράμματος, το ολοκληρωμένο περιβάλλον ανάπτυξης e²studio διαθέτει και debugger. Ο debugger μπορεί να ελέγχει τη ροή του προγράμματος, να θέτει σημεία παύσης του προγράμματος και να εκτελεί το πρόγραμμα γραμμή προς γραμμή διευκολύνοντας έτσι τη διαδικασία της αποσφαλμάτωσης.

Μια άλλη πολύ σημαντική λειτουργία του περιβάλλοντος ανάπτυξης e²studio είναι η ενεργοποίση και ρύθμιση περιφερειακών συσκευών του μικροελεγκτή μέσα από αυτό. Το περιβάλλον ανάπτυξης e²studio παρέχει ένα γραφικό περιβάλλον μέσα από το οποίο μπορεί να ρυθμιστεί η λειτουργία των ακροδεκτών που χρησιμοποιούνται στην εφαρμογή (π.χ. αν θα λειτουργούν ως ακροδέκτες εισόδου ή εξόδου), να ρυθμιστεί η συχνότητα λειτουργίας, να εισαχθεί στην εφαρμογή λογισμικό υποστήριξης για περιφερειακές συσκευές (drivers), να προστεθεί το λειτουργικό σύστημα και πολλά άλλα. Επιλέγοντας τις επιθυμητές ρυθμίσεις και με το πάτημα ενός μόνο κουμπιού το e²studio αναλαμβάνει αυτόματα την παραγωγή του κώδικα που απαιτείται για την πραγματοποίηση των ρυθμίσεων αυτών. Τα πλεονεκτήματα της “γραφικής λύσης” είναι τεράστια, καθώς διευκολύνεται και επιταχύνεται η διαδικασία της ανάπτυξης εφαρμογών, αφού ο προγραμματιστής δεν χρειάζεται να γνωρίζει κάθε λεπτομέρεια του υλικού για τη χρήση κάποιας περιφερειακής συσκευής και ούτε να γράφει κάθε φορά βασικό κώδικα αρχικοποίησης. Στις παρακάτω εικόνες αποτυπώνονται παραδείγματα της χρήσης του γραφικού περιβάλλοντος.



Εικόνα 2.2 : Παράθυρο προς ρύθμιση ακροδεκτών



Εικόνα 2.3 : Παράθυρο για εισαγωγή λογισμικού περιφερειακών συσκευών

2.4 Το “στρώμα” απόκρυψης υλικού

Ο όρος “στρώμα απόκρυψης υλικού” (hardware abstraction layer, HAL) αναφέρεται στο σύνολο των βιβλιοθηκών, των συναρτήσεων και γενικότερα του λογισμικού υποστήριξης, το οποίο έχει σκοπό να αποκρύψει το υλικό από το χρήστη.

Ο προγραμματισμός του μικροελεγκτή και των περιφερειακών συσκευών γίνεται μέσω των καταχωρητών ελέγχου της κάθε συσκευής. Αλλάζοντας τις τιμές στα bit των καταχωρητών μπορεί να γίνει αρχικοποίηση κάποιας περιφερειακής συσκευής και να ρυθμιστεί η λειτουργία της. Αυτό όμως απαιτεί γνώση του υλικού και της αρχιτεκτονικής κάθε συσκευής, ενώ ταυτόχρονα δυσχεραίνεται και επιβραδύνεται η διαδικασία ανάπτυξης εφαρμογών. Προς αντιμετώπιση αυτού, η εταιρεία Renesas έχει αναπτύξει το “στρώμα” απόκρυψης υλικού και ένα πακέτο λογισμικού το οποίο επιταχύνει την ανάπτυξη εφαρμογών, διευκολύνει την ανάγνωση του κώδικα καθώς και τη μεταφορά των εφαρμογών σε άλλους μικροελεγκτές.

Το πακέτο λογισμικού περιέχει το λογισμικό υποστήριξης της πλακέτας (BSP, board support package), το οποίο είναι σχεδιασμένο για την πλακέτα SK-S7G2 και τον μικροελεγκτή S7G2. Μέσω της χρήσης του BSP είναι δυνατή η παραγωγή κώδικα αρχικοποίησης (start-up code) και η ρύθμιση των λειτουργιών της κάρτας. Το BSP περιλαμβάνει το σύνολο των συναρτήσεων που αλληλεπιδρούν με το υλικό (ρύθμιση καταχωρητών, κλπ) και συντελούν τη διεπαφή προγραμματισμού εφαρμογών (API, application programming interface). Ακόμη, οι συναρτήσεις αυτές ομαδοποιούνται σε μονάδες, και κάθε μονάδα αναφέρεται αποκλειστικά σε κάποια περιφερειακή συσκευή. Μέσω των μονάδων αυτών, μπορεί να γίνει χρήση και ρύθμιση της λειτουργίας της αντίστοιχης συσκευής. Το σύνολο αυτών των μονάδων συντελούν το “στρώμα” απόκρυψης υλικού, το οποίο και θα χρησιμοποιήσουμε για να αναπτύξουμε εφαρμογές.

Κεφάλαιο 3: Εφαρμογή 1 - Blink LED

3.1 Εισαγωγή

Η εφαρμογή “Blink LED” αποτελεί μια αφετηρία στον προγραμματισμό μικροελεγκτών και εισάγει κάποιες βασικές έννοιες του προγραμματισμού ενσωματωμένων συστημάτων. Η εφαρμογή είναι πολύ απλή και η λειτουργία της αποτυπώνεται στην ενεργοποίηση και απενεργοποίηση μιας λυχνίας LED με κάποια περιοδικότητα. Η λυχνία που θα χρησιμοποιηθεί είναι η κόκκινη λυχνία της κάρτας ανάπτυξης. Οι λυχνίες της κάρτας είναι συνδεδεμένες με ακροδέκτες του μικροελεγκτή, και η χρήση αυτών θα γίνει μέσω του λογισμικού πακέτου και του HAL.

Η εφαρμογή “Blink LED” στοχεύει στην εξοικείωση με τη διαδικασία και τα εργαλεία για την ανάπτυξη προγραμμάτων σε μικροελεγκτές, και αποτελεί την βάση για ανάπτυξη πιο προηγμένων εφαρμογών σε μικροεπεξεργαστές και ενσωματωμένα συστήματα.

3.2 Ανάπτυξη και υλοποίηση

Η λειτουργία της εφαρμογής “Blink LED” περιγράφεται από τα ακόλουθα βήματα, και για την ανάπτυξη του κώδικα της εφαρμογής, προσφέρεται το παρακάτω διάγραμμα ροής.

