Łapiąc „pluskwy”

TEMAT NUMERU

Dobierając mikrokontroler do aplikacji bierzemy pod uwagę różne czynniki. Najczęściej są to rdzeń, pojemność pamięci, układy peryferyjne i cena. Wydawać by się mogło, że to wyczerpuje kryteria. Jest jednak jeszcze jeden aspekt, którego nie należy pomijać – zintegrowane w układzie mechanizmy debugera (bug – pluskwa) decydujące o możliwościach i wygodzie testowania oprogramowania aplikacyjnego w fazie jego tworzenia i uruchamiania.

Nawet w kontekście powszechnej unifikacji rdzeni mikrokontrolerów często, w ramach tej samej rodziny, od tego samego producenta, są układy integrujące zaawansowane mechanizmy debugera jak i układy pewnych funkcji pozbawione. Standardy zostawiają bowiem niektóre opcje otwarte, do wyboru wdrażającego. Zanim przedstawimy jakie „narzędzia” zapewniają współczesne ARMy, rzućmy okiem wstecz. W „zamierzchłych” czasach do uruchamiania oprogramowania aplikacyjnego w układzie docelowym, w rzeczywistym środowisku używaliśmy emulatorów sprzętowych ICE (*In-Circuit Emulator*) i debugerów programowych ICD (*In-Circuit Debbuger*). Pierwsze, skomplikowane i drogie, były niewątpliwie najlepszymi kiedykolwiek dostępnymi narzędziami. Pozwalały zastąpić procesor w układzie docelowym wtykiem sondy działającej z punktu widzenia układu aplikacyjnego jak rzeczywisty procesor i wysyłającej w tle (bez angażowania czasu i zasobów procesora) do komputera nadzorującego informację o jego stanie. Mechanizmy debugera (wstrzymanie procesora, praca krokowa, warunkowe pułapki, odczyt rejestrów i pamieci, pamięć śladu, …) były realizowane sprzętowo poza procesorem. Drugie, znacząco tańsze, były pierwotnie rozwiązaniami czysto programowymi opartymi o instalowany w układzie docelowym monitor, który programowo realizował podstawowe mechanizmy debugera i za pośrednictwem standardowego interfejsu szeregowego procesora wysyłał odczytywane dane o stanie do komputera. Aby wyeliminować podstawową wadę – blokowanie na czas uruchamiania zasobów mikrokontrolera scalono w strukturze standardowych, seryjnie produkowanych mikrokontrolerów podstawowe mechanizmy i niezależny interfejs emulatora tworząc zintegrowany blok funkcjonalny…

OCD (*On-Chip Debugger*)

Blok OCD oczywiście nie zapewnia pełnego spektrum mechanizmów, które były dostępne w dobrych sprzętowych emulatorach ICE. Projektanci szczególnie w obszarach wrażliwych na jakość i niezawodność kodu (automatyka, medycyna, motoryzacja, lotnictwo, uzbrojenie) tęsknią za starymi, dobrymi czasami. Jednak emulatory sprzętowe odchodzą do lamusa, a metoda uruchamiania ICD oparta o wbudowane w strukturę i rozwiązania sprzętowe OCD zastępuje technologię ICE.

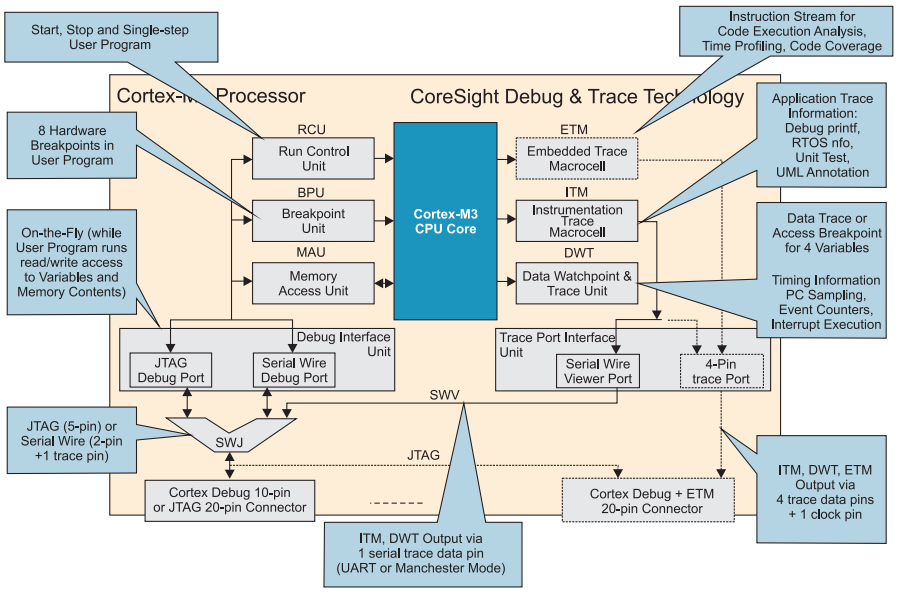
Typowy zintegrowany w strukturze półprzewodnikowej moduł OCD zawsze zawiera najczęściej używane, podstawowe bloki funkcjonalne emulatora. W mikrokontrolerach ARM są to:

- RCU (*Run Control Unit*) – blok kontroli przetwarzania odpowiedzialny za uruchamianie i wstrzymywanie procesora oraz realizację pracy krokowej.
- BPU (*Breakpoint Unit*) – blok pułapek sprzętowych pozwalający zdefiniować kilka adresów zatrzymania procesora na cyklu pobrania instrukcji.
- MAU (*Memory Access Unit*) – blok dostępu do rejestrów i pamięci wewnętrznych mikrokontrolera umożliwiający ich zapis lub odczyt.
- DIU (*Debug Interface Unit*) – blok dedykowanego interfejsu debugera pozwalający na kilku liniach szeregowo transmitować dane między wbudowanym debugerem OCD, a komputerem nadzorującym.

Do uruchamiania konieczny jest układ pośredniczący pomiędzy komputerem nadrzędnym, a blokiem OCD. Od strony komputera taki adapter najczęściej używa interfejsu USB, a w bardziej rozbudowanych rozwiązaniach alternatywą jest Ethernet pozwalający na zdalne śledzenie oprogramowania z miejsc oddalonych od uruchamianego urządzenia. Od strony uruchamianego układu najczęściej stosowane są złącze i interfejs JTAG *(Joint Test Action Group*). Trzeba przy tym zwrócić uwagę, że JTAG stał się synonimem debugera OCD, co może prowadzić do nieporozumień. Jest to bowiem interfejs pierwotnie zdefiniowany i dedykowany do automatycznego testowania pakietów elektronicznych w technice Boundary-Scan (norma IEEE 1149.1), a tylko wykorzystywanym w technice ICD. Nie można więc jednoznacznie wnioskować czy układ ma wbudowany debuger OCD na podstawie informacji, że ma wyprowadzenia interfejsu JTAG.

Spójrzmy dalej na OCD przez pryzmat najpopularniejszych obecnie na rynku mikrokontrolerów nowej generacji ARM Cortex-M3. Zaimplementowano w nich rozbudowane mechanizmy debugera określane mianem *CoreSight*.

W porównaniu do wcześniej wymienionych, podstawowych bloków OCD operujących via JTAG, CoreSight daje przede wszystkim możliwości ustawiania pułapek, podglądu stanu procesora i pamięci w sta-

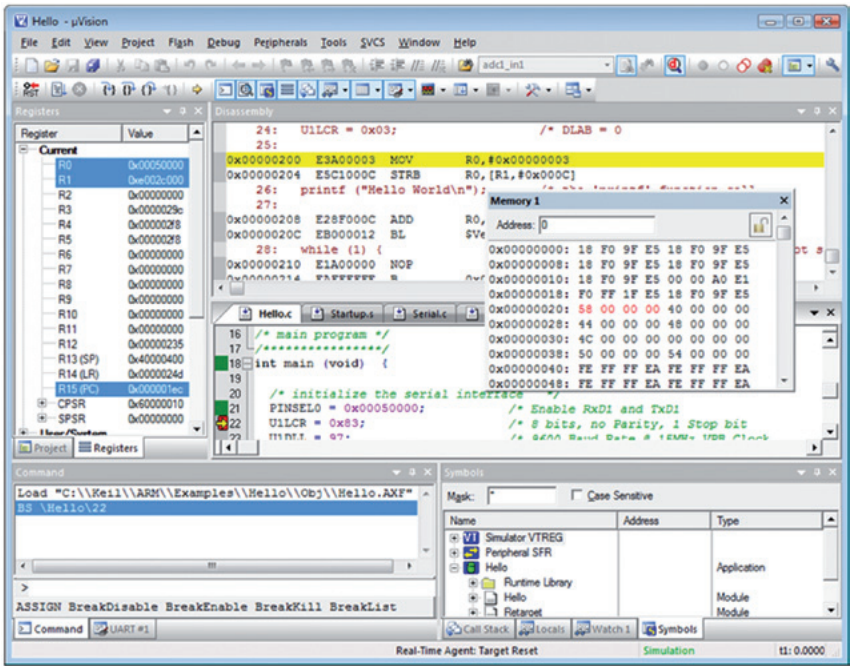


Rysunek 1. Zintegrowane mechanizmy debugera CoreSight w architekturze ARM Cortex-M3

