OPTIMUS
Przedsiębiorstwo Optimus Sp. z o.o.
ul. Jana Pawła II 84K
98-200 Sieradz
+48 43 827 10 07
+48 501 538 149
optsier@onet.pl
NIP 8270018155
REGON 730173184
KRS 0000203770

AIS RW.E3
Aktualizacja firmware w locie (on-the-fly)

Dotyczy produktu: AIS.AS Prototyp autonomicznego i inteligentnego sterownika (Optimus)

2019.12.16

Spis treści

Fundusze Europejskie Program Regionalny
promuje łódzkie
Unia Europejska Europejski Fundusz Rozwoju Regionalnego

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

1. Wprowadzenie

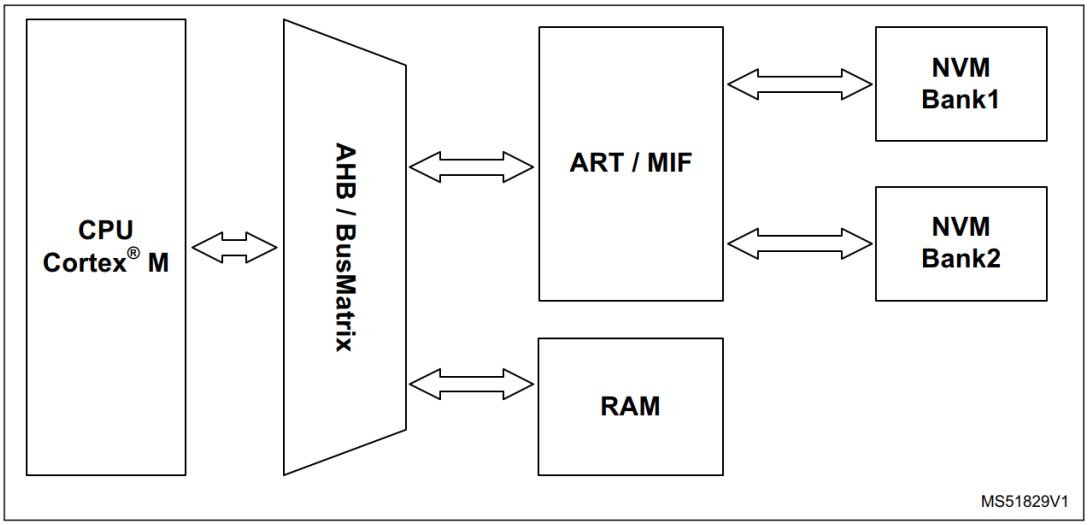
Funkcja podwójnego banku jest cechą wspólną wielu mikrokontrolerów. Celem tego dokumentu jest opisanie, jak korzystać z tej funkcji oraz aktualizacji firmware na żywo (on-the-fly).

Główną zaletą aktualizacji na żywo jest minimalizacja przestojów podczas fazy przejściowej, szczególnie pożądanej w zagadnieniach systemów czasu rzeczywistego.

Dokument dotyczy w szczególności urządzeń opartych o mikrokontrolery serii STM32L0, STM32L4i STM32G4.

2. Podsumowanie podsystemu pamięci

Mikrokontrolery STM32 oparte są na rdzeniach Arm. Podręczniki referencyjne szczegółowo opisują podsystem pamięci. Skrajnie uproszczony schemat przedstawia poniższy diagram.



Rysunek 1 Uproszczony schemat podsystemu pamięci

Seria STM32L0 wykorzystuje Cortex M0+, natomiast seria STM32L4 i STM32G4 korzysta z rdzenia Cortex M4 z osobnymi magistralami danych i instrukcji. Fakt ten przekłada się na większą złożoność magistrali AHB BusMatrix w serii STM32L4 i STM32G4. Kolejna różnica polega na tym, że STM32L0 są oparte na technologii EEPROM.

Wszystkie MCU omówione w tym dokumencie posiadają pamięć podręczną w interfejsie NVM, prostszą w przypadku serii STM32L0, bardziej zaawansowanym ART Accelerator na urządzeniach opartych na Cortex M4.

Pamięć wspierająca obsługę „podwójnego banku" może być skonfigurowana i używana jako pojedynczy duży blok NVM z ciągłym adresowaniem (z kilkoma wyjątkami, nie omówionymi w tym dokumencie). Najbardziej znaczącą zaletą trybu dualbank jest możliwość pisania na jednym banku bez przerywania czytania (i pobierania instrukcji) z drugiego banku. Jest to najważniejszy warunek wstępny do przeprowadzenia aktualizacji bez przerywania wykonania kodu z programu NVM.