Βήμα 1: Εκκίνηση του προγράμματος

Βήμα 2: Αρχικοποίηση των δομών που θα χρησιμοποιηθούν

Βήμα 3: Ενεργοποίηση της κόκκινης λυχνίας

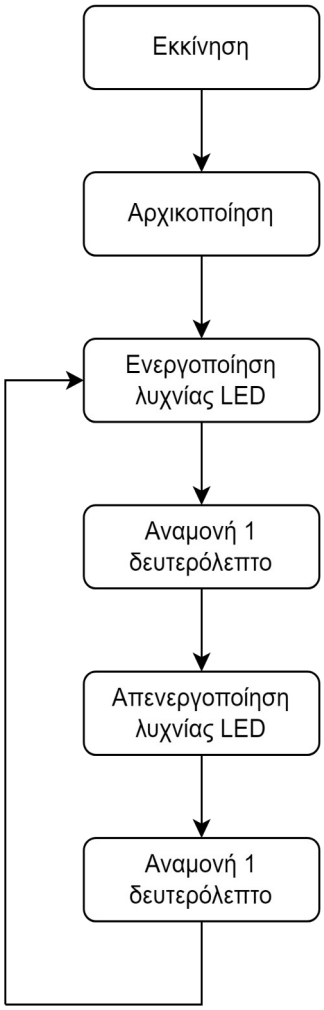
Βήμα 4: Αναμονή 1 δευτερολέπτου

Βήμα 5: Απενεργοποίηση της κόκκινης λυχνίας

Βήμα 6: Αναμονή 1 δευτερολέπτου

Βήμα 7: Επιστροφή στο βήμα 3

Για την υλοποίηση της εφαρμογής είναι απαραίτητα δύο βασικά εργαλεία. Ένα εργαλείο για έλεγχο του ακροδέκτη που συνδέεται με τη λυχνία, και ένα εργαλείο για εφαρμογή της αναμονής στο πρόγραμμα. Και τα δύο αυτά εργαλεία προσφέρονται από το λογισμικό πακέτο BSP με τη μορφή συναρτήσεων. Παρακάτω αποτυπώνεται αναλυτικά η υλοποίηση των βημάτων της εφαρμογής.



Εικόνα 3.1: Εφαρμογή Blink LED: Διάγραμμα Ροής

Το βήμα 1, αφορά την εκκίνηση του προγράμματος η οποία συμβαίνει μόλις η κάρτα συνδεθεί με την πηγή τάσης.

Το βήμα 2 αναφέρεται στην αρχικοποίηση μιας δομής τύπου **bsp_leds_t** και όνομα **leds**, η οποία είναι μέρος του λογισμικού πακέτου BSP και επιτρέπει την χρήση των λυχνιών της κάρτας ανάπτυξης.

Στο βήμα 3, η ενεργοποίηση της κόκκινης λυχνίας γίνεται, μέσω του API, με κλήση της συνάρτησης **pinWrite()**, η οποία δέχεται δύο ορίσματα. Το πρώτο είναι η λυχνία προς ενεργοποίηση, μέσω του πίνακα που περιέχει όλες τις λυχνίες της κάρτας, στη δομή leds. Το δεύτερο είναι μια τιμή που αντιστοιχεί στο επίπεδο της τάσης του ακροδέκτη και κατά επέκταση και της λυχνίας. Η τιμή πρέπει να είναι είτε 0 είτε 1, με το 0 να αντιστοιχεί σε μηδενική τάση ακροδέκτη (ground) ενώ το 1 να αντιστοιχεί σε τάση τροφοδοσίας στον ακροδέκτη.

Εδώ σημειώνεται ότι η λειτουργία της λυχνίας είναι ανεστραμμένη σε σχέση με την τιμή αυτή, δηλαδή η τιμή 0 αντιστοιχεί σε ενεργή λυχνία, ενώ η τιμή 1 αντιστοιχεί σε ανενεργή. Αυτό οφείλεται στη συνδεσμολογία της λυχνίας η οποία διαθέτει “pull up” αντίσταση.

Το βήμα 4, εισάγει μια αναμονή του ενός δευτερολέπτου κατά την οποία η λυχνία θα είναι σε ενεργή κατάσταση. Για την αναμονή εφαρμόζεται η συνάρτηση *R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS)* η οποία καθυστερεί τη ροή του προγράμματος για μια προκαθορισμένη χρονική διάρκεια. Δέχεται δύο ορίσματα, τη χρονική διάρκεια και τις μονάδες χρόνου της διάρκειας αυτής.

Στο βήμα 5, καλείται ξανά η pinWrite, με τη διαφορά ότι αυτή τη φορά με διαφορετική τιμή παραμέτρου ώστε να απενεργοποιεί τη λυχνία.

Στο βήμα 6, εφαρμόζεται και πάλι μια καθυστέρηση του προγράμματος. Οι καθυστερήσεις είναι πολύ σημαντικές καθώς χωρίς αυτές η κατάσταση της λυχνίας θα άλλαζε πολύ γρήγορα και δεν θα ήταν δυνατή η παρατήρηση της αλλαγής αυτής.

Στο βήμα 7, η ροή του προγράμματος επιστρέφει στο βήμα 3 και ο κύκλος αλλαγής της κατάστασης της λυχνίας αρχίζει από την αρχή. Εδώ παρατηρείται η δράση της εντολής while(1).

Η σημαντική διαφορά που παρουσιάζει ο κώδικας σε σχέση με τα κλασικά προγράμματα μη ενσωματωμένων συστημάτων είναι η παρουσία της ατέρμονης επαναληπτικής διαδικασίας (loop) μέσω της εντολής “while(1)”. Η εντολή αυτή συναντάται συχνά στα ενσωματωμένα συστήματα και σε προγράμματα μικροεπερξεγαστών καθώς το εκάστοτε πρόγραμμα πρέπει να εκτελείται όσο το σύστημα βρίσκεται σε λειτουργία. Έτσι η κύρια λειτουργία του προγράμματος θα βρίσκεται μέσα σε αυτή τη while ώστε να επαναλαμβάνεται συνεχώς όσο η κάρτα είναι σε λειτουργία.

Ο πλήρης κώδικας για την εφαρμογή αποτυπώνεται στο παράρτημα και υλοποιεί το παραπάνω διάγραμμα ροής.