nie aktywnym (*on the fly*). Rozbudowuje mechanizmy debugera poprzez implementację mechanizmów śladu, wprowadza nowy interfejs Serial Wire złożony z dwuliniowego SWD (*Serial Wire Debug*) obsługującego bloki debugera i jednoliniowego SWV (*Serial Wire Viewer*) obsługującego bloki śladu. Wszystko to zbliża pod względem możliwości funkcjonalnych technikę uruchamiania oprogramowania ICD do ICE.

CoreSight, w porównaniu do bazowej architektury OCD, wprowadza (**rysunek 1**):

- DWT (*Data Watchpoint & Trace*) – blok umożliwiający śledzenie podczas realizacji programu licznika instrukcji, zdarzeń i przerwań oraz realizację pułapek na dostępie do pamięci pod zadeklarowanym adresem.
- ITM (*Instrumentation Trace Macrocell*) – blok umożliwiający śledzenie „w locie” zmiennych programu oraz sygnalizowanie zdarzeń i przejść w programie w technice *debug printf* tzn. poprzez wstawianie do kodu źródłowego instrukcji *printf(<format>,<zmienna>)*.
- ETM (*Embedded Trace Macrocell*) – opcjonalny blok umożliwiający śledzenie wykonywania instrukcji na potrzeby analizy przetwarzania, przy czym transmisja danych z tego modułu wymaga czterech dodatkowych przewodów.



Rysunek 2. Środowisko IDE µVision4 firmy KEIL

Te dodatkowe bloki są obsługiwane przez tzw. TPIU (*Trace Port Interface Unit*), przy czym sygnały *Debug Interface* i *Trace Port Interface* mogą być wyprowadzone na jedno z trzech złącz: JTAG (20-pin 0,10”), Cortex Debug (10-pin 0,05”) lub Cortex Debug+ETM (20-pin 0,05”). Dwa ostatnie są złączami miniaturowymi dopuszczonymi przez standard Cortex, zajmującymi mniej miejsca na płytce drukowanej niż standardowe złącze JTAG.

Jak wspomniano wcześniej, wszystkie wbudowane mechanizmy OCD są obsługiwane przez komputer nadrzędny za pośrednictwem specjalizowanego urządzenia. Z punktu widzenia możliwości funkcjonalnych i wygody użytkowania kluczowe znaczenie ma więc oprogramowanie...

Rysunek 3. Realizacja *Execution Profiler'a* w środowisku Keil µVision

IDE (*Integrated Development Enviroment*)

IDE jest zintegrowanym środowiskiem programowym dla komputera nadrzędnego, obsługującym OCD za pośrednictwem adaptera USB/Ethernet-JTAG/SW i wizualizującym na ekranie zebrane dane (**rysunek 2**). Wygoda obsługi IDE i sposób wyświetlania danych decydują o użyteczności całego rozwiązania debugera. Istotne jest, aby system okien był maksymalnie elastyczny i konfigurowany przez operatora i aby ze względu na dużą ilość analizowanych danych umożliwiał jednoczesną pracę na kilku monitorach.

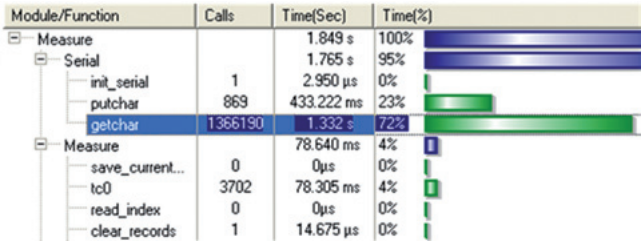
IDE zapewnia interpretację danych odbieranych z OCD. Typowo:

- integruje wizualizację z edytorem i kompilatorem języka C,
- zapewnia podgląd i modyfikację programu, rejestrów, pamięci, układów peryferyjnych, sygnałów we/wy, zmiennych programowych z możliwością symbolicznych odwołań,
- umożliwia debuging na poziomie assemblera i języka C,
- zarządza projektami i procesem uruchamiania poprzez obsługę poleceń operatora i komunikatów.

