

Technologia GSM w elektronice (15)



Obsługa błędów i debugowanie aplikacji Open AT

W tym artykule omówimy zagadnienia związane z obsługą sytuacji wyjątkowych i błędów oraz przedstawimy sposób debugowania aplikacji Open AT, które to zagadnienia są jednymi z ważniejszych podczas tworzenia aplikacji.

Na etapie tworzenia i testowania aplikacji możemy korzystać z makra TRACE (makro to zostało opisane w jednym z pierwszych odcinków cyklu). Jednak czasem zdarza się tak, że pomimo iż urządzenie przeszło wszystkie testy, to jednak po jakimś czasie pracy pojawiają się problemy, gdy urządzenie znajduje się w terenie.

Obsługa błędów

Programowalne moduły i modemy Sierry Wireless zostały wyposażone w mechanizm automatycznego logowania błędów do pamięci nieulotnej EEPROM. W wypadku, gdy z jakiejś przyczyny wystąpił wyjątek krytyczny, kod tego błędu oraz dodatkowe informacje zostają zapisane w pamięci nieulotnej. Możliwe wyjątki, które mogą wystąpić, to między innymi restart modułu przez mechanizm *watchdog* (przerwanie martwej pętli),

nieuprawniony dostęp do pamięci, dzielenie przez zero i inne. Bardzo dobrym sposobem na zapoznanie się z tym zagadnieniem jest uruchomienie aplikacji o nazwie „Bug” znajdującej się w przykładach do *Developer Studio*. Po utworzeniu i skompilowaniu aplikacji wgrujemy ją do modułu i uruchamiamy. Daje nam ona możliwość symulowania błędów różnego typu za pomocą komendy AT. Szczegóły możemy znaleźć w pliku *bug.html* dostępnym w katalogu projektu.

Po wydaniu komendy *AT+BUG=0* powinniśmy zobaczyć rezultat jak na **rysunku 1**.

Efektom działania komendy jest wykonywanie pętli nieskończonej, która po 5 sekundach powoduje restart modułu wywołany przez mechanizm *watchdog*. Informacja o tym jest oczywiście wyświetlana w oknie *Trace*, a trakcie ponownego startu systemu zmienna informująca o warunkach startu systemu – *InitType* – przyjmuje wartość

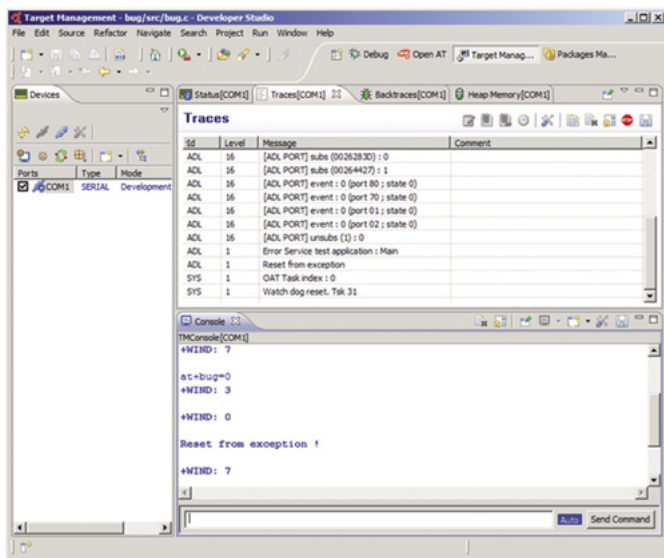
Dodatkowe materiały na CD/FTP:
<ftp://ep.com.pl>, user: 15031, pass: 40nep417
 • poprzednie części kursu

ADL_INIT_REBOOT_FROM_EXCEPTION. Informacja o błędzie powodującym restart systemu jest również zapamiętywana w pamięci EEPROM modułu. Można ją odczytać albo za pomocą *Developer Studio*, albo odpowiednich funkcji API.

