Andrzej BIEŃ, Jakub RZESZUTKO, Tomasz SZYMAŃSKI
AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE, Al. A. Mickiewicza 30, 30-059 Kraków
DELPHI POLAND S.A., ul. Podgórki Tynieckie 2, 30-399 Kraków

Systemy wbudowane do celów pomiarowo-kontrolnych w motoryzacji. Proces projektowania i wstępnej diagnostyki

Dr hab. inż. Andrzej BIEŃ

W Akademii Górniczo-Hutniczej w Krakowie pracuje od 1979 roku. Zajmuje się systemami pomiarowymi opartymi o cyfrowe przetwarzanie sygnałów. Skuteczność prowadzonych prac potwierdzają patenty i wdrożenia. Obszar zastosowań w gospodarce energetycznej pojazdów elektrycznych i diagnostyka odbiorników energii elektrycznej jest aktualnie prowadzoną tematyką badawczą.

e-mail: abien@agh.edu.pl

Mgr inż. Jakub RZESZUTKO

Urodzony 27 maja 1984 roku w Tarnowie. Od 2008 rozpoczął pracę w sektorze automatyki i elektroniki przemysłowej oraz podjął studia doktoranckie na Akademii Górniczo-Hutniczej w Krakowie. Obecnie pracownik firmy Delphi, z którą związany jest od trzech lat. Zajmuje się zarządzaniem projektami oprogramowania dla wiodących producentów samochodów.

e-mail: Jakub.Rzeszutko@delphi.com

Mgr inż. Tomasz SZYMAŃSKI

Urodził się 15 lipca 1988 roku w Tarnowskich Górach. W 2013 roku ukończył studia na wydziale Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej na Akademii Górniczo-Hutniczej w Krakowie. Obecnie pracownik firmy Delphi na stanowisku inżynier oprogramowania. Od trzech lat zajmuje się modelowaniem i automatyczną generacją kodu dla systemów wbudowanych.

e-mail: Tomasz.Szymanski@delphi.com

Streszczenie

W artykule zaprezentowano nowoczesne metody tworzenia aplikacji dla systemów wbudowanych do celów kontrolno-pomiarowych w przemyśle motoryzacyjnym. Przedstawiono podstawy procesu powstania oprogramowania zgodnego ze standardem SPICE. Opisano także wyzwania stojące obecnie przed programistami oraz współcześnie stosowane narzędzia do ich rozwiązywania. Artykuł prezentuje również symulację przykładowego modelu funkcjonalnego zaimplementowanego w środowisku Matlab.

Słowa kluczowe: AUTOSAR, Matlab, modelowanie, model V, Simulink, TargetLink, TPT, MIL, SIL, PIL, SPICE.

Embedded systems dedicated to measurement and control in the automotive industry. The design process and the initial diagnosis

Abstract

Two types of verification methods, i.e. MIL & SIL, are provided by TargetLink. In the current circumstances, due to higher car electrification, the complexity of the software developed for automotive industry is increased. This situation forces investments in new solutions and programming tools for the development and maintenance of the source code. This paper describes the foundations of software development compliant with the SPICE standard and its well-known model implementation, i.e. V Model (Fig. 1.). The authors focused on the description of the concept behind the application development with the aid of functional models (Fig. 5). Those models are designed in the MATLAB environment, followed by the automatic source code generation using TargetLink, which complies with the AUTOSAR standard. This standard allows automated model integration (applicable to functionalities, e.g. control of windscreen wipers or electric windows). Once designed and tested, the model can be reused in many other projects, which guarantees the identical functionality regardless of the chosen system architecture. This paper describes the simulation example of a model responsible for the car ambient temperature measurement. (Figs. 4, 5). In addition, the simulation covers the operation of the model. The simulation results are shown in Figs. 8 and 9. In order to perform runtime checks of the generated code for any faults, the authors used two types of verification methods provided by TargetLink, i.e. Model In the Loop & Software In the Loop.