Podczas projektowania aplikacji korzystającej z urządzenia z dwoma bankami istnieje kilka możliwości wyboru sposobu wykorzystania drugiej połowy pamięci programu.

3. Aktualizacja w locie

Jest to również proces, który pozwala użytkownikowi modyfikować kod i konfigurację bez zakłócania normalnej pracy urządzenia. Jest jeszcze więcej zalet w porównaniu do prostego rozwiązania IAP (*ang. In application programming*):

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika"*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

- kod modułu ładującego może być aktualizowany podczas korzystania z podwójnego banku,
- jeśli ładowanie się nie powiedzie, oryginalny kod nadal działa (operacja może być „atomowa”),
- nie ma potrzeby definiowania stanu modułu ładującego, urządzenie zawsze może załadować kod.

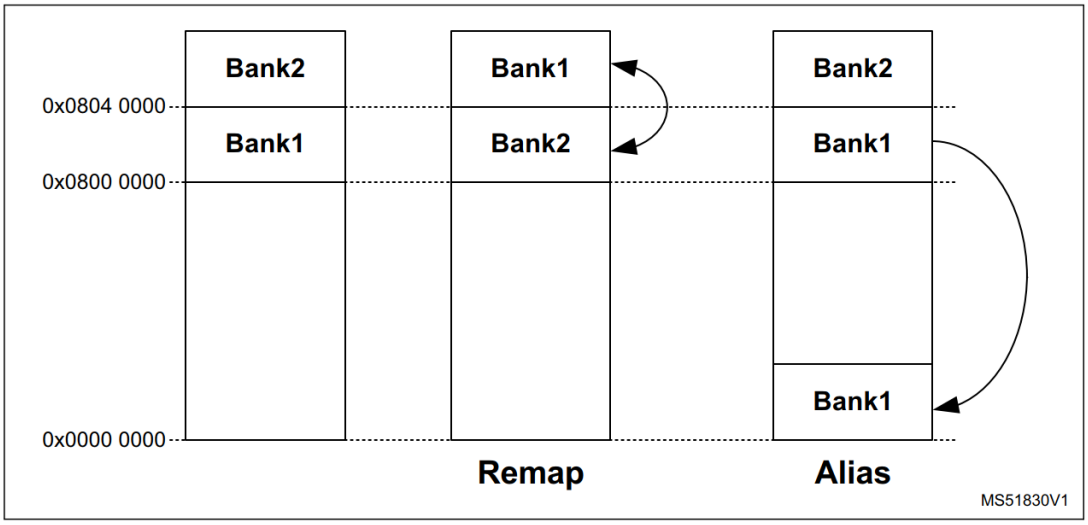
W przykładach opisanych w tym dokumencie celem jest przejście od pełnej operacji opartej na oryginalnym kodzie do pełnej operacji opartej na zaktualizowanym kodzie w ciągu mikrosekund. Dokument ten opisuje również przypadek, gdy ISR (*ang. Interrupt service request*) jest wykonywany z pamięci RAM przy użyciu przeniesionej tabeli wektorów. Jest to powszechne rozwiązanie pozwalające uzyskać niskie opóźnienie przerwania, ale jest trudne do wykonania z aktualizacją „w terenie”.

3.1. Funkcje wspierające MCU

W mikrokontrolerze znajduje się kilka mechanizmów zapewniających działanie oprogramowania wewnętrznego, a wśród nich najważniejsza jest możliwość napisania jednej nieulotnej części podczas wykonywania kodu z drugiej.

3.1.1. Przełącznik odwzorowywania pamięci

Bit kontrolny oznaczony jako FB_MODE dla STM32L4 i STM32G4 oraz UFB dla STM32L0 jest dostępny dla użytkownika. Ten bit w rejestrze konfiguracji systemu steruje mapowaniem pamięci. Jest również używany przez mechanizm rozruchowy z dwoma bankami i przy wystarczającej staranności może być również używany do aktualizacji na żywo.



Rysunek 2 Mapowanie NVI na MCU STM32G4

W zależności od ustawienia flagi albo Bank1, albo Bank2 jest mapowany na początek pod adresem 0x0800 0000 z aliasem na adres 0x0000 0000. Ponieważ operacja nie wpływa na komputer i inne rejestry procesora, CPU po prostu pobierze następną instrukcję z innego banku kiedy bit jest odwrócony.

Kody w obu bankach są zwykle połączone tak, aby zaczynały się pod adresem 0x0800 0000.

3.1.2. Relokowalna tablica wektorów przerwań

CPU posiada konfigurowalne przesunięcie dla IVT. Pozwala oprogramowaniu zdefiniować więcej IVT i przełączać się między nimi w razie potrzeby.