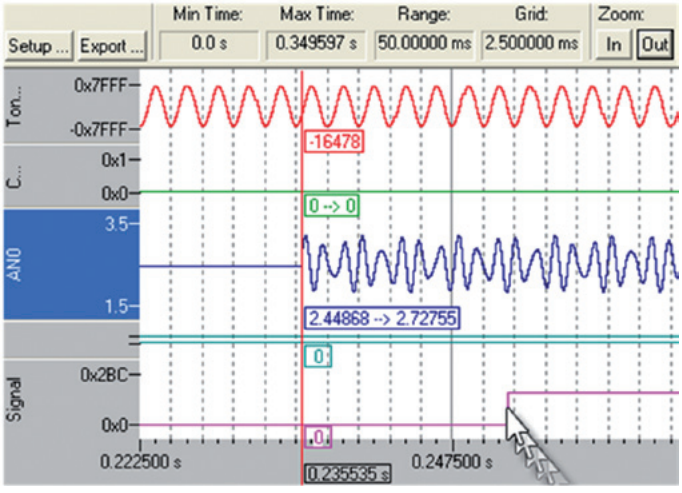
A czasami:

- integruje symulator programowy mikrokontrolera,
- IDE we współpracy z rozbudowanym blokiem OCD (np. CoreSight z modułem ETM) może realizować zaawansowane mechanizmy analiz: śladu (*Trace*), pokrycia kodu (*Code Coverage*), przetwarzania (*Performance Analysis*), profilowania oprogramowania (*Program Profiling*), stanu sygnałów (*Logic Analyzer*).

Mikrokontrolery są dziś wszechobecne. We wrażliwych na błąd aplikacjach jak np. w lotnictwie, motoryzacji czy medycynie kluczowa jest jakość kodu. Tam nie wystarcza zapewnienie programisty, że wszystko zostało sprawdzone. Obowiązujące normy wymuszają udokumentowanie wykonanych testów. Coraz większe zastosowanie mają więc wymienione powyżej zaawansowane mechanizmy debugerów. Przyjrzyjmy się im bliżej.



Rysunek 4. Realizacja *Performance Analyzer'a* w środowisku Keil µVision



Rysunek 5. Realizacja *Logic Analyzer'a* w środowisku Keil µVision

Pokrycie kodu jest techniką systematycznego testowania oprogramowania polegającą na ocenie, które obszary kodu zostały wykonane i sprawdzone w trakcie uruchamiania. Podstawowymi kryteriami oceny pokrycia kodu testami mogą być:

- Pokrycie funkcji – tzn. czy każda funkcja i każdy podprogram zostały wywołane,
- Pokrycie instrukcji – tzn. czy każda instrukcja została wykonana,
- Pokrycie warunków – tzn. czy każdy warunek logiczny został sprowadzony do wartości prawda i fałsz,
- Pokrycie rozgałęzień – tzn. czy każde rozgałęzienie zostało sprawdzone dla warunków prawda i fałsz.

W praktyce wygląda to tak, że program aplikacyjny jest wykonywany pod nadzorem środowiska IDE, rejestrowany jest przy tym ślad, a każda wykonana instrukcja jest mapowana zwrotnie w kodzie źródłowym. Pozwala to na identyfikację obszarów kodu, które nie były wykonywane, a tym samym testowane. Na tej podstawie można modyfikować podejście i docelowo opracować kompletny zbiór testów regresywnych.

Innym istotnym dla optymalizacji oprogramowania jest tzw. *Program Profiler* (**rysunek 3**). Jest to narzędzie analizy przetwarzania, które w podstawowej wersji rejestruje sumaryczny czas wykonania poszczególnych rozkazów. Pozwala identyfikować krytyczne czasowo obszary oprogramowania, optymalizować program poprzez reedycję kodu źródłowego i tym samym skrócić czas potrzebny na wykonanie podprogramów, np. procedur obsługi przerwań.

Oczywistym jest, że mechanizmy Profiler i Code Coverage można bez problemu zaimplementować w symulatorze programowym mikrokontrolera, o ile odtwarza on rzeczywiste relacje czasowe wykonywanych instrukcji. W technologii ICD wymaga już zwrotnej informacji z systemu docelowego do komputera nadzorującego o wykonywanych instrukcjach. Przy czym, aby zaawansowana analiza przetwarzania miała sens, musi być realizowana w czasie rzeczywistym, w tle działającego z rzeczywistą szybkością i w rzeczywistym środowisku mikrokontrolera. Tak więc w ARM'ach w debugerach OCD dopiero technologia CoreSight, a w szczególności wbudowany blok ETM, otworzyły możliwości analizy przetwarzania.