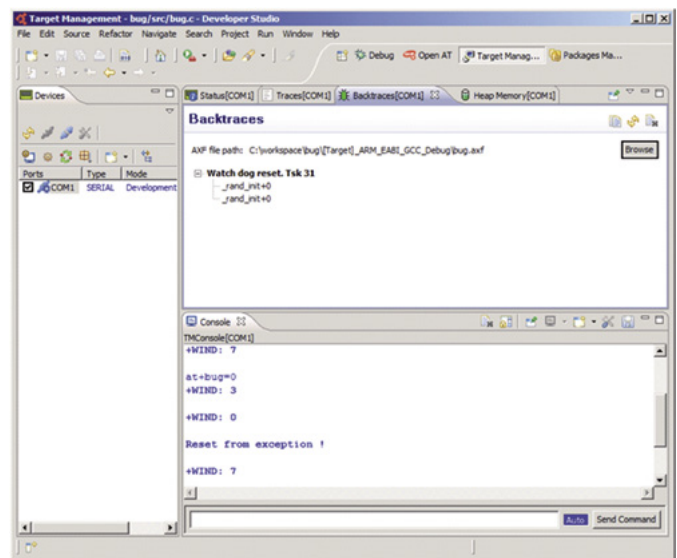
Aby odczytać zarejestrowane błędy za pomocą *Developer Studio*, należy połączyć się z modulem i utworzyć projekt aplikacji, która jest w danym momencie jest wgrana do modułu. Teraz przechodzimy do zakładki *Target Management*, a następnie do zakładki *Backtraces*. Za pomocą przycisku *Browse* należy załadować odpowiedni plik AXF, znajdujący się w katalogu z naszym projektem. Następnie naciskamy przycisk *Odśwież* (przycisk w postaci dwóch żółtych strzałek). Po chwili powinny pojawić się in-

Listing 1. Przykład procedury do odczytu *Backtraces*

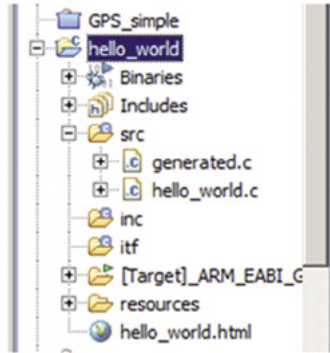
```
s32 sReturn;
do{
    s32 BufferSize;
    u8 * Buffer;
    s8 AnalysisHandle = adl_errStartBacktraceAnalysis();
    BufferSize = adl_errRetrieveNextBacktrace ( AnalysisHandle, NULL, 0 );
    Buffer = adl_memGet ( NextBufferSize );
    sReturn = adl_errRetrieveNextBacktrace ( AnalysisHandle, Buffer, BufferSize );
};
    zapisz_bufor(Buffer, BufferSize);
}while (sReturn != ADL_RET_ERR_DONE);
```



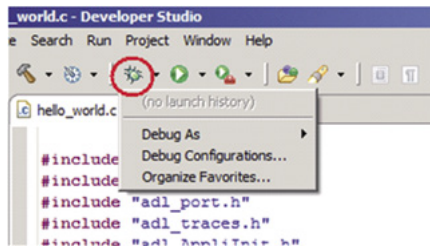
Rysunek 1. Efekt celowo wywołanego błędu



Rysunek 2. Odczyt błędów z pamięci EEPROM



Rysunek 3. Wybór aplikacji

Rysunek 4. Okno *Debug Configuration*

formacje o błędach odczytanych z modułu (rysunek 2). Zazwyczaj pojawia się informacja z jakim rodzajem wyjątku mamy do czynienia (w naszym przypadku *Watchdog reset*) oraz wartości istotnych rejestrów. W niektórych przypadkach, gdy miejsce wystąpienia błędu da się konkretnie wskazać, pojawia się też link, po kliknięciu którego zostajemy przeniesieni do miejsca w programie źródłowym, w którym wystąpił wyjątek systemu (jeśli mamy otworzony właściwy projekt), co znacznie ułatwia odszukanie przyczyny błędu.

Po analizie i dokonaniu poprawek w kodzie, przed ponownym oddaniem urządzenia do eksploatacji, błędy zapisane wcześniej w pamięci EEPROM powinny zostać z niej wykasowane.

Błędy zapisane w pamięci EEPROM można również odczytać za pomocą funkcji API, a następnie wysłać je za pomocą TCP lub zapisać w pliku na serwerze FTP. Przykład procedury do odczytu *Backtraces* zamieszczono na listingu 1.

Każde wywołanie funkcji *adl_errRetrieveNextBacktrace()* pobiera informacje tylko o jednym, kolejnym błędzie z pamięci. Dlatego wywołanie trzeba umieścić w pętli i czekać aż funkcja zwróci wartość *ADL_RET_ERR_DONE* świadcząca, że nie pozostało nic do odczytania. Dane pobrane do *Buffer* zostają przekazane do funkcji *zapisz_bufor()*, która w zależności od potrzeb dokonuje odpowiedniego ich przetworzenia. Sposób odczytu błędów jest również pokazany w przykładzie „Bug” (komenda *AT+GETBKTRC*).