Κεφάλαιο 4: Εφαρμογή 2 - I/O Ports

4.1 Εισαγωγή

Η εφαρμογή “I/O Ports” στοχεύει στην εξοικείωση με τις θύρες εισόδου/εξόδου και την διαδικασία επεξεργασίας εισερχόμενων σημάτων στον μικροελεγκτή. Η εφαρμογή χρησιμοποιεί τους διακόπτες S4 και S5 της κάρτας, οι οποίοι είναι συνδεδεμένοι με ακροδέκτες του μικροελεγκτή. Με το πάτημα κάποιου διακόπτη ανιχνεύεται η αλλαγή τάσης και η εφαρμογή ενεργοποιεί κατάλληλα κάποια από τις λυχνίες της κάρτας.

4.2 Ανάπτυξη και υλοποίηση

Η λειτουργία της εφαρμογής “I/O Ports” περιγράφεται από τα ακόλουθα βήματα.

Βήμα 1: Εκκίνηση του προγράμματος

Βήμα 2: Αρχικοποίηση μεταβλητών και ορισμός λειτουργίας ακροδεκτών

Βήμα 3: Έλεγχος του επιπέδου τάσης των διακοπτών

Βήμα 4: Ενεργοποίηση της κατάλληλης λυχνίας

Βήμα 5: Επιστροφή στο βήμα 3

Για την υλοποίηση της εφαρμογής είναι απαραίτητος ο έλεγχος των ακροδεκτών που αντιστοιχούν στις λυχνίες και στους διακόπτες. Ο έλεγχος αυτός γίνεται μέσω των συναρτήσεων **read** και **write** που προσφέρονται από το λογισμικό πακέτο για τον μικροελεγκτή. Παρακάτω αποτυπώνεται αναλυτικά η υλοποίηση των βημάτων της εφαρμογής.

Το βήμα 1, αφορά την εκκίνηση του προγράμματος η οποία συμβαίνει μόλις η κάρτα συνδεθεί με την πηγή τάσης.

Στο βήμα 2 ορίζονται δύο μεταβλητές στις οποίες θα αποθηκεύεται το επίπεδο τάσης των διακοπτών. Ακόμη ορίζεται η λειτουργία του κάθε ακροδέκτη που χρησιμοποιείται στην εφαρμογή, δηλαδή ορίζεται ως ακροδέκτης εισόδου ή εξόδου. Ο ορισμός ενός ακροδέκτη ως εισόδου ή εξόδου γίνεται μέσω της συνάρτησης **pinDirectionSet**(), η οποία δέχεται ως όρισμα τον macro ορισμό ενός ακροδέκτη, και τον macro ορισμό για την επιθυμητή λειτουργία του ακροδέκτη αυτού. Στη προκειμένη περίπτωση, οι ακροδέκτες που συνδέονται με τους διακόπτες της κάρτας ορίζονται ως ακροδέκτες εισόδου, ενώ οι ακροδέκτες που συνδέονται με τις λυχνίες, ορίζονται ως ακροδέκτες εξόδου. Σημειώνεται,

ακόμη, πως η αντιστοίχιση των συσκευών με τους ακροδέκτες στους οποίους συνδέονται είναι η ακόλουθη:

- Led1: port6, pin0 (IOPORT_PORT_06_PIN_00)
- Led2: port6, pin1 (IOPORT_PORT_06_PIN_01)
- Button S4: port0, pin6 (IOPORT_PORT_00_PIN_06)
- Button S5: port5, pin5 (IOPORT_PORT_00_PIN_05)

Στο βήμα 3 ελέγχεται το επίπεδο της τάσης κάθε διακόπτη. Για τον έλεγχο χρησιμοποιείται η συνάρτηση **pinRead**(), η οποία δέχεται 2 ορίσματα. Το πρώτο είναι ένας macro ορισμός του ακροδέκτη που ελέγχεται, και το δεύτερο είναι η διεύθυνση μιας μεταβλητής στην οποία θα αποθηκευτεί το επίπεδο τάσης του ακροδέκτη. Η **pinRead**() χρησιμοποιείται δύο φορές ώστε να “διαβαστεί” το επίπεδο τάσης σε κάθε διακόπτη και να αποθηκευτεί στις μεταβλητές που ορίστηκαν στο προηγούμενο βήμα.

Στο βήμα 4, ανάλογα με το επίπεδο τάσης του κάθε διακόπτη, γίνεται η ενεργοποίηση της κατάλληλης λυχνίας μέσω της συνάρτησης **pinWrite()**. Πιο συγκεκριμένα αν έχει πατηθεί ο διακόπτης S4 (επίπεδο τάσης 0) τότε ενεργοποιείται η λυχνία LED1, αν έχει πατηθεί ο διακόπτης S5 τότε ενεργοποιείται η λυχνία LED2.

Στο βήμα 5, η ροή του προγράμματος επιστρέφει στο βήμα 3.

Ο πλήρης κώδικας για την εφαρμογή αποτυπώνεται στο παράρτημα.

Κεφάλαιο 5: Εφαρμογή 3 – Interrupts

5.1 Εισαγωγή

Η εφαρμογή “Interrupts” στοχεύει στην χρήση και εξοικείωση με τη μονάδα διακοπών του μικροελεγκτή και τη διαδικασία διαχείρισης διακοπών σε μια εφαρμογή. Η εφαρμογή χρησιμοποιεί τις λυχνίες LED1 και LED2 καθώς και τον διακόπτη S5 της κάρτας. Η εφαρμογή θα αναβοσβήνει τη μια από αυτές τις λυχνίες, με περίοδο ενός δευτερολέπτου, ενώ κάθε φορά που θα αλλάζει η κατάσταση του διακόπτη S5, θα αλλάζει και η λυχνία που αναβοσβήνει. Η εναλλαγή μεταξύ των δύο λυχνιών θα γίνεται με interrupt που πυροδοτούνται από τον διακόπτη S5.

5.2 Εισαγωγικές έννοιες στα interrupt

Τα interrupt όπως δηλώνει και το όνομα, αναλαμβάνουν τη διακοπή της εκτέλεσης ενός προγράμματος ή μιας διεργασίας και την αίτηση για άμεση εκτέλεση κάποιας άλλης από τον επεξεργαστή. Με την χρήση των interrupt είναι δυνατό να ειδοποιηθεί ο επεξεργαστής ακόμα και όταν εκτελεί κάποια διεργασία, καθώς και να τεθεί να εκτελέσει κάποια διεργασία μεγαλύτερης προτεραιότητας.