Analizator przetwarzania (**rysunek 4**) pozwala na rejestrowanie czasu wykonywania zadeklarowanych procedur lub fragmentów programu i wyświetlanie tej informacji w postaci bezwzględnego, sumarycznego czasu lub jego procentowego udziału w globalnym czasie przetwarzania, wizualizowanego dodatkowo wykresem paskowym. Taka informacja pozwala łatwo zidentyfikować krytyczne czasowo fragmenty programu, które należy poddać optymalizacji poprzez reedycję kodu źródłowego.

Rysunek 6. Adapter ULINK Pro firmy Keil

Rysunek 7. Analizator iC5000 firmy iSYSTEM

Analizator stanów logicznych (**rysunek 5**) pozwala na rejestrowanie podczas pracy CPU zmiennych programu i sygnałów oraz na wygodną wizualizację ich przebiegów czasowych w postaci graficznej. Pozwala analizować zależności czasowe między poszczególnymi sygnałami i identyfikować instrukcje programu aplikacyjnego zmieniające stan tych sygnałów w miejscach wskazanych przez operatora kursorem.

Na koniec rozważań o zaawansowanych mechanizmach debugera pewien pomysł na rozwiązanie problemu ich braku w mikrokontrolerach bez wbudowanego bloku ETM. Został on inspirowany układami Bond-Out i faktem, że producenci układów półprzewodnikowych starają się zachować kompatybilność funkcjonalną i mechaniczną między różnymi mikrokontrolerami w ramach jednej rodziny. Często na czas uruchamiania urządzenia można zastosować w układzie aplikacyjnym większy procesor z rozbudowanym blokiem OCD, o ile tylko jest zgodny z docelowym. Brak kompatybilności mechanicznej da się przy tym obejść stosując adapter lub używając do celów uruchamiania prototypu wyposażonego w bardziej rozbudowany procesor.

Stwierdziliśmy, że możliwości współczesnych debugerów implikowane są przez cechy funkcjonalne wbudowanych OCD i środowiska IDE. Oczywistym jest, że aby wykorzystać te możliwości, obydwa czynniki muszą współgrać, czyli wymagają właściwego urządzenia sprzęgającego układ aplikacyjny z komputerem. Przedstawimy krótko dwa rozwiązania wzorcowe.

ULINK

ULINK-2, ULINK-Pro (**rysunek 6**) są produktami oferowanymi przez firmę Keil, będącą obecnie marką firmy ARM. Są uniwersalnymi adapterami sprzęgającymi uruchamiany układ z komputerem nadrzędnym. Do połączenia z układem docelowym służą interfejsy JTAG i SW, a z komputerem nadrzędnym – USB. Podobne adaptery są oferowane też przez inne firmy zajmujące się narzędziami dla mikrokontrolerów np. j-LINK firm IAR, SEGGER itp. za cenę kilkuset EUR, zbliżoną do ceny ULINK-2. Jednak najnowszy, droższy ULINK-Pro wyróżnia się na tym tle, ponieważ obsługuje pełny zakres mechanizmów *CoreSight* m.in. blok ETM i realizuje tzw. „ślad strumieniowy” (*Streaming Trace*). Odbiera z OCD z prędkością 100 MB/s strumień danych śladu, poddaje kompresji i wysyła do komputera z prędkością 25 MB/s. Odebrane przez komputer dane są pod nadzorem IDE buforowane na dysku twardym. Szybkości te są wystarczające do obsługi mikrokontrolerów Cortex-M pracujących z zegarem do 200 MHz. Pojemność bufora jest ograniczona tylko wielkością dysku. Ślad może więc być analizowany na bardzo dużej przestrzeni czasowej, a gromadzona informacja zawiera wszystkie dane potrzebne do śledzenia i analizy przetwarzania m.in: wartości licznika rozkazów, zapisy/odczyty danych, liczniki zdarzeń, informacje o cyklach i czasie wykonania instrukcji.