Keywords: AUTOSAR, Matlab, V-model, Simulink, TargetLink, TPT, MIL, SIL, PIL, SPICE.

1. Wstęp

Sektor motoryzacyjny przeznacza obecnie olbrzymie nakłady finansowe na najnowsze zdobycze nauki i techniki aby zapewnić kierowcom i pasażerom pojazdów wygodę, komfort jazdy oraz bezpieczeństwo. Osiągane jest to między innymi przez zwiększenie ilości zainstalowanych czujników pomiarowych oraz systemów elektronicznych do ich obsługi. Coraz większa elektronizacja samochodów wymusza między innymi konieczność implementacji nowych sterowników, wzrost skomplikowania istniejących algorytmów sterujących oraz rozbudowanie ich funkcjonalności. Ma to olbrzymi wpływ na rosnącą złożoność powstającego oprogramowania. W związku z tym pojawiła się potrzeba wdrożenia narzędzi programistycznych, które umożliwią szybkie tworzenie kodu programu, charakteryzującego się łatwością przenoszenia na różne architektury sprzętowe oraz prostotą utrzymania i rozbudowy.

W tym celu do elektronicznych systemów samochodowych zaczęto stosować zaawansowane narzędzia programistyczne, m.in. MATLAB [9], TargetLink [6, 7, 8], Simulink [10], umożliwiające automatyczną generację kodu oraz przeprowadzanie automatycznych testów na podstawie zaprojektowanych wcześniej modeli funkcjonalnych. Model taki jest teoretycznym tworem reprezentującym rzeczywistość za pomocą zmiennych, logiki oraz zależności pomiędzy nimi, w którym zachodzą zjawiska opisane odpowiednimi pojęciami matematycznymi. Główną zaletą modelu w porównaniu do ręcznie przygotowanego kodu jest hierarchiczna struktura, która pozwala na zbieranie zagadnień w podsystemy. Dzięki temu inżynier może w pełni skupić się na realizacji funkcjonalności, a konstruowanie kodu w C pozostawić generatorowi.

Odpowiednio tworzone schematy blokowe stanowią intuicyjną reprezentację zawartej w nich funkcjonalności i mogą być traktowane, jako dokumentacja potrzebna dla automatycznego wygenerowanego kodu oraz jego testowania. Często zdarza się, że model funkcjonalny jest dostarczany przez klienta i traktuje się go jako wymaganie dla jakiejś funkcjonalności np. kontroli wycieraczek. Zaletą takiego rozwiązania jest fakt, iż producent samochodu może przekazywać swoje modele wielu dostawcom wykonującym oprogramowanie dla różnych modeli aut, mając gwarancję, że funkcjonalność dostarczonego oprogramowania będzie w każdym przypadku identyczna. Zadanie programistów sprowadza się do integracji kodu wygenerowanego na podstawie dostarczonego modelu z implementowanym systemem, lub zaprojektowania dodatkowych wymagań rozszerzających podstawową funkcjonalność modelu [3].

2. Proces wytwarzania oprogramowania dla systemów wbudowanych w branży motoryzacyjnej

Europejscy klienci z branży motoryzacyjnej wymagają by tworzone oprogramowanie odbywało się zgodnie z procesem SPICE (Software Process Improvement Capability dEtermination) [2]. SPICE dotyczy przede wszystkim systemu, oprogramowania, testowania i zarządzania projektem. Opisuje cykl życia produktu, począwszy od formułowania wymagań do analizy gwarancji. SPICE ocenia również poziom dojrzałości danego projektu poprzez tworzenie dokumentacji potwierdzającej spełnienie wyznaczonych standardów. Proces ten nakazuje spełnienie wymogów dotyczących:

- Podstawowej praktyki, która musi być wykonana dla każdego procesu,
- Dokumentów oraz danych, które muszą zostać wytworzone w wyniku każdej podstawowej praktyki,
- Atrybutów dla każdego produktu,
- Wykazywania, że projekt jest odpowiednio zarządzany,
- Wykazywania, że projekt jest odpowiednio wspierany.