Για να γίνει πιο προφανής η σημαντικότητα των interrupt, ας εξεταστεί η εφαρμογή “I/O Ports” της προηγούμενης ενότητας. Η εφαρμογή αυτή, αντιδρά στο πάτημα ενός διακόπτη από το χρήστη και ενεργοποιεί την κατάλληλη λυχνία LED. Η εφαρμογή διαθέτει μια επαναληπτική διαδικασία, η οποία ελέγχει την τάση των ακροδεκτών. Αυτός ο περιοδικός έλεγχος της κατάστασης μιας μονάδας ονομάζεται μέθοδος “polling” και αποτελεί μια χρήσιμη τεχνική, η οποία δεν χρειάζεται κάποια επιπρόσθετη περιφερειακή μονάδα για να υλοποιηθεί. Η τεχνική polling μπορεί να γίνει πολύ προβληματική όταν η περίοδος ελέγχου, δηλαδή το χρονικό διάστημα μεταξύ δύο αλλεπάλληλων ελέγχων, είναι πολύ μεγάλη. Όταν η περίοδος ελέγχου είναι μεγάλη, τότε μπορεί να προκύψει το σενάριο στο οποίο υπάρχει αλλαγή της κατάστασης (του διακόπτη) αλλά όχι ανίχνευση της ίδιας και επομένως ούτε και κατάλληλη αντίδραση από τον επεξεργαστή. Σε συστήματα πραγματικού χρόνου, όπως για παράδειγμα ιατρικές εφαρμογές, αυτή η καθυστέρηση στην ανίχνευση και αντίδραση μπορεί να είναι καταστροφική. Με χρήση των interrupt, η αλλαγή κατάστασης πυροδοτεί μια αίτηση διακοπής στον επεξεργαστή, ο οποίος αντιδρά άμεσα σε αυτή (με καθυστέρηση μερικών επεξεργαστικών κύκλων, ανάλογα την αρχιτεκτονική). Τα interrupt λοιπόν, είναι απαραίτητα σε εφαρμογές που απαιτούν ακρίβεια και αμεσότητα.

Τα interrupt μπορούν να πυροδοτηθούν, είτε από εσωτερικές περιφερειακές συσκευές του μικροελεγκτή, είτε από εξωτερικές, οι οποίες θα συνδέονται στους ακροδέκτες που γεννούν αιτήσεις για διακοπή (IRQ pin). Τη διαχείριση των αιτήσεων αυτών αναλαμβάνει η μονάδα διαχείρισης διακοπών, η οποία είναι σχεδιασμένη να ειδοποιεί τον επεξεργαστή

για διακοπές λαμβάνοντας υπ' όψιν τη σειρά και τη προτεραιότητα τους. Την εξυπηρέτηση των αιτήσεων διακοπής αναλαμβάνει η ρουτίνα ISR (Interrupt Service Routine) η οποία καλείται από την κεντρική μονάδα επξεργασίας (CPU). Μια τυπική διαδικασία εξυπηρέτησης μιας αίτησης για διακοπή είναι η ακόλουθη. Η CPU δέχεται την αίτηση από την μονάδα διακοπών και κατόπιν διακόπτει την εκτέλεση του τρέχοντος προγράμματος και αποθηκεύει όλες τις απαραίτητες πληροφορίες για αυτό στη μνήμη (program counter, δεδομένα κλπ). Στη συνέχεια, εκτελεί το πρόγραμμα που σχετίζεται με την αίτηση διακοπής. Με την ολοκλήρωση του προγράμματος διακοπής γίνεται επαναφορά των πληροφοριών του αρχικού προγράμματος και συνεχίζεται η εκτέλεση του.

5.3 Ανάπτυξη και υλοποίηση

Το κύριο μέρος της εφαρμογής αποτελείται από μια επαναληπτική διαδικασία στην οποία θα αναβοσβήνει μια εκ των λυχνιών. Πιο συγκεκριμένα, ορίζεται μια μεταβλητή level η οποία αλλάζει από 0 σε 1 ή ανάποδα ανάλογα την προηγούμενη της κατάσταση και σύμφωνα με τη μεταβλητή αυτή θα αλλάζει και η τάση στη λυχνία LED. Η ενεργοποίηση ή απενεργοποίηση της λυχνίας θα γίνεται μέσω της συνάρτησης *"g_ioport.p_api→pinWrite(leds.p_leds[led_selection_var], level)"* που δέχεται σαν ορίσματα ένα πίνακα που περιέχει τις λυχνίες LED και τη μεταβλητή level. Για τη δημιουργία περιόδου ενός δευτερολέπτου χρησιμοποιείται η συνάρτηση *"R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS)"* η οποία καθυστερεί την επαναληπτική διαδικασία και κατά συνέπεια το χρόνο που θα είναι αναμμένη ή σβηστή η λυχνία LED. Για την επιλογή της λυχνίας που θα αναβοσβήνει ορίζεται η μεταβλητή led_selection_var η οποία παίρνει τιμές 0 ή 1 και αλλάζει κάθε φορά που πυροδοτείται ένα interrupt. Για την πυροδότηση των interrupt ρυθμίζεται ο ακροδέκτης IRQ10, ο οποίος συνδέεται με τον διακόπτη S5, να ανιχνεύει interrupt κατά την αλλαγή σε χαμηλή τάση (falling edge), δηλαδή με το πάτημα του διακόπτη. Κατά την ανίχνευση μιας τέτοιας αλλαγής τάσης η ρουτίνα εξυπηρέτησης ISR, καλεί την ρουτίνα ext_irq10_callback η οποία αλλάζει τη τιμή της μεταβλητής led_selection_var από 0 σε 1 ή από 1 σε 0.

Για την ρύθμιση της λειτουργίας της μονάδας διαχείρισης διακοπών χρησιμοποιείται το γραφικό περιβάλλον του e²studio το οποίο επιτρέπει την επιλογή ρυθμίσεων, όπως τη λειτουργία ανίχνευσης και την ενεργοποίηση του ψηφιακού φίλτρου. Ο απαραίτητος για τις ρυθμίσεις αυτός κώδικας παράγεται αυτόματα από το e²studio και οι ρυθμίσεις αποθηκεύονται σε δομές, οι οποίες θα αποτελέσουν ορίσματα για την συνάρτηση *"g_external_irq10.p_api→open(g_external_irq10.p_ctrl,g_external_irq10.p_cfg)".* Η συνάρτηση αυτή ενεργοποιεί τα interrupt και τη χρήση της μονάδας ICU στην εφαρμογή.

