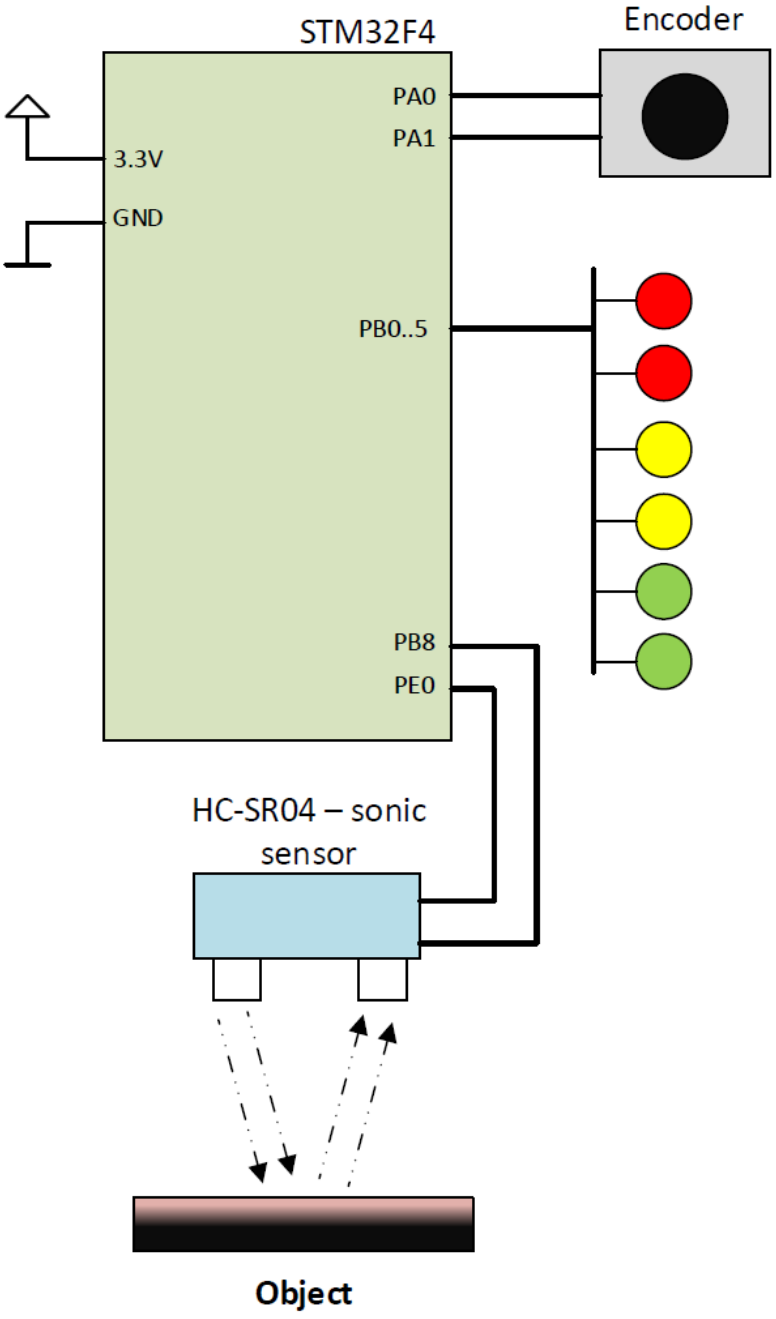
Systemy czasu rzeczywistego – projekt

Autor: Kamil Malski

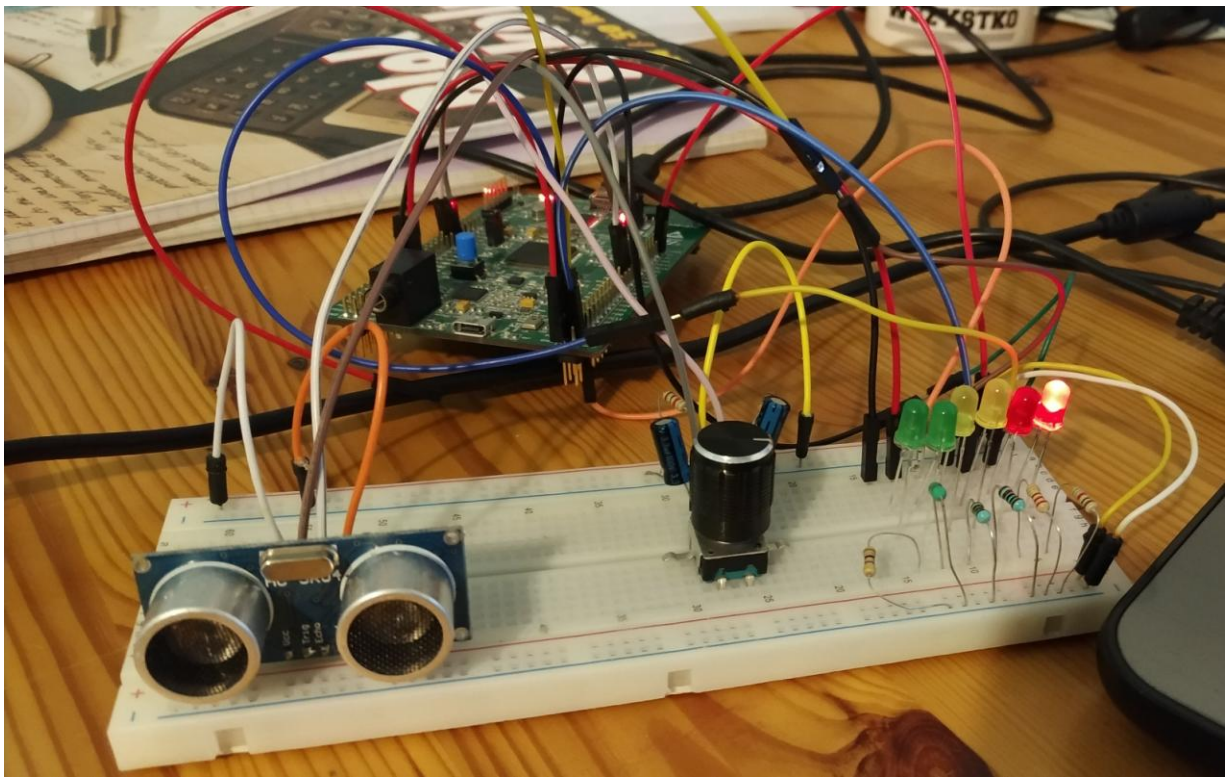
Data: styczeń 2020

1. Wstęp

Projekt polegał na zaprojektowaniu oprogramowania działającego w ramach systemu operacyjnego FreeRTOS wykorzystującego wątki, semafory, kolejki i przerwania sprzętowe.
W ramach projektu stworzony został program realizujący przykład radaru wykrywającego przeszkodę za pomocą czujnika ultradźwiękowego i w zależności od odległości od obiektu zapalając kolejno diody od czerwonej dla obiektu najbliżej aż do zielonej dla obiektu maksymalnie oddalonego. Dodatkowo dodany został enkoder mechaniczny mogący zmienić progi poszczególnych stopni, tak aby dopasować je do potrzeb użytkownika.



Rys 1. Schemat ideowy układu



Rys 2. Rzeczywisty obwód

2. Użyty sprzęt

Platformą sprzętową użytą w projekcie został zestaw Discovery STM32F407 wraz z czujnikiem odległości HC-SR04, enkoderem mechanicznym oraz zestawem sześciu diod.

Czujnik odległości działa na zasadzie zapytanie-odpowiedź, czyli po otrzymaniu stanu wysokiego na pin TRIGGER wysyła wiązkę ultradźwiękową i na jej podstawie oblicza odległość od przeszkody. Na pinie ECHO pojawia się stan wysoki na czas proporcjonalny do wyznaczonej odległości. Procesor za pomocą timera wyznacza czas stanu wysokiego i oblicza odległość

Enkoder po przekręceniu o jeden krok zwiera jedno ze swoich wyjść do masy. Wyjście to podłączone jest do wejścia mikroprocesora i ustawione w odpowiedni tryb Timera który zlicza impulsy. Po zliczeniu ustalonej ilości wystawia przerwanie sprzętowe. Działa to tak samo dla obrotu w prawo i w lewo, detekcja kierunku możliwa jest za pomocą sprawdzenia czy licznik przepełnił się od góry czy od dołu.