Domyślna wartość przesunięcia tablicy wektorowej to 0x0000 0000, co zwykle wskazuje na alias pamięci programu. Ważne jest, aby zarezerwować wystarczającą ilość miejsca na tablicę wektorów, 4 bajty na każdy wektor przerwań. Istnieją również ograniczenia dotyczące wartości adresu przesunięcia, które muszą być wyrównane do wielokrotności 512 bajtów.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

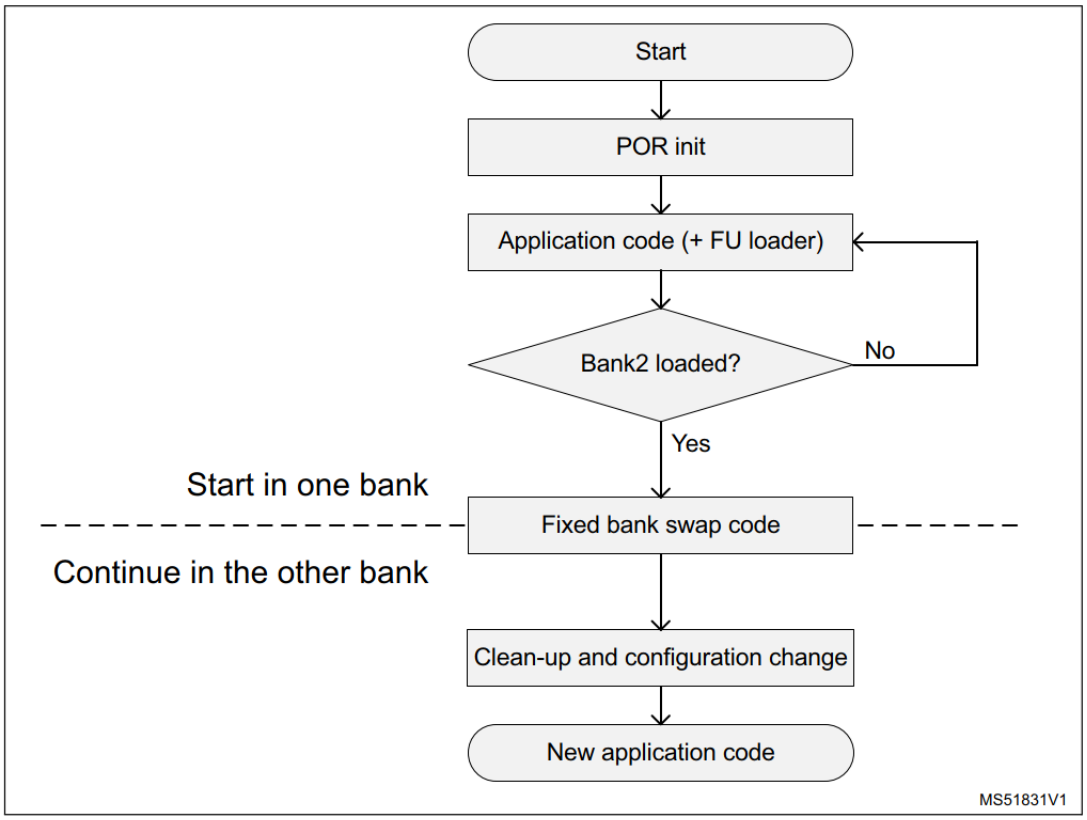
3.1.3. Flaga BFB2 w bajtach opcji użytkownika

Flaga zasadniczo uruchamia próbę uruchomienia z banku 2 po zresetowaniu. Ważne jest, aby utrzymać flagę BFB2 ustawioną, gdy nie ma kodu w banku 1, na wypadek nieoczekiwanych awarii zasilania. Po ustawieniu BFB2 uruchamiany jest moduł ładujący system, aby ocenić, czy kod jest obecny w banku 2, a następnie, jeśli to możliwe, kontroluje go (szczegóły podano w instrukcjach referencyjnych lub w AN2606). Następnie oprogramowanie układowe musi wykryć, że kod w banku 1 musi zostać wymieniony, a proces działa z banku 2.

W tej aplikacji BFB2 jest używany tylko jako mechanizm awaryjny w przypadku nieoczekiwanego odcięcia zasilania, gdy kod jest wykonywany z Bank2. BFB2 musi pozostać aktywny, gdy Bank1 nie zawiera poprawnego kodu, jest kasowany, gdy nowy kod jest kopiowany z Banku2 do Banku 1.

