Μικροεπεξεργαστές και Περιφερειακά

Αναφορά Εργαστηρίου 2

Ονοματεπώνυμο	ΑΕΜ	E-mail
Κωνσταντίνος Πράττης	10056	kprattis@ece.auth.gr
Βασιλική Ρίζου	10021	vasilikirn@ece.auth.gr

Εισαγωγή

Η παρούσα αναφορά αποσκοπεί στην επεξήγηση του προγράμματος σε κώδικα C που υλοποιήθηκε για τον προγραμματισμό ενός μικρο-ελεγκτή ARM, στα πλαίσια του 3ου εργαστηρίου του μαθήματος Μικροεπεξεργαστές και Περιφερειακά. Συγκεκριμένα, το πρόγραμμα ζητάει μία φορά στην αρχή την εισαγωγή του ΑΕΜ από τον χρήστης, μέσω της συνάρτησης ***uart_get_AEM*** . Κατόπιν, με την χρήση αισθητήρα DHT11, λαμβάνεται η θερμοκρασία και η υγρασία με συγκεκριμένο ρυθμό δειγματοληψίας, η οποία εκτυπώνεται μέσω σειριακής επαφής UART. Σε περίπτωση που η μετρούμενη θερμοκρασία υπερβεί τους 25°C, ανάβει το LED. Αν αυτή είναι χαμηλότερη των 20°C σβήνει το LED, και για τις ενδιάμεσες τιμές το LED αναβοσβήνει με περίοδο 1second, καλώντας την ISR routine: ***timer_isr***. Αρχικά, η περίοδος δειγματοληψίας είναι 2 second. Σε περίπτωση που πατηθεί ο διακόπτης switch, εκτελείται η ISR routine: ***switch_isr.*** Αν πατήθηκε για πρώτη φορά, τότε αλλάζει η περίοδος δειγματοληψίας και γίνεται ίση με το άθροισμα των 2 τελευταίων ψηφίων του ΑΕΜ που εισήγαγε αρχικά ο χρήστης. Διαφορετικά, για όλες τις επόμενες φορές που θα εκτελεστεί ο isr του διακόπτη, η περίοδος δειγματοληψίας θα γίνει ίσης με 3 sec αν είναι μονή η φορά και 4 sec αν είναι άρτιος ο αριθμός φορών που έχει πατηθεί ο διακόπτης.

uart_get_AEM

Input: *void*

Output: *void*

Η συγκεριμένη συνάρτηση καλείται μία φορά, κατά την αρχή της εκτέλεσης του προγράμματος και ζητάει το ΑΕΜ από τον χρήστη. Αρχικοποιείται μία ουρά (τύπου Queue*), στην οποία θα γίνονται push ένας ένας οι χαρακτήρες που εισάγει ο χρήστης για το ΑΕΜ του. Έως ότου ο εισαγόμενος χαρακτήρας c είναι διαφορετικός του *carriage return* ('\r') κάνουμε *queue_enqueue(queue, c);* . Μόλις ο χρήστης ολοκληρώσει την εισαγωγή του ΑΕΜ, δλδ c = '\r', ξεκινάμε να κάνουμε pop τα στοιχεία από την ουρά (*queue_dequeue(queue, &ci), i = {1, 2}*), και αποθηκεύονται διαδοχικά στις uint8_t c1, c2. Η διαδικασία αυτή γίνεται διαδοχικά σε 2 μεταβλητές, ούτως ώστε μόλις q*ueue_is_empty(queue) == TRUE*, και άρα σταματήσουμε να κάνουμε pop από την ουρά, οι 2 αυτές μεταβλητές να έχουν λάβει την τιμή των 2 τελευταίων ψηφίων του ΑΕΜ του χρήστη. Επιπλέον, για να λάβουμε το αλγεβρικό άθροισμα των c1, c2, αφαιρούμε την τιμή του αριθμού 0 σε ASCII, δλδ. *AEM_last_digits = c1 + c2 - 2 * (ASCII_0 = 48)*. Τέλος, σε περίπτωση του *AEM_last_digits ≤ 2*, τότε *AEM_last_digits = 4*. Και έτσι, έχουμε αποθηκεύσει την τιμή δειγματοληψίας μόλις πατηθεί ο διακόπτης για πρώτη φορά.