Serwis *Error Managment* daje również programiście możliwość samodzielnego generowania wyjątków. Służy do tego funkcja *adl_errHalt()*. Argumentami funkcji są kod

Listing 2. Deklarowanie handlera do obsługi błędów

```
bool MyErrorHandler ( u16 ErrorID, ascii * ErrorStr )
{
...
<obsługa błedu>
return TRUE; //FALSE jesli reset nie powinien być wykonany
}

const ascii * MyErrorString = „Application Generated Error”;

void main_task ( void )
{
// Subscribe to error service
adl_errSubscribe ( MyErrorHandler );
// wywołaj blad
adl_errHalt ( ADL_ERR_LEVEL_APP + 1, MyErrorString );
}
```

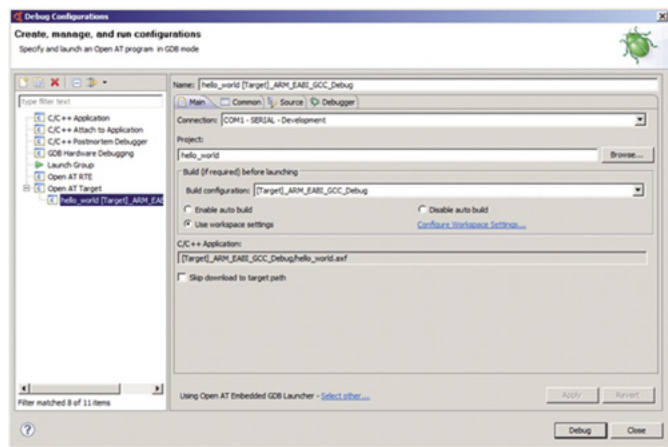
błędu (powinien być równy lub większy od *ADL_ERR_LEVEL_APP*) oraz tekst opisujący błąd. Informacje te, jak w wypadku błędu systemowego, będą zapisane w pamięci EEPROM. Natomiast jeśli za pomocą funkcji *adl_errSubscribe()* zadeklarujemy handler do obsługi błędów, to każdorazowo po wystąpieniu błędu ten handler będzie wywoływany przez system. W funkcji handlera mamy możliwość ustalenie sposobu reakcji na błąd. Jeżeli handler po obsłudze błędu zwróci wartość TRUE, to nastąpi zapisanie informacji o błędzie w pamięci EEPROM i restart modułu, natomiast jeśli zwracaną wartością będzie FALSE, to informacje nie zostaną zapisane w pamięci EEPROM, a modem nie wykona restartu (listing 2).

Debugowanie aplikacji

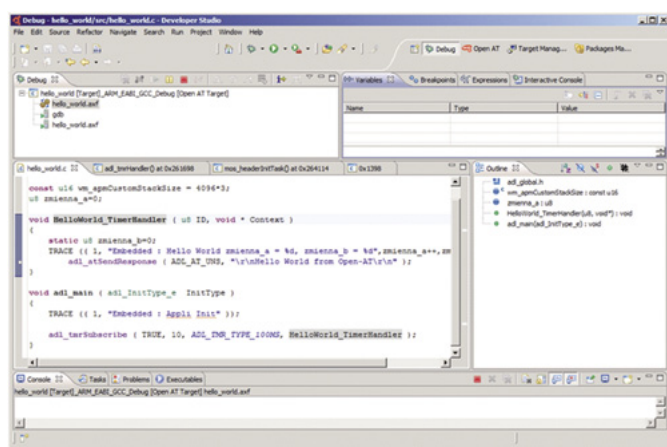
Wykorzystując środowisko programistyczne *Developer Studio* w wersji od 2.35 programista ma również możliwość debugowania aplikacji. Funkcjonalność ta wykorzystuje tryb *remote mode* debugera GNU: gdb. Tryb ten daje możliwość zatrzymywania aplikacji, obserwowania zmiennych oraz zakładania pułapek (breakpoint'ów). Poniżej w kilku krokach zaprezentowano sposób, jak wykorzystać debugowanie dla aplikacji *Hello-World*.

Na początek wgrzywamy odpowiednią aplikację do modułu i uruchamiamy. Moduł powinien pracować w trybie *Development Mode*, tak aby widoczne były komunikaty w konsoli (kolor niebieski) oraz *Trace*.

REKLAMA



Rysunek 5. Okno otwierane po wyborze *Open AT* → *New*



Rysunek 6. Okno *Debug Perspective*

Przechodzimy do zakładki *Open AT* i zaznaczamy w *Project Explorer*'ze właściwą aplikację (rysunek 3).

Rozwijamy listę przy przycisku *Debug* i wybieramy *Debug Configuration* (rysunek 4).

Klikamy prawym przyciskiem na *Open AT Target* i wybieramy *New*. Powinno pojawić się okno, jak na rysunku 5. Sprawdzamy czy ustawienie portu szeregowego jest prawidłowe. Powinny być wybrany ten sam port COM, do którego podłączony jest zestaw z modulem. Jeśli wszystko jest w porządku naciskamy *Debug*.