4. Kod niemodyfikowalny

Jak wspomniano wcześniej, podczas wymiany banku, CPU pobierze następną instrukcję z drugiego banku, ze względu na mapowanie banku. Zalecanym podejściem jest utworzenie sekcji kodu, najlepiej w ramach funkcji `main()`, w której stos jest na najniższym poziomie i uczynienie tej sekcji niemodyfikowalnym kodem. Ten niemodyfikowalny i stały kod zajmie się samym przejściem. Jeśli kod w obu bankach nie jest spójny w pobliżu punktu przejścia, wykonanie może być nieprzewidywalne, a nawet ulec awarii.



Rysunek 3 Kod wymiany banku

Ten kod może w rzeczywistości stanowić bardzo małą część całego projektu (im mniejszy, tym lepszy). Ponieważ nie jest zwykle modyfikowany przez aktualizację na żywo, musi być wolny od błędów.

Ten ustalony kod wymiany banku wykonuje następujące działania:

- Wyłącza przerwania
- Wyłącza i opróżnia pamięć podręczną
- Zmienia przesunięcie tablicy wektorów przerwań
- Odwraca bank

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

- Przywraca konfigurację pamięci podręcznej
- Włączyć przerwania

Należy pamiętać, że posiadanie identycznego kodu źródłowego nie oznacza, że zostaną wygenerowane identyczne kody binarne. Lepiej jest wygenerować bibliotekę do połączenia ze wszystkimi przyszłymi wersjami, zachować plik obiektowy lub (być może) nawet ostrożnie edytować wynikowy plik binarny.

5. Kod w pamięci RAM

Jeśli rozwiązanie z niemodyfikowalnym kodem zostanie uznane za zbyt ryzykowne dla zamierzonej aplikacji, możliwe jest, że funkcja będzie opcjonalnie manipulować wskaźnikiem stosu w pamięci RAM, a następnie przeskoczy do bezpiecznego punktu wejścia w kodzie pamięci Flash dla wymiany banku.

W podanym przykładzie pamięć RAM jest używana tylko dla kodu ISR. Uzasadnieniem jest to, że aplikacje wymagające aktualizacji w locie zazwyczaj tak robią, ponieważ muszą reagować na sygnały w deterministycznym czasie, przy minimalnym opóźnieniu. Pamięć Flash jest wolniejsza niż pamięć RAM, dlatego krytyczne czasowo ISR są umieszczane w pamięci ulotnej, gdzie nowe instrukcje po rozgałęzieniu można pobrać bez opóźnień (przynajmniej do momentu, gdy funkcja ISR wywoła funkcję w pamięci Flash).

Jednak chociaż zazwyczaj istnieją różne obszary SRAM, nie można ich ponownie mapować, jak banki pamięci Flash. Zastąpienie kodu w pamięci nietrwałej zwykle obejmuje wyłączenie przerwań, skopiowanie kodu zastępczego, a następnie ponowne włączenie przerwań, z ryzykiem totalnej awarii w przypadku NMI. Alternatywnie można użyć tymczasowego kodu przerwania umieszczonego w pamięci Flash podczas aktualizowania zawartości pamięci RAM, ze świadomością związanego z tym opóźnienia.

W naszym przykładzie użyto trzeciej opcji. Rozwiązaniem jest zdefiniowanie w RAM dwóch sekcji, z których każda poświęcona jest kodowi ISR dla jednego banku. Plik binarny jest połączony z dwiema identycznymi kopiami każdej funkcji w pamięci RAM. Proces może współpracować z każdym z nich, w zależności od banku pamięci Flash, w którym jest wykonywany. Nieużywaną kopię można zaktualizować bez zakłócania normalnej funkcjonalności, podobnie jak podczas programowania drugiego banku w pamięci Flash. Ale zamiast ponownego mapowania przesunięcie tablicy wektorowej służy do przełączania między dwiema tabelami wektorowymi, z których każda zawiera adresy różnych kopii funkcji ISR. W ten sposób ryzyko NMI jest zmniejszone, ale procedura obsługi NMI musi pozostać niezmieniona i nie odwoływać się do innej pamięci.

Przesunięcie tablicy wektorów jest przechowywane w rejestrze Arm zdefiniowanym w CMSIS jako SCB → VTOR. Jest dostępny dla użytkownika.

Dla funkcji RAM, które nie są ISR, możliwe jest zdefiniowanie podobnej tabeli referencyjnej (o ile złożoność wątków nadal sprawia, że umieszczanie kodu w pamięci RAM jest korzystne). W tym przykładzie nie zastosowano takiego kodu.