timer_isr

Input: *void*

Output: *void*

Έχοντας αρχικοποιήσει τον timer, έτσι ώστε να "σκάει" κάθε 1 μsec (timer_init(1e6)), κάθε 1μsec εκτελείται η ISR ρουτίνα του που χρησιμοποιήθηκε, για την περιοδική δειγματοληψία και εμφάνιση της θερμοκρασίας. Αρχικά, επειδή η περίοδος δειγματοληψίας μεταβάλλεται κατά την διάρκεια εκτέλεσης του προγράμματος, και θα πρέπει να μπορεί να τροποποιείται και από άλλες isr ρουτίνες, ορίζουμε στην αρχή 2 static μεταβλητές *static int period, static int seconds.* Η period, διατηρεί την τιμή της τρέχουσας περιόδου δειγματοληψίας (αρχικά period = 2). Κάθε φορά που καλείται η timer_isr, αυξάνεται η τιμή της μεταβλητής seconds, και ελέγχεται αν έχει περάσει χρόνος ίσος με period, ούτως ώστε να εκτυπωθούν τα ζητούμενα. Εάν *seconds == 0,* διαβάζεται η θερμοκρασία και ανανεώνεται η τιμή της υγρασίας από τον DHT11, μέσω της *temperature = DHT11_get_temperature(&humidity).* [1] Σε κάθε περίπτωση αυτή εκτυπώνεται μέσω UART. Επιπλέον, εάν temperature ≥ 25°C , τότε ανάβει το LED, μέσω της *gpio_set(LED, 1),* ενώ αν temperature ≤ 20°C , gpio_set(LED, 0), σβήνει το LED. Ανεξάρτητα αν έχει περάσει χρόνος ίσος με period, ελέγχεται αν η θερμοκρασία είναι μεταξύ 20°C και 25 °C, και αν κάτι τέτοιο ισχύει, γίνεται toggle το LED (αν προηγουμένως ήταν high, τίθεται low, και αντίστροφα).

Drivers για τον αισθητήρα DHT11

Για την επιτυχή ανάγνωση θερμοκρασίας από τον αισθητήρα υλοποιήθηκαν τα αρχεία dht11.h και dht11.c που περιέχουν τις συναρτήσεις DHT11_Start, DHT11_Read, DHT11_Check_Response και την DHT11_get_temperature.

Οι συναρτήσεις DHT11_Start, DHT11_Read, DHT11_Check_Response υλοποιήθηκαν με βάση το datasheet του αισθητήρα, όπου αναφέρεται ότι για την ανάγνωση της θερμοκρασίας αρχικά θα πρέπει να σταλεί το σήμα έναρξης από τον επεξεργαστή - low για 18ms. Στη συνέχεια, περιμένει 40 ms για την απάντηση του αισθητήρα (low 80 ms). Τέλος, ο τρόπος για να διαβαστεί ένα ένα το bit μιας 8bitης λέξης από τις 4 που πρόκειται να στείλει ο αισθητήρας βασίζεται στην διαφορετική διάρκεια αποστολής 0 και 1. Έτσι, αν μετά από 40 ms το pin του αισθητήρα είναι low τότε το bit είναι 0, αλλιώς είναι 1.

Σημειώνεται επίσης ότι καθώς ο αισθητήρας DHT11 είναι αρκετά διαδεδομένος, οι τρεις αυτές συναρτήσεις βρέθηκαν από διάφορες πηγές υλοποιημένες. Επιλέχθηκε η υλοποίηση που παρουσιάζεται εδώ ως template και πάνω σε αυτήν έγιναν οι κατάλληλες αλλαγές, ώστε να χρησιμοποιούνται οι drivers του μαθήματος και επίσης ώστε να χρησιμοποιείται το Pull-up mode κατά την ανάγνωση του αισθητήρα, σύμφωνα και με το datasheet. Για delay χρησιμοποιήθηκαν οι συναρτήσεις που έχουν δοθεί στη θεωρία του μαθήματος (delay.h - delay.c, delay_cycles.s).