W branży motoryzacyjnej cykl tworzenia aplikacji wbudowanej jest mocno sformalizowany i realizowany zgodnie z przyjętym procesem, np. Automotive SPICE [1]. Standard ten został zaproponowany przez Special Interest Group (SIG) zrzeszającą czołowych producentów samochodów. Konieczność jego stosowania wynika ze specyfiki branży motoryzacyjnej. Obecnie szacuje się, że 85% funkcjonalności nowoczesnych samochodów kontrolowana jest za pomocą oprogramowania. Wdrażając proces Automotive SPICE redukuje się niemal do zera prawdopodobieństwo pominięcia niektórych wymagań klienta. Jest to realizowane między innymi poprzez zapewnienie, iż każde z wymagań musi posiadać przypisany scenariusz testowy (ang. traceability). Proces wymusza również planowanie wszystkich zadań, analizę wymagań, tworzenie architektury i specyfikacji do programów, przeglądy kodu itd. Tym samym zmniejsza się prawdopodobieństwo wystąpienia błędów, a zarazem rośnie prawdopodobieństwo ich wykrycia [1]. Podejście takie jest niezbędne, w celu zapewnienia wysokiej jakości oprogramowania, w szczególności jeżeli kontroluje ono moduły odpowiedzialne za funkcje bezpieczeństwa.

Jednym z powszechnie stosowanych implementacji standardu Automotive SPICE jest tzw. model V (rys. 1), który opisuje kolejność wykonywania kolejnych kroków procesowych. W pierwszym etapie (Requirements Elicitation) klient przedstawia wszystkie wymagania dotyczące tworzonego przez siebie produktu. W następnym kroku wykonywana jest analiza wymagań systemu (System Requirements Analysis). Innymi słowy ze zbioru wszystkich wymagań wyodrębnia się jedynie te, które dotyczą tworzonego przez nas modułu. Oczywiste jest, że dla dostawcy systemu alarmowego nie są istotne wymagania dotyczące np. maksymalnego czasu zadziałania poduszek powietrznych.

W kolejnym etapie projektuje się architekturę systemową (System Architectural Design) wytwarzanego produktu by zdefiniować jego części składowe i zapewnić poprawną współpracę pomiędzy nimi. W tym czasie decyduje się między innymi jaki mikrokontroler zostanie użyty w projekcie. Nieprawidłowe oszacowanie obciążenia procesora lub ilości niezbędnej pamięci RAM/ROM/EEPROM może uniemożliwić implementację wszystkich wymagań klienta. Skutkiem tego ponosi się duże koszty związane z czasem potrzebnym do przeprojektowania układu i integracją nowego mikrokontrolera. Koszty te znacząco rosną jeżeli klient musi wstrzymać produkcję samochodu w oczekiwaniu na nowe, działające moduły.

Następnie analizowane są wymagania dotyczące oprogramowania (Software Requirements Analysis) i projektowana jest architektura aplikacji (Software Architectural Design).

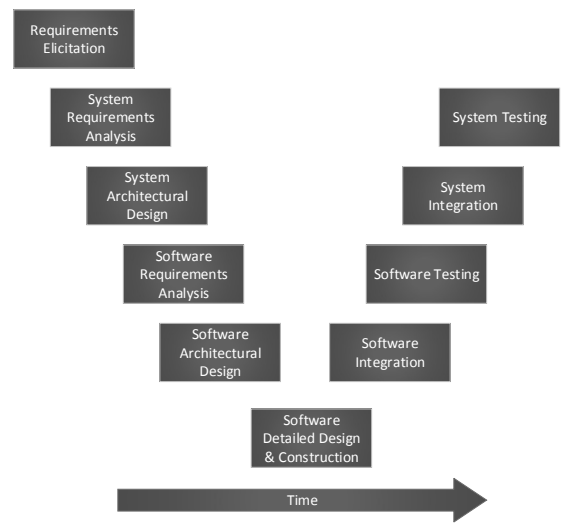
W kolejnych procesach następuje tworzenie oprogramowania (Software Detailed Design & Construction), jego integracja (Software Integration) oraz testowanie (Software Testing). Te trzy etapy są powtarzane do czasu uzyskania działającego kodu.