Diody sterowane są bezpośrednio z wyjść mikroprocesora.

Podsumowując, **w projekcie użyto 3 timery, jedno przerwanie sprzętowe oraz 7 pinów wyjściowych.**

3. Szczegóły implementacyjne

Projekt został wykonany w środowisku Atollic True Studio 9.0.0 i oparty o system czasu rzeczywistego FreeRTOS v10.2.1_191129.
Całość oprogramowania znajduje się w publicznym repozytorium i jest w całości własnością autora projektu:
https://github.com/komakow/SuperDuperSuperVisor

W projekcie znajdują się 3 wątki:

- **TaskEncoder** – odpowiada za obsługę enkodera:
  - Stack Depth: 100 bajtów
  - Priority: 2
- **TaskSonicSensor** – odpowiada za obsługę czujnika odległości :
  - Stack Depth: 100 bajtów
  - Priority: 3
- **TaskServoMotor** (docelowo zamiast diod miało być servo, jednak z powodów technicznych pojawiły się diody, a nazwa wątku pozostała) odpowiada za sterowanie diodam.
  - Stack Depth: 100 bajtów
  - Priority: 2

oraz 4 kolejki:

- **EncoderQue** – kolejka łącząca przerwanie sprzętowe od en kodera z wątkiem TaskEncoder. Informuje wątek o tym, że enkoder został przekręcony o ustalony na etapie implementacji kąt oraz o kierunku obrotu. Rozmiar pojedynczej wiadomości: 1 bajt
- **DiodePlusQue** – kolejka łącząca wątek TaskEncoder z wątkiem TaskServoMotor i informująca o potrzebie zwiększenia progu zaświecenia diod o dodatkowe 10cm. Rozmiar pojedynczej wiadomości: 1 bajt
- **DiodeMinusQue** – kolejka łącząca wątek TaskEncoder z wątkiem TaskServoMotor i informująca o potrzebie zmniejszenia progu zaświecenia diod o dodatkowe 10cm. Rozmiar pojedynczej wiadomości: 1 bajt
- **SonicSensorQue** – kolejka łącząca wątek TaskSonicSensor z wątkiem TaskServoMotor i przesyłająca zmierzoną odległość od obiektu. Rozmiar pojedynczej wiadomości 2 bajty.

```
//create tasks
xTaskCreate( vSonicSensor, "sonic", 100, NULL, 3, &xSonicSensor );
xTaskCreate( vServoMotor, "servo", 100, NULL, 2, &xServoMotor );
xTaskCreate( vEncoder, "encoder", 100, NULL, 2, &xEncoder );

//create queue
xEncoderQue = xQueueCreate(1, sizeof(uint8_t));
xDiodePlusQue = xQueueCreate(1, sizeof(uint8_t));
xDiodeMinusQue = xQueueCreate(1, sizeof(uint8_t));
xSonicSensorQue = xQueueCreate(1, sizeof(uint16_t));
```