Ο πλήρης κώδικας της εφαρμογής αποτυπώνεται στο παράρτημα.

Κεφάλαιο 6: Εφαρμογή 4 – Timers

6.1 Εισαγωγή

Η εφαρμογή “Timers” στοχεύει στην εξοικείωση με τη μονάδα GPT, η οποία διαθέτει 14 κανάλια χρονιστών-απαριθμητών με το καθένα να περιέχει έναν μετρητή των 32 bit. Η εφαρμογή χρησιμοποιεί την μονάδα για τη δημιουργία δύο διαφορετικών χρονικών περιόδων. Με τη λήξη της κάθε χρονικής περιόδου καλούνται συνάρτησεις οι οποίες ελέγχουν τη λειτουργία των λυχνιών LED1 και LED2.

6.2 Ανάπτυξη και υλοποίηση

Η εφαρμογή χρησιμοποιεί δύο κανάλια της μονάδας GPT, όπου το καθένα μετράει διαφορετική χρονική διάρκεια. Το πρώτο κανάλι, με όνομα “g_timer0”, μετράει χρονική διάρκεια ενός δευτερολέπτου και η λειτουργία του ορίζεται να είναι “one-shot”, δηλαδή μετράει μόνο μια φορά. Με το πέρας της χρονικής διάρκειας του ενός δευτερολέπτου, καλείται και εκτελείται η συνάρτηση “timer0_callback()” η οποία αναλαμβάνει την ενεργοποίηση της λυχνίας LED1.

Το δεύτερο κανάλι, με όνομα “g_timer1”, μετράει χρονική διάρκεια 3 δευτερόλεπτα και η λειτουργία του ορίζεται να είναι “periodic”, δηλαδή η μέτρηση θα επαναλαμβάνεται, και κάθε φορά που λήγει η χρονική περίοδος ο μετρητής ξεκινάει εκ νέου τη μέτρηση. Ακόμη, κάθε φορά που θα λήγει η χρονική περίοδος θα καλείται η συνάρτηση “timer1_callback()” η οποία θα ενεργοποιεί ή απενεργοποιεί τη λυχνία LED2, ανάλογα με την προηγούμενη της κατάσταση. Με αυτό τον τρόπο, πετυχαίνεται πρακτικά η λειτουργία της εφαρμογής “Blink LED”, μόνο που σε αυτή την περίπτωση δεν επιβαρύνεται ο επεξεργαστής και δεν υπάρχει αναμονή που καθυστερεί τη λειτουργία του ίδιου.

Οι ρυθμίσεις του κάθε καναλιού αποτυπώνονται στους παρακάτω πίνακες και ο πλήρης κώδικας για την εφαρμογή αποτυπώνεται στο παράρτημα.

g_timer0 Timer Driver on r_gpt

Settings | API Info

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_timer0 Timer Driver on r_gpt	
Name	g_timer0
Channel	0
Mode	One Shot
Duty Cycle Range (only applicable in PWM mode)	Shortest: 2 PCLK, Longest: (Period - 1) PCLK
Period Value	1
Period Unit	Seconds
Duty Cycle Value	50
Duty Cycle Unit	Unit Raw Counts
Auto Start	True
GTIOCA Output Enabled	False
GTIOCA Stop Level	Pin Level Low
GTIOCB Output Enabled	False
GTIOCB Stop Level	Pin Level Low
Callback	timer0_callback
Overflow Interrupt Priority	Priority 4

g_timer1 Timer Driver on r_gpt

Settings | API Info

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_timer1 Timer Driver on r_gpt	
Name	g_timer1
Channel	1
Mode	Periodic
Duty Cycle Range (only applicable in PWM mode)	Shortest: 2 PCLK, Longest: (Period - 1) PCLK
Period Value	3
Period Unit	Seconds
Duty Cycle Value	50
Duty Cycle Unit	Unit Raw Counts
Auto Start	True
GTIOCA Output Enabled	False
GTIOCA Stop Level	Pin Level Low
GTIOCB Output Enabled	False
GTIOCB Stop Level	Pin Level Low
Callback	timer1_callback
Overflow Interrupt Priority	Priority 4

Κεφάλαιο 7: Εφαρμογή 5 – Παιχνίδι χρόνου απόκρισης

7.1 Εισαγωγή

Στην ενότητα αυτή θα αναπτυχθεί εφαρμογή, η οποία θα μετράει το χρόνο απόκρισης του χρήστη σε ένα οπτικό ερέθισμα. Το οπτικό ερέθισμα στην προκειμένη περίπτωση θα είναι μια λυχνία LED, η οποία θα ανάβει και ο χρήστης θα πρέπει να πατήσει όσο πιο γρήγορα γίνεται το διακόπτη ώστε να μετρηθεί ο χρόνος απόκρισης του. Πιο συγκεκριμένα, με την ενεργοποίηση της εφαρμογής θα ενεργοποιείτε και ένας χρονιστής ο οποίος θα μετράει χρονική διάρκεια ενός δευτερολέπτου. Με το πέρας της χρονικής διάρκειας αυτής θα ανάβει μια λυχνία LED και θα ενεργοποιείται πάλι ο χρονιστής ο οποίος θα μετράει έως και τρία δευτερόλεπτα. Σε αυτό το διάστημα ο χρήστης θα πρέπει να πατήσει το διακόπτη. Με το πάτημα θα πυροδοτείται ένα interrupt το οποίο θα σηματοδοτεί το σβήσιμο της λυχνίας, την παύση του χρονιστή και την αποθήκευση του χρόνου απόκρισης σε μια μεταβλητή. Για την ανάπτυξη και υλοποίηση της ακόλουθης εφαρμογής θα γίνει χρήση της μονάδας ICU, της μονάδας GPT, των λυχνιών LED και του διακόπτη S4. Στόχος της εφαρμογής αυτής είναι να αναδείξει τη δυνατότητα υλοποίησης εφαρμογών που συνδυάζουν δύο ή και περισσότερες περιφερειακές συσκευές.

7.2 Ανάπτυξη και υλοποίηση

Η λειτουργία της εφαρμογής περιγράφεται από τα ακόλουθα βήματα.

