**Opis rozwiązania**

**STM32CubeIDE**

Moim zadaniem było stworzyć program, który będzie migał diodą LED podłączoną do mikrokontrolera o częstotliwości 5Hz. Dioda miała być podłączona do pinu PA5.

**Krok 1**

Na samym początku utworzyłem nowy projekt w edytorze STM32CUBEIDE o nazwie „zadanie1”, wybierając model mikrokontrolera STM32F205RGT6.

**Krok 2**

Od razu po stworzeniu projektu ustawiłem RCC („Reset and Clock Control”) na HSE („High Speed External”). HSE jest trochę bardziej skomplikowany w podłączeniu niż HSI, ale oferuje o wiele większą stabilność i dokładność. Także ustawiłem HSE na Crystal/Ceramic Resonator.

**Krok 3**

Teraz musiałem wybrać pin, do którego będzie podłączona dioda. Ustawiłem pin PA5 w tryb GPIO\_Output (żeby wysyłał sygnał, a nie odbierał). Również nadałem pinowi łatwą do zapamiętywania nazwę „DIODA”, dzięki czemu w kodzie mogłem wymienić DIODA\_Pin, żebym nie musiał zapamiętywać pinu.

**Krok 4**

Włączyłem i precyzyjnie ustawiłem timer - TIM2. Wybrałem Obliczyłem dokładnie Prescaler i Counter Period, żeby generował przerwania co 0.2 sekundy (5 Hz). Aby osiąnąć taki wynik, skorzystałem ze wzoru Czas = Prescaler/Częstotliwość. Częstotliwość mam ustawioną na 16 \* 10^6 Hz, więc po podstawieniu we wzór tej wartości i Czasu 0.0001 sekund, uzyskałem 1600 (wartość Prescalaru). Po czym, aby mieć w końcu 0.2 sekundy, ustawiłem Counter Period = 2000, dlatego, że to jest wartość do której timer liczy przed wygenerowaniem przerwania. Także włączyłem TIM2 Global interrupt. To oznacza, że gdy timer osiągnie swoją wartość docelową, mikrokontroler przerwie wykonywanie programu, zmieni stan diody, a potem wróci do normalnego działania.

**Krok 5**

W końcu przyszedł czas na watchdoga. Aby był on jak najskuteczniejszy, włączyłem IWDG (Independent Watchdog), który ma własny zegar i działa całkowicie autonomicznie.

**Krok 6** Po ustawieniach w pliku .ioc , zapisałem plik i wygenerowałem kod. Po czym dodałem swój kod między komentarzami /\*USER CODE BEGIN n\*/ a /\*USER CODE END n\*/. Po pierwsze, wywołałem HAL\_TIM\_Base\_Start\_IT(), funkcję, która odpowiada za start timera i generowanie przerwań „IT” - „interrupt”.

Wreszcie, w USER CODE BEGIN 4 zaimplementowałem funkcję to przełączania pinu DIODA\_Pin, jeśli timer TIM2 wywołał przerwanie.

**Algorytmy geometryczne**

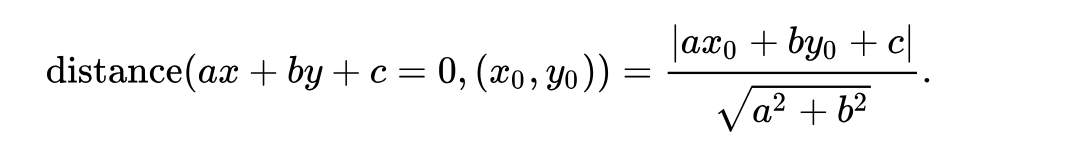
**Podpunkt 1**

Zacznę od tego, że próbowałem bardzo dużo metod i algorytmów, które wymyśliłem sam, ale kiedy znalazłem najlepsze rozwiązanie, okazało się, że wszystko już dawano było wymyślone.