ULINK-Pro jest więc relatywnie tanim narzędziem pozwalającym w pełni wykorzystać w połączeniu ze środowiskiem µVision pakietu MDK-ARM, wszystkie wbudowane mechanizmy debugera technologii CoreSight.

iC5000

Innym rozwiązaniem jest pokazany na **rysunku 7** analizator iC5000 firmy iSYSTEM. Tym, co odróżnia go od wcześniej prezentowanego ULINK'a jest uniwersalność w sensie obsługi różnych rdzeni od różnych producentów. Rekonfigurowalny układ pozwala bowiem adaptować urządzenie do różnych bloków OCD i różnych interfejsów. Resztę zapewnia elastyczne środowisko programowe winIDEA (**rysunek 8**). I tak, iC5000 obsługuje rdzenie ARM, Power PC, Coldfire/S12/S08, TriCore/XC2000.

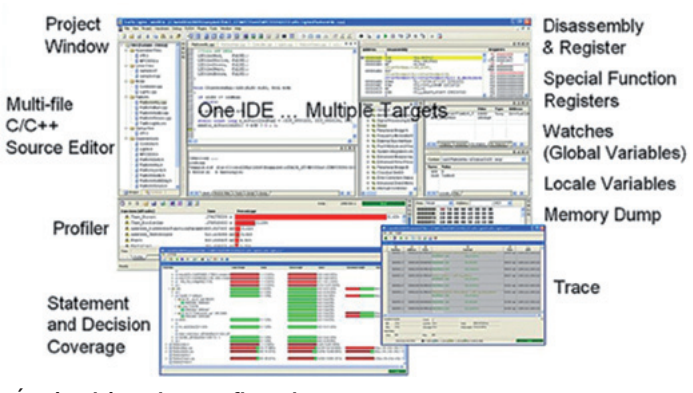
Analizator iC5000 jest w porównaniu z ULINK'iem bardziej rozbudowany sprzętowo. Zawiera m.in. bufory pamięci śladu zapewniające transmisję danych z maksymalną, dopuszczalną przez procesor prędkością. Z komputerem nadrzędnym może komunikować się via USB lub Ethernet. W drugim przypadku jest możliwość uruchamiania oprogramowania z miejsc odległych od prototypu. Ponadto, opcjonalny moduł we./wy. umożliwia monitorowanie sygnałów cyfrowych i analogowych w układzie aplikacyjnym oraz generowanie wymuszeń. Środowisko winIDEA zapewnia też bardziej zaawansowane mechanizmy analizy przetwarzania, a w szczególności analizy pokrycia.

Podsumowanie

Kiedyś wybór procesora do aplikacji był niezależny od możliwości debugingu. Wybieraliśmy najlepszy, dostępny procesor, a później staraliśmy się o jakikolwiek emulator lub częściej – z oczywistych powodów finansowych – zadawalaliśmy się metodą prób i błędów. Jak już zainwestowaliśmy w narzędzia i poznaliśmy jakąś rodzinę mikrokontrolerów, to przez lata obracaliśmy się w jej kręgu. Te czasy minęły. W wyniku galopującego postępu i unifikacji możemy bez problemu, stosując np. ARM'y, w każdym kolejnym projekcie wybierać inny, nowszy procesor i to każdorazowo innej marki. Obecnie bowiem, optymalizując koszty i rozmiary projektowanego urządzenia, szyjemy na miarę – dobieramy przy każdym projekcie mikrokontroler pod względem zasobów dokładnie do potrzeb aplikacji. Narzędzia uruchomieniowe, którymi przy tym dysponujemy, nie są tak rozbudowane jak kiedyś, lecz są za to bardziej uniwersalne i pokrywają szersze spektrum procesorów od różnych producentów. A przede wszystkim są łatwiej dostępne.

Postawiona w artykule teza, że w pewnym sensie każdorazowo kupujemy debuger razem z mikrokontrolerem, skłania przy nowym projekcie do zastanowienia się w jakim zakresie chcemy testować oprogramowanie i do analitycznego spojrzenia na to, jakie narzędzia rzeczywiście dostaniemy do ręki z wybranym mikrokontrolerem. Skłania czasami do wyboru bardziej rozbudowanego i droższego mikrokontrolera kontrolera, aby w newralgicznych pod względem jakości i niezawodności kodu aplikacjach, zapewnić sobie większe możliwości analizy oprogramowania. Od mikrokontrolera zależy bowiem debuger, a od debugera zależy nasz – najczęściej krytyczny – „time to market”.

Tadeusz Górnicki
WG Electronics Sp. z o.o.



Rysunek 8. Środowisko winIDEA firmy iSYSTEM