Βήμα 1: Εκκίνηση του προγράμματος

Βήμα 2: Αρχικοποίηση δομών και περιφερειακών συσκευών

Βήμα 3: Αναμονή 1 δευτερολέπτου

Βήμα 4: Ενεργοποίηση μιας εκ των λυχνιών που θα λειτουργήσει ως οπτικό ερέθισμα και επαναφορά του χρονιστή

Βήμα 5: Αναμονή μέχρι ο χρήστης να πατήσει το διακόπτη

Βήμα 6: Με το πάτημα του διακόπτη, καταγράφεται η τιμή του timer και υπολογίζεται ο χρόνος απόκρισης

Για την υλοποίηση της εφαρμογής ρυθμίζονται κατάλληλα οι περιφερειακές μονάδες ICU και GPT, σύμφωνα με τους παρακάτω πίνακες.

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_external_irq11 External IRQ Driver on r_icu	
Name	g_external_irq11
Channel	11
Trigger	Rising
Digital Filtering	Disabled
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK / 64
Interrupt enabled after initialization	True
Callback	switch_callback
Pin Interrupt Priority	Priority 5

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_timer0 Timer Driver on r_gpt	
Name	g_timer0
Channel	0
Mode	One Shot
Duty Cycle Range (only applicable in PWM mode)	Shortest: 2 PCLK, Longest: (Period - 1) PCLK
Period Value	1
Period Unit	Seconds
Duty Cycle Value	50
Duty Cycle Unit	Unit Raw Counts
Auto Start	True
GTIOCA Output Enabled	False
GTIOCA Stop Level	Pin Level Low
GTIOCB Output Enabled	False
GTIOCB Stop Level	Pin Level Low
Callback	timer0_callback
Overflow Interrupt Priority	Priority 4

Παρακάτω αποτυπώνονται αναλυτικά τα βήματα της εφαρμογής.

Το βήμα 1, αφορά την εκκίνηση του προγράμματος η οποία συμβαίνει μόλις η κάρτα συνδεθεί με την πηγή τάσης.

Στο βήμα 2, αρχικοποιείται η δομή για τις λυχνίες καθώς και οι μονάδες ICU και GPT. Ακόμη απενεργοποιούνται όλες οι λυχνίες της κάρτας και ξεκινάει η μέτρηση από τον χρονιστή. Για την εκκίνηση του χρονιστή χρησιμοποιείται η συνάρτηση **start()**.

Στο βήμα 3, η ροή του προγράμματος παγιδεύεται σε μια while και παραμένει σε αυτή μέχρι την αλλαγή της μεταβλητής time_expired. Η μεταβλητή αυτή αλλάζει από την συνάρτηση timer0_callback(), η οποία καλείται με τη λήξη της χρονικής διάρκειας που μετράει ο χρονιστής.

Στο βήμα 4 ενεργοποιείται η λυχνία LED0, η οποία θα λειτουργήσει ως το οπτικό ερέθισμα. Ακόμη γίνεται επαναφορά του χρονιστή και ρυθμίζεται να μετρήσει χρονική διάρκεια έως και 3 δευτερόλεπτα. Για την επαναφορά και τον ορισμό της καινούργιας περιόδου χρησιμοποιούνται οι συναρτήσεις **reset()** και **periodSet()**.

Στο βήμα 5 η ροή του προγράμματος παγιδεύεται ξανά σε μια while και παραμένει σε αυτή μέχρι να αλλάξει η τιμή της μεταβλητής button_pressed. Η μεταβλητή αυτή αλλάζει στη συνάρτηση switch_callback() η οποία καλείται όταν πατηθεί ο διακόπτης και πυροδοτηθεί μια αίτηση για διακοπή.

Στο βήμα 6, και αφού μόλις έχει πατηθεί ο διακόπτης, καταγράφεται η τιμή που έχει ο χρονιστής μέσω της συνάρτησης **counterGet()**, η οποία αποθηκεύει την τιμή του χρονιστή στη μεταβλητή value. Ο χρονιστής μετράει κύκλους ρολογιού, με το κάθε κύκλο να διαρκεί 8.33 ns. Πολλαπλασιάζοντας την τιμή του χρονιστή με την περίοδο κάθε κύκλου, υπολογίζεται ο συνολικός χρόνος απόκρισης του χρήστη και αποθηκεύεται στη μεταβλητή response_time.

Ο πλήρης κώδικας της εφαρμογής αποτυπώνεται στο παράρτημα.

Κεφάλαιο 8: Εφαρμογή 6 – PWM

8.1 Εισαγωγή

Η εφαρμογή “PWM” στοχεύει στην εξοικείωση με τη διαδικασία δημιουργίας σημάτων PWM της μονάδας GPT. Σε αντίθεση με τις προηγούμενες εφαρμογές, δεν θα γίνει χρήση του HAL και των συναρτήσεων του, αλλά η ανάπτυξη της εφαρμογής θα γίνει με εξ΄ ολοκλήρου χρήση των καταχωρητών ελέγχου της μονάδας GPT.

8.2 Ανάπτυξη και υλοποίηση

Για την εφαρμογή δημιουργείται σήμα PWM με συχνότητα 1KHz και duty cycle 50%, ενώ για έξοδο του σήματος επιλέγεται ο ακροδέκτης p107 της κάρτας. Για την δημιουργία του σήματος είναι απαραίτητη η κατάλληλη ρύθμιση των πεδίων των αντίστοιχων καταχωρητών. Οι καταχωρητές αυτοί είναι “memory mapped”, δηλαδή υπάρχουν δεσμευμένες θέσεις μνήμης για κάθε έναν από αυτούς. Η εγγραφή στα πεδία ενός καταχωρητή γίνεται με εγγραφή στη διεύθυνση της μνήμης που αντιστοιχεί στον καταχωρητή αυτό, ενώ για την εγγραφή στις διευθύνσεις μνήμης ορίζονται δείκτες.

Η διαδικασία δημιουργίας του σήματος με εγγραφή στους καταχωρητές είναι η ακόλουθη.