Najpierw wpadłem na pomysł połączyć skrajne punkty po każdej stronie prostymi liniami i sprawdzać, czy ktoryś punkt nie okazał się poza otyczką (jeśli tak, dodać do otyczki). Ale, wbrew moim oczekiwaniom algorytm był nie dość, że mało efektywny, to bardzo trudny do zaimplementowania.

Drugim moim pomysłem był algorytm, który zaczyna budować otyczkę od jednego punktu i łączy wszystkie po kolei (według wskazówek zegara). Ta idea już wydawała się bardziej realistyczną, natomiast pojawiły się pewne trudności z kodem. Algorytm, według mojego pomysłu, miał działać na zasadzie kątów. Zaczynając od punktu pierwszego (najbardziej skrajny punkt po lewej i od dołu), rysować prostą do każdego innego punktu i porównywać kąty w które byłyby tworzone przez poprzedni punkt, startowy i następny. Odpowiednio zawsze następnym punktem w otoczce byłby punkt z którym kąt wychdził największy, dlatego, że znajdował się najbardziej po lewej na płaszczyźnie. Starałem się liczyć kąt według wzoru z twierdzenia cosinusów, i nawet udało mi się ten wzór zaimplementować, ale pojawiało się bardzo dużo problemów obliczeniowych i, niestety, nie dałem rady sam te problemy rozwiązać. Natomiast, kiedy szukałem odpowiedzi na pytania dotyczące obliczeń w moim algorytmie, znalazłem algorytm, działający bardzo, i to bardzo podobnie do mojego. Nazywa się on Alorytm Grahama. Jedynie różnił się tym, że skręt do kolejnego punktu w otoczce znajdował nie za pomocą kątu bezpośrednio, tylko przez iloczyn skalarny wektorów (jeśli > 0, to skręt w lewo, < 0 - skręt w prawo, == 0 - punkty współliniowe). Zatem, wiedząc dokąd skręca wektor, mogłem wyjaśnić, który będzie następny punkt.

**Podpunkt 2**

Drugie zadania poszło mi szybciej, przez to, że użyłem w rozwiązaniu wynik pierwszego zadania (otyczkę). Po prostu musiałem znaleźć minimalną odległość między równoległymi prostymi, które zawierają punkty na otyczce (żeby były jak najmniej oddalone od siebie) wspierającymi otoczkę. Oczywiście, że też pujawiły się problemy związane z implementacją pomysłu, najdłużej nie mogłem poradzić z obliczeniem odległości od punktu, do linii, ale pomógł mi w tym wzór .

**Podpunkt 3**

Na pierwszy rzut oka ten problem wydawał się najmniej trudnym ze wszystkich, biorąc po uwagę, że już mam wyniki z poprzednich podpunktów, z których mogę korzystać, ale główny problem polegał na tym, żeby nie używać najprostszego algorytmu, przez O(n^2). Chociaż w końcu użyłem go do obliczenia odległości między trzema punktami, bo dla tej iłości punktów po prostu nie dało się stosować skuteczniejszego algorytmu.

Pierwszy niezły pomysł, na który wpadłem było sortowanie wszystkich punktów względem jednej ze współrzędnych (w moim przypadku sortowałem do X) i sprawdzać tylko sąsiadów każdego punktu. Było to już lepsze od O(n^2), ale wciąż czułem, że mogę coś naprawić, ulepszyć. Wtedy zacząłem czytac o podobnych algorytmach i znalazłem ten, który działa podonie do mojego, ale jest już O(nlog(n)). Algorytm również polegał na sortowaniu punktów wegług współrzędnej, ale potem jeszcze dzielił zbiór na dwie połowy „pionową” linią. Pozostawało tylko rekurencyjnie znaleść najbliższe pary w obu połówkach i sprawdzić pas wokół podziału, a potem zwrócić parę punktów z minimalną odległością od siebie.

**Testing**

Rzecz jasna, algorytmy musiałem wielokrotnie testować używając przerównych wartości. Poprawność testów spradzałem na kartce i za pomocą kalkulatora.