6. Zmienne dane

Celem aktualizacji na żywo jest przejście do nowej wersji kodu bez konieczności resetowania systemu. Po zresetowaniu kod startowy (zwykle domyślnie pochodzący od dostawcy narzędzia programistycznego) inicjuje pamięć RAM przed przekazaniem kontroli kodowi użytkownika. Zazwyczaj polega to na czyszczeniu adresów obszaru pamięci RAM zainicjalizowanych zerowo i kopiowaniu wartości domyślnych z NVM do obszarów zwanych ciągłą inicjalizacją.

Większość dzieje się „za kulisami”, zauważany jest tylko dłuższy czas uruchamiania (w szczególności w przypadku nadmiernej ilości zainicjowanych zmiennych globalnych).

Korzystając z aktualizacji „w locie”, programista musi zarządzać rozmieszczaniem i zawartością lotnych struktur danych. Jako absolutne minimum należy sprawdzić plik mapowania i zobaczyć zmiany w zakresie adresów RAM.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

6.1. Unikanie zmian w niestabilnej strukturze danych

Jeśli nowy plik mapowania pokazuje znaczące niepożądane i nieprzewidziane zmiany w rozmieszczeniu danych w pamięci RAM, istnieje kilka sposobów umieszczenia ich w pierwotnej lokalizacji. Niestety nie ma uniwersalnego sposobu. Nic nie zadziała automatycznie, niezależnie od użytego środowiska kompilacji i budowania. Linker decyduje o umieszczeniu danych w pamięci, a wszystkie opcje linkera są specyficzne dla danego rozwiązania.

Niektórym modyfikacjom można zapobiec, unikając zmian w ustawieniach kompilatora i konsolidatora, ale bardziej niezawodnym rozwiązaniem jest całkowite usunięcie źródeł i zastąpienie ich plikami obiektowymi wygenerowanymi w oryginalnej wersji.

Nowo dodane dane można przenieść do osobnej, wcześniej nieużywanej sekcji pamięci. Jeśli zmienna zostanie usunięta, co powoduje przesunięcie adresu, wystarczy zastąpić ją atrapą – nieużywaną zmienną o tym samym rozmiarze.

W każdym razie zaleca się zapoznanie z dokumentacją linkera, aby zobaczyć, jaką pomocą może on służyć w celu ułatwienia procedury aktualizacji.

6.2. Zarządzanie modyfikacjami niestabilnej struktury danych

Możliwe jest zdefiniowanie pięciu podstawowych poziomów istotności modyfikacji, jak wyszczególniono w poniższej tabeli.

Tabela 1 Poziomy istotności modyfikacji

Severity	Case	Action after update
0	Map file of RAM is identical	No action
1	Variables removed	No action (but double-check addresses)
2	New variable added	Initialize as needed
3	Address modified	Move data
4	Structure modified	Code must convert old data to the new format

Zalecanym rozwiązaniem jest posiadanie dedykowanej funkcji w nowym kodzie – inicjalizatora, który wykonuje się tylko raz po załadowaniu aktualizacji. Ten jednorazowy kod zajmie się zawartością pamięci RAM, zanim rzeczywisty kod funkcjonalny będzie musiał uzyskać dostęp do zmiennych.

Ten dokument nie obejmuje przypadku dynamicznie alokowanej pamięci stosu. Nie ma gwarantowanej funkcjonalności aktualizacji stosu pamięci RAM w przypadku aktualizacji w locie i lepiej jej unikać. Specyficzne implementacje mogą być użyteczne, ale z ograniczeniami.

6.3. Koordynacja (timing)

Nie trzeba wykonywać wymiany banku natychmiast po zakończeniu procesu ładowania. Załadowane dane pozostaną dostępne w drugim banku, dopóki FW nie zdecyduje, że nadszedł właściwy czas.

W zależności od rzeczywistej aplikacji mogą występować różne zadania o różnej częstotliwości. W niektórych przypadkach możliwe jest określenie przedziału czasowego, w którym mniej prawdopodobne jest wystąpienie przerwania o znaczeniu krytycznym.

Przerwanie o najwyższym priorytecie jest zwykle najważniejsze, na przykład pętla sterowania. Jeśli można przewidzieć, że nie pojawi się ponownie przez wystarczające okno czasowe, jest to dobry moment na zainicjowanie wymiany banku.

Flaga w systemie działającym na MCU może być sygnałem do kontynuacji procesu aktualizacji. Może to być prosty bit lub znacznik czasu, w zależności od architektury systemu oprogramowania.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

Aby zminimalizować czas potrzebny na zamianę, należy ustawić zegar systemowy na wartość maksymalną.