Rys 3. Screen z kodu tworzącego wątki i kolejki

W większości zastosowane zostały domyśle wartości konfiguracyjne FreeRTOSa. Jedyną wartością zmienioną była wartość ‘configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY’ z 5 na 16, tak aby system mógł poprawnie reagować na przerwanie sprzętowe udostępnione przez rdzeń dla użytkownika.

```
#define configUSE_PREEMPTION      1
#define configUSE_IDLE_HOOK       0
#define configUSE_TICK_HOOK       1
#define configCPU_CLOCK_HZ        ( (uint32_t)168000000 )
#define configTICK_RATE_HZ        ( ( TickType_t ) 1000 )
#define configMAX_PRIORITIES      ( 5 )
#define configMINIMAL_STACK_SIZE    ( ( unsigned short ) 130 )
#define configTOTAL_HEAP_SIZE     ( ( size_t ) ( 75 * 1024 ) )
#define configMAX_TASK_NAME_LEN     ( 10 )
#define configUSE_TRACE_FACILITY    1
#define configUSE_16_BIT_TICKS      0
#define configIDLE_SHOULD_YIELD     1
#define configUSE_MUTEXES       1
#define configQUEUE_REGISTRY_SIZE   8
#define configCHECK_FOR_STACK_OVERFLOW  0
#define configUSE_RECURSIVE_MUTEXES   1
#define configUSE_MALLOC_FAILED_HOOK  0
#define configUSE_APPLICATION_TASK_TAG  0
#define configUSE_COUNTING_SEMAPHORES 1
#define configGENERATE_RUN_TIME_STATS 0

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES     0
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )

/* Software timer definitions. */
#define configUSE_TIMERS        1
#define configTIMER_TASK_PRIORITY   ( 2 )
#define configTIMER_QUEUE_LENGTH    10
#define configTIMER_TASK_STACK_DEPTH  ( configMINIMAL_STACK_SIZE * 2 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */
#define INCLUDE_vTaskPrioritySet    1
#define INCLUDE_uxTaskPriorityGet   1
#define INCLUDE_vTaskDelete       1
#define INCLUDE_vTaskCleanUpResources 1
#define INCLUDE_vTaskSuspend      1
#define INCLUDE_vTaskDelayUntil     1
```

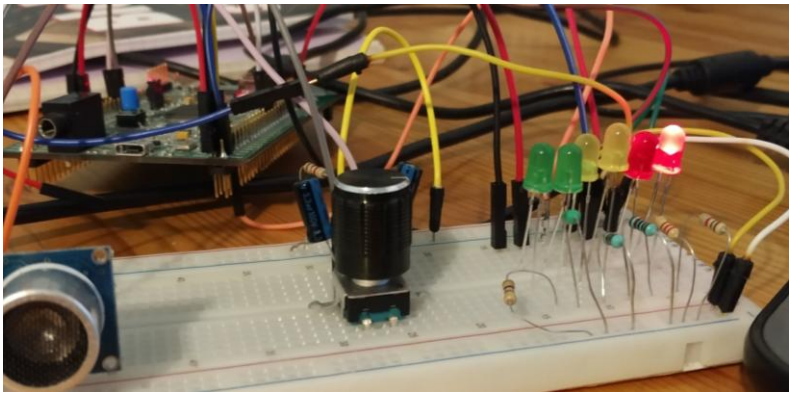
Rys 4. Fragment pliku konfiguracyjnego FreeRTOSa

Za pomocą makr systemowych ‘vTaskDelay()’ każdy wątek był usypiany na określoną ilość czasu zależną od zadania które wykonywał. Czas ten mieścił się w zakresie 20-50ms.

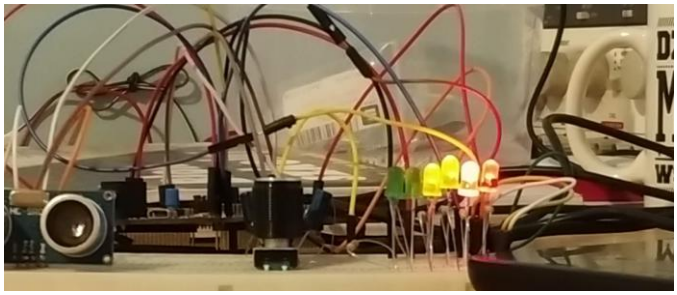
4. Wynik prac

Projekt zakończył się sukcesem. Za jego pośrednictwem autor poznał pracę systemu czasu rzeczywistego w realnym środowisku przy realnym projekcie.
Poniżej zdjęcia przedstawiające działanie. Czym obiekt jest dalej, tym więcej diod się świeci.

1. Jedna dioda czerwona włączona – obiekt znajduje się blisko



2. Dwie czerwone i dwie żółte diody włączone – obiekt znajduje się w okolicach połowy mierzonego zakresu.



3. Wszystkie diody włączone – obiekt poza zakresem mierzonego obszaru.