- Αρχικά, επιλέγεται και ενεργοποιείται το κανάλι 8 της μονάδας GPT, γράφοντας την τιμή ‘0’ στο bit 6 του καταχωρητή **MSTPCRD** (module stop control register D). Ο καταχωρητής αυτός βρίσκεται στη διεύθυνση 0x40047008 της μνήμης και είναι υπεύθυνος για την ενεργοποίηση και απενεργοποίηση μονάδων με στόχο τη μικρότερη κατανάλωση ισχύος στον μικροελεγκτή.
- Το κανάλι ρυθμίζεται με τέτοιο τρόπο ώστε ο μετρητής της μονάδας να ξεκινάει από το 0 και να αυξάνεται. Η ρύθμιση γίνεται γράφοντας την τιμή ‘1’ στο bit 0 του καταχωρητή **GTUDDTYC** (general PWM timer count direction and duty setting register), ο οποίος βρίσκεται στη διεύθυνση 0x40078830.
- Στη συνέχεια ρυθμίζεται ο καταχωρητής **GTPR** (general PWM timercycle setting register), ο οποίος βρίσκεται στη διεύθυνση 0x40078864 και περιέχει τον αριθμό των κύκλων που αντιστοιχούν στην επιθυμητή περίοδο του σήματος. Στη συγκεκριμένη περίπτωση για περίοδο 1 ms και δεδομένου ότι το ρολόι της μονάδας έχει συχνότητα λειτουργίας 120 MHz, αντιστοιχούν 120000 – 1 κύκλοι. Δηλαδή ο καταχωρητής θα έχει την τιμή 119999 ή 0x1D4BF.
- Στη συνέχεια ρυθμίζεται ο καταχωρητής **GTCCRA** (general PWM timer compare capture register A), ο οποίος βρίσκεται στη διεύθυνση 0x4007884c και περιέχει τον

αριθμό κύκλων που αντιστοιχούν στο duty cycle. Στη προκειμένη περίπτωση ο καταχωρητής περιέχει την τιμή 0xEA5F, η οποία αντιστοιχεί σε duty cycle 50%.

- Όσον αφορά στη λειτουργία εξόδου του καναλιού, πρέπει αρχικά η έξοδος να βρίσκεται στη κατάσταση '1', μετά να μηδενίζεται όταν ο μετρητής φτάσει την τιμή στον καταχωρητή GTCCRA (duty cycle) και τέλος να επανέρχεται στη κατάσταση '1' με τη λήξη της περιόδου PWM (1 ms). Για τη λειτουργία αυτή ρυθμίζονται κατάλληλα τα πεδία OAE και GTIOA του καταχωρητή **GTIOR** (general PWM timer I/O control register), η πρόσβαση στον οποίο γίνεται μέσω της διεύθυνσης 0x40078834.
- Για την εκκίνηση του μετρητή στο κανάλι 8 ρυθμίζεται με την τιμή '1' το πεδίο CST του καταχωρητή **GTCR** (general PWM timer control register), που έχει διεύθυνση μνήμης 0x4007882c.
- Τέλος για τη σύνδεση της εξόδου του καναλιού με τον ακροδέκτη p107, ρυθμίζονται κατάλληλα οι καταχωρητές **PWPR** (write protect register) και **P107PFS** (port 107 pin function select register) οι οποίοι βρίσκονται στις διευθύνσεις 0x40040d03 και 0x4004085C, αντίστοιχα. Ο PWPR ρυθμίζεται με την τιμή 0x40 και ενεργοποιεί την εγγραφή σε καταχωρητές που ελέγχουν τη λειτουργία των ακροδεκτών. Για τον P107PFS, που ορίζει τη λειτουργία του ακροδέκτη και τον συνδέει με την έξοδο της μονάδας, ρυθμίζονται το bit 16 με την τιμή '1', και τα bit 28-24 με την τιμή '0b0011'.

Ο πλήρης κώδικας της εφαρμογής αποτυπώνεται στο παράρτημα.

Βιβλιογραφία

Renesas Electronics Corporation. S7G2 Microcontroller Group Datasheet

Renesas Electronics Corporation. S7G2 Microcontroller Group User's Manual

Renesas Electronics Corporation. Renesas Synergy Starter Kit SK_S7G2 User's Manual

Renesas Electronics Corporation. Renesas Synergy Software Package User's Manual

Douglas Renaux, Robson Linhares 2020. Embedded Systems. UTFPR,esystech,Renesas Electronics Corporation

Joseph Yiu. Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors

Wikipedia. https://en.wikipedia.org/wiki/Microprocessor

Wikipedia. https://en.wikipedia.org/wiki/Microcontroller

Παράρτημα

Εφαρμογή 1 – Blink LED

```
#include "hal_data.h"
void hal_entry(void)
{

    bsp_leds_t Leds;
    R_BSP_LedsGet(&Leds);

    while(1)
    {

        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED1], 1);


        R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);


        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED1], 0);


        R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);

    }
}
```

Εφαρμογή 2 – I/O Ports

```
#include "hal_data.h"



void hal_entry(void)
{
    ioport_level_t button_S4, button_S5;


    //Input
    g_ioport.p_api->pinDirectionSet(IOPORT_PORT_00_PIN_06,IOPORT_DIRECTION_INPUT);
    g_ioport.p_api->pinDirectionSet(IOPORT_PORT_00_PIN_05,IOPORT_DIRECTION_INPUT);
    //Output
    g_ioport.p_api-
>pinDirectionSet(IOPORT_PORT_06_PIN_00,IOPORT_DIRECTION_OUTPUT);
    g_ioport.p_api-
>pinDirectionSet(IOPORT_PORT_06_PIN_01,IOPORT_DIRECTION_OUTPUT);


    while(1)
    {
        g_ioport.p_api->pinRead(IOPORT_PORT_00_PIN_06, &button_S4);
        g_ioport.p_api->pinRead(IOPORT_PORT_00_PIN_05, &button_S5);


        if (button_S4 == 0)
        {
            g_ioport.p_api->pinWrite(IOPORT_PORT_06_PIN_01,0);
        }
        if (button_S5 == 0)
        {
            g_ioport.p_api->pinWrite(IOPORT_PORT_06_PIN_02,0);
        }
    }
}
```

```
#include "hal_data.h"


volatile uint32_t led_selection_var = 0;

void hal_entry(void)
{

    /* LED type structure */
    bsp_leds_t leds;

    /* Get LED information for this board */
    R_BSP_LedsGet(&leds);

    /* LED state variable */
    ioport_level_t level = IOPORT_LEVEL_HIGH;

    /*open external irq10*/
    g_external_irq10.p_api->open(g_external_irq10.p_ctrl,g_external_irq10.p_cfg);

    /*turn off LEDs*/
    for (uint32_t i = 0; i < leds.led_count; i++)
    {
        g_ioport.p_api->pinWrite(leds.p_leds[i],IOPORT_LEVEL_HIGH);
    }

    while(1)
    {
        /* change LED state*/
        if (level == IOPORT_LEVEL_HIGH)
        {
```

```
            level = IOPORT_LEVEL_LOW;
        }
        else
        {
            level = IOPORT_LEVEL_HIGH;
        }


        g_ioport.p_api->pinWrite(leds.p_leds[led_selection_var],level);


        /*Delay*/
        R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS);

    }

}


void ext_irq10_callback(external_irq_callback_args_t *p_args)
{
    if (led_selection_var == 1)
    {
        led_selection_var = 0;
    }
    else
    {
        led_selection_var = 1;
    }
}
```