6.4. Przykład oprogramowania układowego

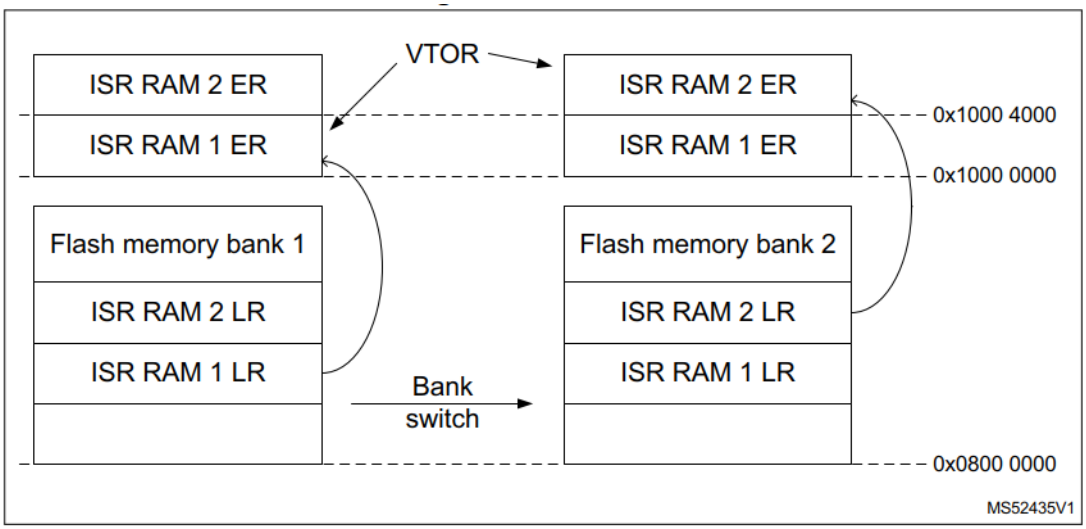
Pakiet FW o nazwie X-CUBE-DBFU zawiera proste przykładowe rozwiązanie przeznaczone do użycia z wybranymi płytami STM32 EVAL. Jest ściśle związany z aplikacją IAP, z istotną różnicą. Istnieje tylko jedno oprogramowanie wewnętrzne, identyczne dla obu banków, służące zarówno jako moduł ładujący, jak i aplikacja.

6.4.1. Przykładowa architektura rozwiązania

Jednym z wyzwań poruszonych w tym przykładzie jest wykonanie procedur obsługi przerwań z pamięci RAM. Każdy obsługiwany produkt jest unikalny pod względem pamięci RAM (pojedynczy blok pamięci RAM dla STM32L0, dwa odrębne obszary pamięci RAM dla STM32L4 i trzy dla STM32G4). Nie wpływa to na funkcjonalność przykładów, ponieważ pamięci RAM nie obsługują dwóch banków. Oznacza to, że dwa zakresy adresów RAM nie mogą zostać zamienione przez remaping sprzętowy.

Zamiast tego stosuje się rozwiązanie opisane w rozdziale „Kod w pamięci RAM”. Druga tabela jest zadeklarowana w pliku źródłowym `stm32xxxx_it2.c` dla każdego konkretnego projektu. Korzystając z tego przykładu jako punktu wyjścia do dalszego rozwoju, ważne jest aktualizowanie obu kopii kodu procedury przerwania. W przeciwieństwie do sekcji kodu niemodyfikowalnego, nie muszą one mieć identycznych plików binarnych, tylko funkcjonalność powinna być taka sama (chyba że istnieje intencja, aby produkt zachowywał się inaczej podczas uruchamiania z Bank2).

Ustawienie tabeli wektorów jest pierwszą rzeczą wykonaną w kodzie. W tym przykładzie odbywa się to w funkcji `main()`, ale możliwe jest zmodyfikowanie `SystemInit()` w pliku `system_stm32xxxx.c` alternatywnie. Oczywiście najpierw należy zainicjować rzeczywistą tabelę, ponieważ jest ona przechowywana w pamięci ulotnej, a następnie wartość VTOR jest odpowiednio modyfikowana.



Rysunek 4 ISR w pamięci RAM

6.4.2. Konfiguracja HW

Podłącz port USART2 do komputera za pomocą kabla szeregowego RS-232 (zalecane jest użycie standardowego adaptera USB działającego jako wirtualny port COM). Użyj aplikacji terminalowej obsługującej protokół YMODEM. Tera Term oferuje solidną, bezpłatną alternatywę dla Windows HyperTerminal.