Ostatnie dwa kroki to integracja systemu (System Integration) i testowanie systemu (System Testing). Testowanie całości systemu daje odpowiedź na pytanie czy zaimplementowane funkcjonalności odpowiadają wymaganiom klienta i pozwalają na usunięcie błędów powstałych podczas integracji jego poszczególnych komponentów. Zazwyczaj raport z testów systemowych jest podstawą dla klienta do zaakceptowania dostarczonego komponentu.

Z uwagi na fakt, że funkcje nowoczesnych pojazdów są kontrolowane przez wiele współpracujących i komunikujących się ze sobą modułów wbudowanych, bez zdefiniowania procesu tworzenia oprogramowania utrzymanie jego jakości, a tym samym niezawodności całego samochodu, byłoby – przy tej skali złożoności – niemożliwe.

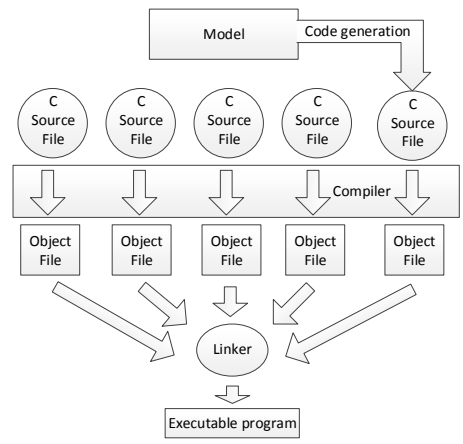
3. Modelowanie

W przemyśle motoryzacyjnym powszechnie stosowany jest standard AUTOSAR (Rys. 3) [4], który dostarcza wspólną infrastrukturę oprogramowania dla systemów wbudowanych stosowanych w pojazdach. AUTOSAR umożliwia przenoszenie aplikacji bez względu na zastosowany system operacyjny czy mikrokontroler. Taka standaryzacja gwarantuje, że dostarczane przez producenta samochodów modele funkcjonalne, zgodne z AUTOSAR-em mogą być przenoszalne między różnymi projektami.



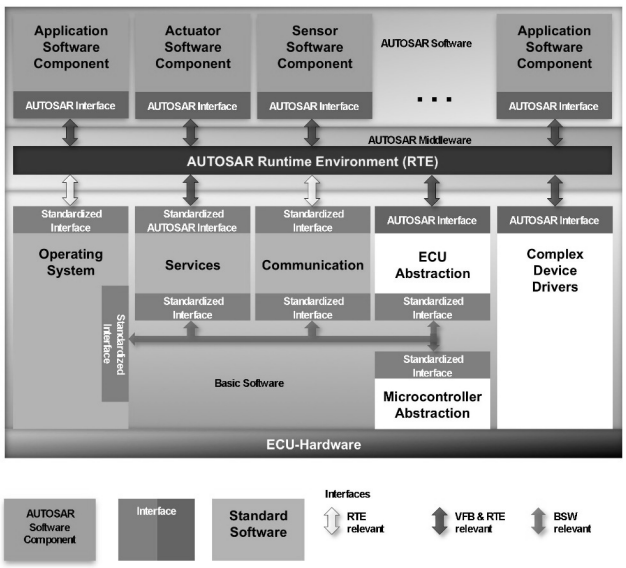
Rys. 1. Schemat rozwoju projektu softwarowego według modelu V
Fig. 1. V-model – software development process

Pojedyncze komponenty oprogramowania są modelami. Generowany z nich kod jest kompilowany wraz z całym projektem (rys. 2).