Εφαρμογή 4 – Timers

```c
#include "hal_data.h"
void hal_entry(void)
{

    bsp_leds_t Leds;
    R_BSP_LedsGet(&Leds);
    volatile int level = 0;

    g_timer0.p_api->open(g_timer0.p_ctrl,g_timer0.p_cfg);
    g_timer1.p_api→open(g_timer1.p_ctrl,g_timer1.p_cfg);

    g_timer0.p_api→start(g_timer0.p_ctrl);
    g_timer1.p_api→start(g_timer1.p_ctrl);

    while(1)
    {
    }
}

void timer0_callback(timer_callback_args_t * p_args)
{
    g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED1], 0);
}

void timer1_callback(timer_callback_args_t * p_args)
{
    //Change level
    if (level == 0)
    {
        level = 1;
    }
    else
```

```
    {
        level = 0;
    }

    g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED2], level);
}
```

Εφαρμογή 5 – Παιχνίδι χρόνου απόκρισης

```
#include <stdio.h>
#include "hal_data.h"


volatile uint32_t button_pressed = 0;
volatile uint32_t time_expired = 0;
uint32_t value = 0;
float response_time = 0.0;
extern void initialise_monitor_handles(void);

void hal_entry(void)
{
    initialise_monitor_handles();

    /* LED type structure */
    bsp_leds_t leds;

    /* LED state variable */
    ioport_level_t level = IOPORT_LEVEL_HIGH;

    /* Get LED information for this board */
    R_BSP_LedsGet(&leds);


    /* turn off LEDs*/
    for(uint32_t i = 0; i < leds.led_count; i++)
    {
        g_ioport.p_api->pinWrite(leds.p_leds[i], IOPORT_LEVEL_HIGH);
    }

    g_external_irq11.p_api->open(g_external_irq11.p_ctrl,g_external_irq11.p_cfg);
```

```
/* init timer*/
g_timer0.p_api->open(g_timer0.p_ctrl,g_timer0.p_cfg);
g_timer0.p_api->start(g_timer0.p_ctrl);

/* wait for 1 second */
while (time_expired == 0)
{
    //wait until flag is set
}


/* turn LEDs on */
for(uint32_t i = 0; i < leds.led_count; i++)
{
    g_ioport.p_api->pinWrite(leds.p_leds[i], IOPORT_LEVEL_LOW);
}

g_timer0.p_api->reset(g_timer0.p_ctrl);
g_timer0.p_api->periodSet(g_timer0.p_ctrl, 3, TIMER_UNIT_PERIOD_SEC);
g_timer0.p_api->start(g_timer0.p_ctrl);

/*wait for S4 to be pressed */
while (button_pressed == 0)
{
    //wait
}

g_timer0.p_api->counterGet(g_timer0.p_ctrl,&value);


response_time = (float)(value)*(float)(8.33333)*(float)(0.000000001);

printf("your time is %f", response_time);
```

```
    while(1)
    {
        /* Determine the next state of the LEDs */
        if(IOPORT_LEVEL_LOW == level)
        {
            level = IOPORT_LEVEL_HIGH;
        }
        else
        {
            level = IOPORT_LEVEL_LOW;
        }

        /* Update all board LEDs */
        for(uint32_t i = 0; i < leds.led_count; i++)
        {
            g_ioport.p_api->pinWrite(leds.p_leds[i], level);
        }

        /* Delay */
        R_BSP_SoftwareDelay(1, BSP_DELAY_UNITS_SECONDS );
    }
}

void timer0_callback(timer_callback_args_t * p_args)
{
    time_expired = 1;
}

void switch_callback(external_irq_callback_args_t * p_args)
{
    button_pressed = 1;
}
```

```
#include "hal_data.h"

void hal_entry(void) {

    volatile uint32_t * pP107PFS =  (uint32_t *) 0x4004085C;
    volatile uint8_t  * pPWPR =     (uint8_t *)  0x40040d03;
    volatile uint32_t * pMSTPCRD =  (uint32_t *) 0x40047008;
    volatile uint32_t * pGTCR =     (uint32_t *) 0x4007882c;
    volatile uint32_t * pGTUDDTYC = (uint32_t *) 0x40078830;
    volatile uint32_t * pGTIOR =    (uint32_t *) 0x40078834;
    volatile uint32_t * pGTCCRA =   (uint32_t *) 0x4007884c;
    volatile uint32_t * pGTPR =     (uint32_t *) 0x40078864;

    *pPWPR      = 0;
    *pPWPR      = 0x40;
    *pP107PFS  |= (0x3<<24 | 1<<16);
    *pMSTPCRD  &= (uint32_t) ~0x40;
    *pGTUDDTYC  = 0x1;
    *pGTIOR     = 0x119;
    *pGTCCRA    = 0xea5f;
    *pGTPR      = 0x1d4bf;

    *pGTCR = 0x1;

      /* Define the units to be used with the software delay function */
      const bsp_delay_units_t bsp_delay_units = BSP_DELAY_UNITS_MILLISECONDS;
      /* Set the blink frequency (must be <= bsp_delay_units */
    const uint32_t freq_in_hz = 2;
      /* Calculate the delay in terms of bsp_delay_units */
    const uint32_t delay = bsp_delay_units/freq_in_hz;
```

```
    /* LED type structure */
    bsp_leds_t leds;
    /* LED state variable */
    ioport_level_t level = IOPORT_LEVEL_HIGH;

    /* Get LED information for this board */
    R_BSP_LedsGet(&leds);

    while(1)
    {
        /* Determine the next state of the LEDs */
        if(IOPORT_LEVEL_LOW == level)
        {
            level = IOPORT_LEVEL_HIGH;
        }
        else
        {
            level = IOPORT_LEVEL_LOW;
        }

        /* Update all board LEDs */
        for(uint32_t i = 0; i < leds.led_count; i++)
        {
            g_ioport.p_api->pinWrite(leds.p_leds[i], level);
        }

        /* Delay */
        R_BSP_SoftwareDelay(delay, bsp_delay_units);
    }
}
```