TM - Projekt nr 4

Adam Kowalewski i Maciej Kłos.

Zadanie:
Animowanie zawartości wyświetlacza 8-pozycyjnego - dwa przyciski do „strzelania” - „pociski” są przesuwane z lewa na prawo i z prawa na lewo, a po kolizji znikają

Opis:

Aplikację-grę podzieliliśmy na trzy osobne moduły. Jeden z nich odpowiedzialny jest za obsługę przycisków, drugi za kontrolę wyświetlaniem cyfr na wyświetlaczu, a trzeci zawiera w sobie logikę gry i interrakcje z nią związane. Każdy z modułów należy przed użyciem zainicjować, jednak zbiorcza konfiguracja wszystkich timerów użytych w programie umieszczona została w pliku głównym programu.

System korzysta z dwóch źródeł zegarowych – kwarcu wysokiej częstotliwości ~8MHz oraz kwarcu zegarkowego 32768Hz. Ten pierwszy używany jest tylko jako źródło taktowania MCLK (zegar CPU), natomiast ten drugi jako zegar napędzający oba timery. Dzięki takiej konfiguracji, możemy w stanie oczekiwania uśpić mikrokontroler nawet w tryb LPM3 (aktywny tylko ACLK).

Pora na krótki opis poszczególnych modułów:

Jest tylko jeden moduł korzystający z TimeraB:

- Moduł obsługujący przyciski **buttons** – Odpowiedzialny jest za przechwytywanie przerwań zgłaszanych przez przyciski a następnie za ich odszumianie(debouncowanie). Odszumianie zrealizowane jest jako n - krotne próbkowanie stanu przycisku co pewien interwał czasowy (tutaj – 1024Hz). Po spróbkowaniu przycisku sprawdzany jest jego faktyczny stan i jego stan poprzedni. W chwili wykrycia prawdziwego naciśnięcia przycisku ustawiana jest odpowiednia flaga o zdarzeniu i mikrokontroler jest wyprowadzany ze snu. Za odszumianie odowiedziany jest TimerB, który ustawiony jest w tryb UP i TACCR0 wynoszący 31 (a tak właściwie 32 – 1)

Poniższe moduły korzystają z TimeraA:

- Moduł obsługujący wyświetlacz **display** – Odpowiedzialny jest za odświeżanie wyświetlacza 7-segmentowego. Realizowane jest to z częstotliwością $\frac{32768\,Hz}{81} \approx 404{,}5\,Hz$ w przerwaniach TimeraA jednostki TACCR0 ustawionej na wartość (80, a właściwie 81 – 1). W przerwaniu odświeżania ustawiana jest na wyświetlaczu następna pozycja i odpowiadające temu odpowiednie zapalenie segmentów. Wartość, która ma zostać wyświetlona, umieszczona jest w tablicy *displayArray*, a dostęp do niej realizowany jest przez funkcję *displaySetDigit(digitNumber, digitSegments)*. Należy uwzględniać fakt, iż zarówno segmenty jak i całe bloki wyświetlaczy aktywowane są wartościami ‘0’ na porcie wyjściowym.

- Moduł obsługujący grę **game** – Zawarta w nim jest cała logika gry jak i mechanizm odpowiedzialny za generowanie interwałów czasowych potrzebnych do przesuwania pocisków. Opóźnienia te tworzone są z częstotliwością $8\,Hz$, a odpowiadająca mu jednostka TACCR1 wynosi 4095 (4096 – 1). Interwał ten jest drugim źródłem (po przyciskach), który może wybudzać mikrokontroler z głębokiego snu, co sygnalizuje u siebie flagą *isTimerCycled* . Po wykryciu takiej flagi, należy wykonać procedurę *gameGoNextCycle*, w której następuje przesuwanie pocisków i wykrywanie wybuchów. Samo „wystrzeliwanie pocisków" realizowane jest przez procedury *gameBulletLeftAdd(), gameBulletRightAdd().* Dodatkowym atutem jest uwzględnienie w konfiguracji Buzzera, który po każdym wybuchu wydaje krótki dźwięk (długość jego określana jest przez jednostkę TACCR2 równą 255, co daje interwał $7.8125\,sek$). Stan wyświetlacza nie jest automatycznie aktualizowany – należy zażądać tego jawnie wywołując funkcję *gameUpdate*(). Ma ona dwa zadania – jeśli coś faktycznie jest do wyświetlenia, to aktualizuje tablicę wyświetlacza (i ew. wykrywa wybuch włączając przy tym buzzer), jeśli natomiast nie ma nic do wyświetlania, wyłącza TimerA, co pomaga zaoszczędzić energię. Sama logika gry nie jest skomplikowana. Każdy gracz posiada zmienną obrazującą stan jego przycisków *leftGamer, rightGamer*, które są cyklicznie przesuwane. Jednocześnie, w ślad za tym, każdy gracz posiada indeks *leftLastBulletPos, rightLastBulletPos*, mówiące o aktualnej pozycji ostatniego pocisku każdego z graczy. Wybuch generowany jest w trzech przypadkach:
  - Gdy pociski obu graczy spotkają się na tej samej pozycji (sytuacja ta występuje wtedy, gdy pociski są od siebie oddalone o nieparzystą liczbę pól)
  - Gdy pociski obu graczy 'wyprzedzą się' nawzajem (sytuacja ta ma miejsce, gdy pociski są oddalone od siebie o parzystą liczbę pól)
  - Gdy pocisk uderzy w ścianę przeciwnika (czyli dojdzie do końca).

  Miejsce wybuchu(albo wybuchów, tak jak w drugim przypadku) sygnalizowane jest przez zmienną *boomVar*, która jest resetowana podczas każdej kolejnej iteracji gry.

Moduł główny programu **main** jest stosunkowo prosty. W pętli głównej sprawdzane jest, czy nie nastąpiło naciśnięcie któregoś z przycisków, albo czy czasem nie nastąpiło kolejne cyknięcie zegara gry. Gdy którekolwiek z tych wydarzeń ma miejsce, wykonywane jest odpowiadające mu zadanie, a na koniec dokonuje się aktualizacji gry *gameUpdate().* W przeciwnym wypadku, mikrokontroler zasypia głębokim, bo LPM3, snem.