Zostaniemy zapytani czy ponownie wgrać aplikację do modułu, co nie jest konieczne oraz czy otworzyć *Debug perspective*, na co zgadzamy się naciskając *Yes*. Powinniśmy zobaczyć okno jak na rysunku 6. Teraz nasza aplikacja pracuje w trybie *Debug*. Aplikację możemy zatrzymywać za pomocą przycisku *Suspend* (ikona pauzy)

oraz wznawiać (ikona odtwarzania), a także przechodzić do zakładki *Target Management* i obserwować *TRACE*.

Kolejnym krokiem może być ustawienie pułapki (breakpoint). W tym celu klikamy prawym przyciskiem (rysunek 7) i klikamy *Toggle Breakpoint*. Aplikacja zatrzyma się po osiągnięciu linii kodu, w której ustawiliśmy pułapkę. Mamy wtedy możliwość podejrzania rejestrów mikrokontrolera (niestety na razie nie ma możliwości ich zmiany).

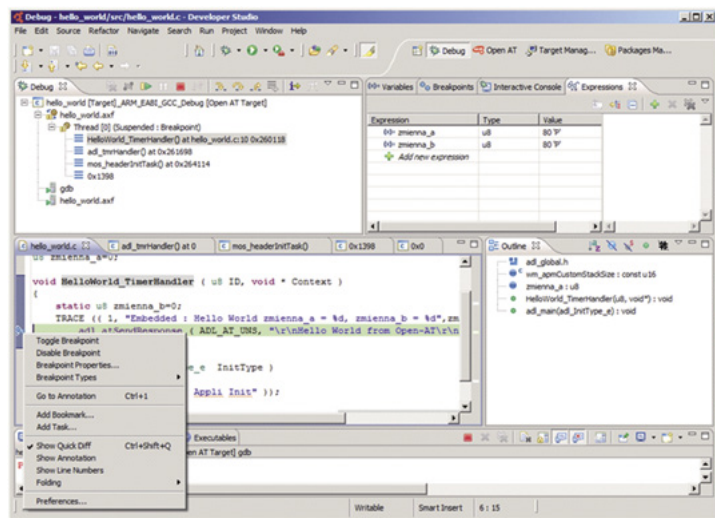
W zakładce *Expression* możemy obserwować wartości zmiennych. Jeśli zakładka jest nieaktywna, to okienko *Expression* włączamy poprzez wybranie odpowiedniej pozycji z menu dostępnym po naciśnięciu przycisku z symbolem „+” znajdującego się w lewym dolnym rogu.

Opcją dodatkową związaną z zatrzymaniem programu jest możliwość podania warunku, przy którym nastąpi wstrzymanie jego działania. W tym celu klikamy prawym

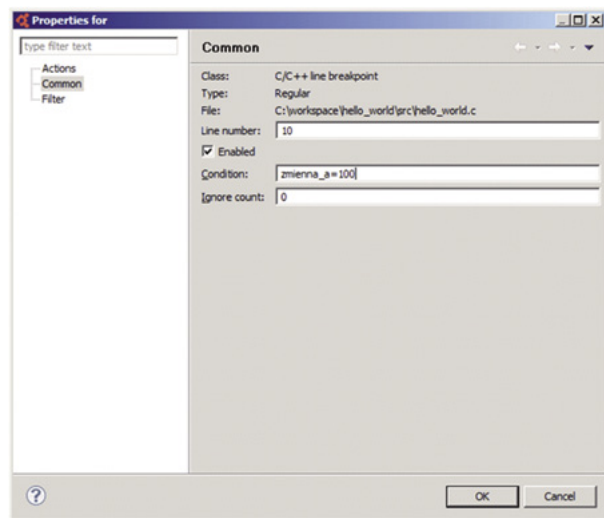
przyciskiem myszy na ustawionej pułapce i wybieramy pozycję *Breakpoint Properties*. Pojawi się ono jak na rysunku 8. W polu *Condition* wpisujemy warunek zatrzymania się programu. Po osiągnięciu zadanego warunku program zatrzyma się na pułapce.

Debugger oraz mechanizm *Backtraces* są bardzo przydatnym uzupełnieniem tego, co uzyskiwaliśmy do tej pory za pomocą makr *TRACE* oraz *DUMP*. Sprawne posługiwanie się nimi znacznie upraszcza proces odnajdywania oraz poprawiania błędów w aplikacji. Więcej informacji na temat produktów *Sierra Wireless* można znaleźć na stronach producenta: www.sierrawireless.com lub kontaktując się z firmą *ACTE Sp. z o.o.*, która jest oficjalnym dystrybutorem opisywanych produktów oraz zapewnia pełne wsparcie techniczne.

Adrian Chrzanowski
Acte Sp. z o.o.



Rysunek 7. Ustawienie pułapki (*toggle breakpoint*)



Rysunek 8. Właściwości pułapki (*breakpoint properties*)

REKLAMA

<http://forum.ep.com.pl>