Rys. 2. Schemat wytwarzania oprogramowania z użyciem generacji kodu
Fig. 2. Diagram of software development using code generation

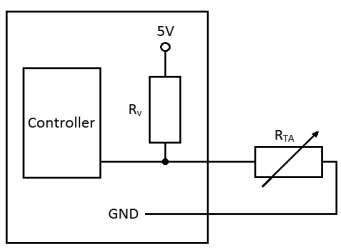
Popularnym środowiskiem do modelowania jest program Simulink [10] wraz z generatorem kodu (np. Embedded Coder lub TargetLink). Rozwiązanie to stosowane jest m.in. w firmie Delphi[1]. W TargetLink-u istnieje możliwość automatycznego generowania interfejsów AUTOSAR-owych na podstawie zaimportowanych informacji z narzędzia, w którym tworzona jest architektura systemu. Dzięki temu inżynier może skupić się przede wszystkim na realizacji właściwej funkcjonalności modułu.



Rys. 3. Architektura systemu softwarowego w oparciu o AUTOSAR. Modele uruchamiane są w warstwie aplikacji: Application Software Component. Źródło rysunku [4]
Fig. 3. Software architecture based on AUTOSAR. Models are run in application layer [4]

Korzyścią stosowania standardu AUTOSAR jest fakt, że interfejsy komunikacyjne zdefiniowane są w narzędziu do tworzenia architektury. Dzięki możliwości importowania informacji o interfejsach programista nie musi poświęcać czasu na ich implementację. Na podstawie tych danych można nie tylko odpowiednio skonfigurować bloki wejść i wyjść, ale także wygenerować całe otoczenie systemu służące do komunikacji.

Przystępując do tworzenia nowego modelu powinno się zwrócić szczególną uwagę na to, aby tworzony model nie był bardziej skomplikowany niż to absolutnie potrzebne. Dzięki temu funkcjonalność będzie czytelna a wykorzystanie zasobów sprzętowych zostanie ograniczone do minimum. Modelowi należy nadać hierarchiczną strukturę oddzielając od siebie poszczególne komponenty, które można umieścić w bibliotece. Takie zbieranie zagadnień w podsystemy ułatwia późniejszą analizę modelu. Inżynier oprogramowania może wybrać wygodne dla siebie metody implementacji poszczególnych części systemu wykorzystując np. diagramy bloków, maszyny stanów oraz tabele prawdy.

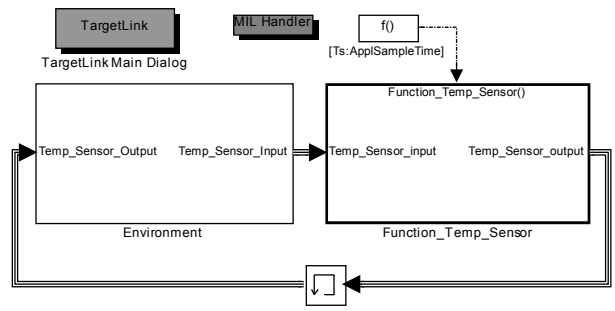


Rys. 4. Schemat ideowy czujnika pomiaru temperatury R_{TA}
Fig. 4. Temperature sensor schematic

[1] Delphi – jest wiodącym światowym dostawcą rozwiązań elektronicznych i technologii systemowych dla branży motoryzacyjnej. Koncern posiada na świecie ponad 100 zakładów produkcyjnych i 33 Centra Techniczne, zatrudniając ponad 100 tysięcy pracowników.

Rezystancyjny czujnik temperatury podłączony jest do mikrokontrolera w sposób przedstawiony na rysunku nr 4.