Τέλος, υλοποιήθηκε και η DHT11_get_temperature η οποία χρησιμοποιεί τις άλλες τρεις συναρτήσεις ώστε να διαβάσει και να επιστρέψει τη θερμοκρασία και την υγρασία, εκτελώντας Start, Check_Response και έπειτα 4 φορές Read. Για τον συνδυασμό του ακέραιου με το δεκαδικό

[1] Να σημειωθεί ότι η μεταβλητή temperature είναι επίσης static, καθώς θα θέλαμε να διατηρεί την τιμή της και κατά την ολοκλήρωση της εκτέλεσης της isr ρουτίνας. Αυτό, γιατί, σε περίπτωση που seconds != 0, δεν θα διαβαστεί καινούργια τιμή θερμοκρασίας, παρόλα αυτά θα θέλαμε, σε περίπτωση που temperature $\in (20,25)$ να συνεχίσει να αναβοσβήνει το LED.

μέρος, εφαρμόστηκε για την θερμοκρασία απλά ο τύπος: $T = T_{integer} + \frac{T_{decimal}}{10}$, ενώ ο ίδιος τύπος εφαρμόστηκε και για την υγρασία.

switch_isr

Input: *int status*

Output: *void*

Πρόκειται για την isr ρουτίνα, που εκτελείται όποτε πατιέται ο διακόπτης switch. Αρχικά, να αναφέρουμε οτι στην αρχή του προγράμματος έχει οριστεί η static int counter = 0, η οποία πρακτικά θα διατηρεί τον αριθμό των φορών που έχει πατηθεί το κουμπί. Έτσι, ανάλογα την τιμή του counter, διαφοροποιείται η τιμή της period. Την πρώτη φορά period = AEM_last_digits, και κάθε επόμενη period = 4 - (counter%2). Σε κάθε περίπτωση, τελικά εκτυπώνεται μέσω σειριακής διεπαφής με την UART η τιμή του counter, καθώς και η τιμή της μεταβλητής period, δλδ. ο ρυθμός δειγματοληψίας.

Συνάρτηση main

Εφόσον υλοποιήθηκαν οι interrupt handlers, ο timer, και οι drivers του DHT11 η βασική δουλειά της main είναι να αρχικοποιήσει το σύστημα και μετά να μπει σε μία άδεια while(1) λούπα, όπου θα περιμένει να έρθουν τα interrupts από τα περιφερειακά.

Για την αρχικοποίηση:

- Ενεργοποιούμε τα interrupts με ***__enable_irq()***.
- Αρχικοποιούμε τη UART με ***uart_init(115200)*** και ***uart_enable()***.
- Καλούμε την ***leds_init*** του leds.h ώστε να αρχικοποιηθούν τα leds.
- Για το κουμπί, αρχικά θέτουμε για το αντίστοιχο pin (***P_SW***) το mode σε ***PullDown*** και το trigger σε ***Rising*** και ορίζουμε τον interrupt handler του pin να είναι ο switch_ISR, όλα με τις κατάλληλες συναρτήσεις από το gpio.h, παρόμοια με το lab2.
- Αρχικοποιούμε την τιμή της period σε 2.
- Καλούμε την συνάρτηση uart_get_AEM ούτως ώστε να λάβουμε το άθροισμα των 2 τελευταίων ψηφίων του ΑΕΜ του χρήστη.
- Για τον timer: Αρχικοποιείται, ούτως ώστε να σκάει κάθε δευτερόλεπτο ***timer_init(1e6)***, και ορίζουμε τον interrupt handler του timer να είναι ο ***timer isr***. Πριν την ατέρμονη while, κάνουμε enable τον timer, ***timer_enable()***.
- Ακολουθεί τελικά η κενή while, με την __WFI(), στην οποία απλά περιμένουμε να έρθουν τα interrupts από τα περιφερειακά, ούτως ώστε να εκτελεστούν οι αντίστοιχες isr ρουτίνες.