Skonfiguruj prędkość komunikacji na: 115200, 8 bitów danych, bez parzystości, 1 bit stopu.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

6.4.3. Przykładowa procedura postępowania

Dzięki interfejsowi użytkownika zapewnianemu przez terminal SW postępowanie jest łatwe. Przykładowe oprogramowanie aktualizujące najpierw wyświetla nagłówek i menu. Naciśnięcie klawiszy z cyframi na klawiaturze komputera powoduje wybranie opcji.

1. Pobierz plik do drugiego banku
   Pierwszy wybór menu inicjuje pobranie pliku binarnego do innego banku. Poprzednia zawartość drugiego banku jest usuwana i nadpisywana. Rozmiar pliku jest ograniczony przez rozmiar pamięci. Wybrany plik do pobrania, zostanie on odszyfrowany i zapisany. Komunikacja przebiega przy użyciu protokołu YMODEM.
2. Usuń zawartość drugiego banku
   Ta opcja unieważnia zawartość innego banku.
3. Przepisz treść drugiego banku
   Kod z aktywnego banku jest kopiowany do przeciwnego banku pamięci.
4. Sprawdź integralność innego banku
   Ta opcja inicjuje sprawdzenie zawartości innego banku. W przypadku serii L0 przykładowy FW przechowuje wartość integralności kodu CRC w pamięci danych EEPROM. W przypadku innych urządzeń kontrola ogranicza się do prostej obecności.
5. Przełącz wybór banku
   Jeśli występuje prawdopodobieństwo, że w drugim banku znajduje się kod funkcjonalny, tabela wektorów jest przepisywana, a banki są przełączane.
6. Przełącz bank systemowy
   Programowanie bajtów opcji w celu zmodyfikowania wartości bitu BFB2 (bootowanie z Bank2).

Proponowaną sekwencję aktualizacji w locie można opisać sekwencją 1, 4, 5, 6 (BFB2 ON), 3, 4, 5, 6 (BFB2 OFF). Dołączono opcję 2 (kasowanie), aby zasymulować stan awarii.

6.4.4. Opcja szyfrowania

Jest jasne, że adres IP zawarty w pliku aktualizacji oprogramowania układowego może być cenny dla właściciela i mogą istnieć obawy dotyczące poufności kodu. W połączeniu z ochroną integralności dodatkowe szyfrowanie symetryczne zapewnia podstawową ochronę przed wprowadzeniem fałszywego lub fałszywego oprogramowania układowego.

Kilka mikrokontrolerów ST zawiera opcjonalną opcję sprzętowego koprocesora AES. W takim przypadku możliwe jest użycie STM32G484 zamiast STM32G474, STM32L486 zamiast STM32L476 i STM32L083 zamiast STM32L073, aby wykorzystać funkcjonalność szyfrowania.

6.4.5. Szyfrowanie pliku binarnego

Schemat szyfrowania zastosowany w tym przykładzie to zwykły AES używany w trybie licznika (CTR). Zaletą tej metody jest to, że nie jest konieczne całkowite wypełnienie, tylko rzeczywista zawartość jest szyfrowana. Chociaż nie jest to istotne w przypadku aktualizacji pola, tryb licznika jest również zastosowany z tego powodu, że nie propaguje błędów komunikacji (tylko bajty, które uległy uszkodzeniu podczas komunikacji, pozostają uszkodzone w odszyfrowanej wiadomości).

Co więcej, nie ma potrzeby opracowywania dedykowanego narzędzia do szyfrowania i można wykorzystać publicznie dostępne biblioteki.

Jeśli nie masz pewności, jak przygotować dane wejściowe, wykonaj następujące czynności:

1. pobierz bibliotekę open source Crypto++ (www.cryptopp.com),
2. zbuduj cryptest.exe,
3. przetestuj bibliotekę za pomocą `cryptest.exe v`,
4. jeśli wszystkie testy zostaną zaliczone, wyświetli się komunikat „All tests passed!”,
5. wygeneruj plik binarny,

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

6. wywołaj cryptest.exe z parametrami:
   `cryptest.exe ae <HexKey> <HexIV> Project.bin Firmware.aes`

Do projektu dołączono plik wsadowy, aby usunąć wszelkie wątpliwości dotyczące formatu klucza.

Program `cryptest.exe` wygeneruje plik `*.aes`, który będzie użyteczny dla urządzeń z peryferiami AES.

6.4.6. Konfigurowanie deszyfrowania

Dodaj lub usuń znacznik kometarza #define ENCRYPT w projekcie. Będzie działać tylko na urządzeniach z peryferiami AES (w tym przypadku STM32L083, STM32L486, STM32G484).

Zmień klucz zastępczy i wektor inicjalizacji dowolną inną wartością w pliku `main.c`, aby spersonalizować projekt.