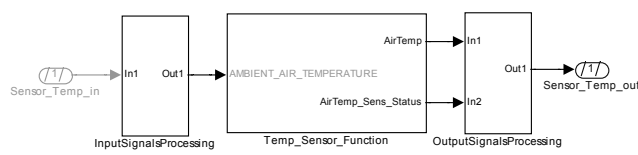
Proces przetwarzania wejściowej wartości napięcia z czujnika rozpoczyna się w warstwie Basic Software (rys. 3). Na tym etapie wartość odczytana z przetwornika ADC jest normalizowana do 5V i obliczana jest wartość rezystancji. Następnie informacja zostaje przekazana przez standardowe interfejsy do warstwy RTE (AUTOSAR Runtime Environment), która odpowiada za komunikację pomiędzy modułami systemu. W tym przypadku docelowym miejscem jest moduł generowany z modelu Temp_Sensor (rys. 5). Funkcją modelu jest zamiana rezystancji na temperaturę, analiza mierzonych wartości w celu wykrycia błędów oraz przesyłanie informacji na magistralę CAN.



Rys. 5. Model obsługujący sensor temperatury – wygląd podstawowy
Fig. 5. Model of temperature sensor functionality - basic view

Model składa się z części symulującej otoczenie oraz właściwego sterownika, w którym zawarta jest cała funkcjonalność modułu i warstwa odpowiedzialna za komunikację zgodną ze standardem AUTOSAR (rys. 6).

Wejściem do modelu jest 16-bitowy sygnał AMBIENT_AIR_TEMPERATURE przenoszący obliczoną wartość rezystancji z Basic Software. Opis sygnałów wyjściowych znajduje się w tabeli 1.



Rys. 6. Blok zawierający funkcjonalność otoczony interfejsami AUTOSAR
Fig. 6. Subsystem containing functionality surrounded by AUTOSAR interfaces

Gdy mierzona temperatura wyjdzie poza zakres, sygnał AirTemp przyjmuje wartość krańcową -40°C lub 85°C a po 2 sekundach (czas detekcji błędu) informacja o przekroczeniu mierzalnego zakresu jest przesyłana na magistralę CAN za pośrednictwem sygnału statusu („LOW" – poniżej zakresu lub „HIGH" - powyżej zakresu).

W przypadku wykrycia zwarcia do masy albo rozłączenia czujnika sygnał AirTemp po 2 sekundach przyjmuje wartość SNA=87,5 (ang. Signal Not Available), a sygnał statusu komunikuje rodzaj błędu. Detekcja błędów polega na obserwacji zarówno wartości temperatury jak i jej gradientu. Maksymalny dopuszczalny gradient w kierunku dodatnim i ujemnym wynosi 20°C/s. Przekroczenie tego progu jest informacją o możliwości wystąpienia przerwy w obwodzie lub zwarcia.

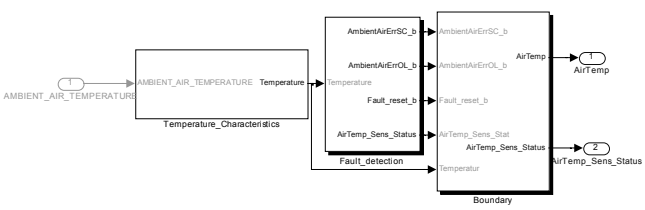
Logika modelu została zdekomponowana na część dotyczącą wyznaczenia temperatury na podstawie charakterystyki, część wykrywającą błędy oraz część ostatecznie obliczającą prawidłowe wartości sygnałów wyjściowych (rys. 7).

W celu ukazania działania modelu przeprowadzono symulacje Model-In-Loop oraz Software-In-Loop na podstawie przykładowego wektora wejściowego opisującego rezystancję czujnika. Z charakterystyki czujnika podawana jest wartości rezystancji dla krańcowych temperatur: 85°C = 320 Ω oraz -40°C = 100524 Ω.