6.4.7. Ograniczenia prostego rozwiązania

Podstawowe rozwiązanie szyfrowania symetrycznego zastosowane w tym przykładzie ma pewne ograniczenia.

Używa tego samego klucza dla wszystkich współpracujących urządzeń. Powszechne stosowanie jednego klucza sprawia, że system kryptograficzny jest bardziej podatny na atak. Nie tylko daje to atakującemu więcej możliwości uzyskania tajnego klucza, ale po ujawnieniu klucza wszystkie urządzenia są otwarte. Ryzyko to można częściowo zmniejszyć, generując klucz przy użyciu pewnych właściwości kodu, na przykład wersjonowania.

Ponadto uwierzytelnianie opiera się wyłącznie na znajomości tajnego klucza, nie ograniczając całkowicie możliwości dostarczenia nieoryginalnego kodu.

Fakt, że kilka pierwszych bajtów kodu (składających się ze początkowego wskaźnika stosu i tabeli wektorów) jest w większości przewidywalny (atak „znany tekst jawny”) może doprowadzić do osłabienia właściwości kryptograficznych. Tę słabość można łatwo rozwiązać, dodając losową sekwencję (tzw. „sól”) do „zwykłego tekstu” przed szyfrowaniem. Następnie „sól” jest ignorowana po odszyfrowaniu.

6.4.8. Inne opcje kryptografii

Ochrona kodu to nie tylko kwestia algorytmu szyfrowania, ale raczej złożony system kryptograficzny, obejmujący wymogi bezpieczeństwa wdrożone we wszystkich fazach życia produktu, od projektowania i rozwoju, przez produkcję i serwis, aż po dezaktywację i recykling.

Role uwierzytelniania i ochrona integralności również muszą zostać poważnie wzięte pod uwagę.

6.5. Inne opcje realizacji aktualizacji

W tej sekcji opisano różne techniki często stosowane przy wdrażaniu mechanizmów aktualizacji w terenie, ale nieużywane w przykładzie X-CUBE-DBFU, aby zachować prostotę i ogólność.

6.5.1. Plik aktualizacji pola

W naszym przykładzie plik załadowany do systemu jest zwykłym plikiem binarnym, co najwyżej zaszyfrowanym. Zwykle nie jest to zalecane. Plik powinien być przynajmniej oznaczony wersją i identyfikacją produktu, aby zapobiec ładowaniu niekompatybilnego kodu do produktu, głównie uniknąć przypadkowego obniżenia wersji lub tylko dla dodatkowej wygody.

Pakiet aktualizacji zbiorowej zwykle zawiera nie tylko kod, ale także dane. Może istnieć sekcja zawierająca nową, zaktualizowaną konfigurację i sekcja zawierająca wartości inicjujące dla nowo dodanych zmiennych RAM.

W bardziej złożonych systemach pojedynczy pakiet może również zawierać dane, które powinny zostać przesłane do innych układów na PCB po udanej aktualizacji.

Proces aktualizacji na żywo może również zidentyfikować sekcję kodu, którą należy natychmiast skopiować do pamięci RAM, i punkt przejścia dla kodu z działaniami po aktualizacji, które zostaną wykonane tylko raz i zadbają o przeniesienie i porządkowanie.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.

6.5.2. Dane w NVM

Innym popularnym powodem korzystania z podwójnego banku jest użycie drugiego banku do danych, często przy użyciu algorytmu emulacji EEPROM. Możliwe jest połączenie mechanizmu aktualizacji w locie opisanego w tym dokumencie z emulacją pamięci EEPROM, jak opisano w AN4894.

7. Wnioski

Funkcja podwójnego banku, gdy jest właściwie używana, ma kilka zalet w szerokim zakresie zastosowań.

Przykład i opis zawarty w tej nocie aplikacyjnej obejmują tylko podstawowe aspekty, ale służą jako funkcjonalne elementy składowe dla realnych projektów.

Bardzo ważne jest dokładne przetestowanie powstałego rozwiązania oraz posiadanie solidnego i niezawodnego kodu, który można aktualizować na bieżąco.

Projekt pt. ***„Prace rozwojowe oraz testy w warunkach rzeczywistych autonomicznego i inteligentnego sterownika”*** nr Umowy z Województwem Łódzkim, w imieniu którego działa Centrum Obsługi Przedsiębiorcy: ***RPLD.01.02.02-10-0006/18-00*** realizowany w ramach Poddziałania I.2.2 Regionalnego Programu Operacyjnego Województwa Łódzkiego na lata 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Rozwoju Regionalnego.