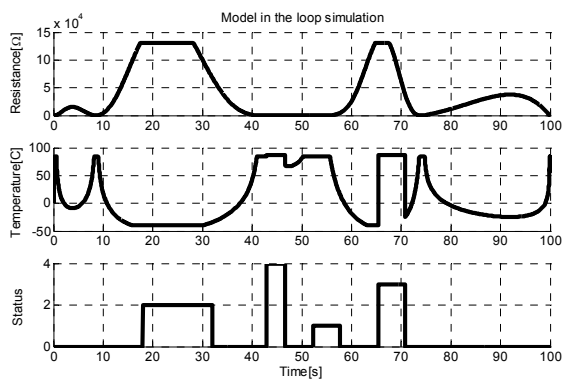
Scenariusz testowy miał na celu sprawdzenie poprawności wykrywania błędów przez model. W 16 sekundzie symulacji (rys. 8, 9) wartość rezystancji jest poniżej zakresu mierzalnych temperatur, bowiem nie jest spełniony warunek gradientowy. W związku z tym po 2 sekundach od tego zdarzenia ustawiany jest status ”LOW”. Dopiero w 64s testu, gdy przy podobnej sytuacji został przekroczony gradient 20°C/s model wykrył zdarzenie OPL (open load).

Tab. 1. Opis sygnałów AirTemp oraz AirTemp_Sens_Status
Tab. 1. Signal description of AirTemp and AirTemp_Sens_Status

AirTemp		**AirTemp_Sens_Status**		
Rozmiar	8-bit	**Nazwa**	**Wartość**	**Status**
Rozdzielczość	0.5°C	OK	0	Odczyt poprawny
Zakres	od -40°C do 85°C	HIGH	1	Temperatura powyżej zakresu
"SNA"	87.5	LOW	2	Temperatura poniżej zakresu
		OPL	3	Obwód pomiarowy rozłączony
		S2G	4	Zwarcie w obwodzie pomiarowym
		NDEF5	5	Nie używane
		NDEF6	6	Nie używane
		SNA	7	Sygnał niedostępny



Rys. 7. Podstawowy podział funkcjonalności
Fig. 7. Basic functionality decomposition



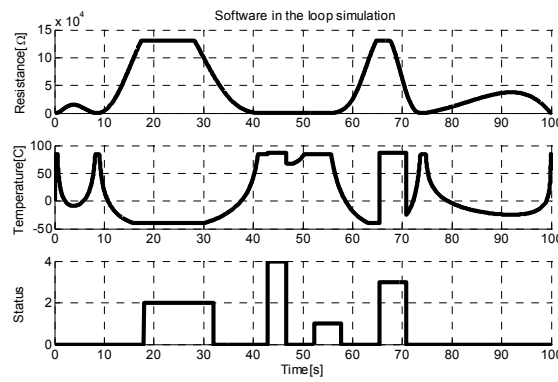
Rys. 8. Przebiegi rezystancji, temperatury oraz statusu w symulacji typu MIL. Rysunek ilustruje również różne rodzaje awarii jakie mogą wystąpić od lewej: LOW, S2G, HIGH, OPL
Fig. 8. Signal waveforms of resistance, temperature and measurement status during MIL simulation. Following errors can be detected by the model: LOW, S2G, HIGH, OPL.

Wyjścia systemu podczas symulacji SIL mają identyczne przebiegi jak podczas symulacji MIL co potwierdza poprawność działania wygenerowanego kodu.

4. Testowanie modelu

Narzędzia służące do modelowania pozwalają na tworzenie środowiska testowego, dzięki któremu możliwe jest dokładne przetestowanie systemu sterowania poprzez jego symulację. Istnieją trzy etapy weryfikacji, są to kolejno:

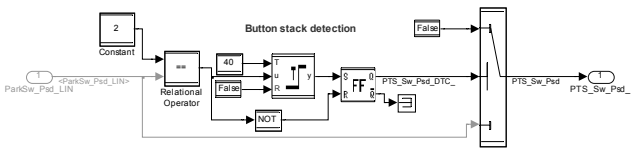
1. **Symulacja Model In the Loop** (MIL), gdzie testowany jest model a obliczenia realizowane są na liczbach zmiennoprzecinkowych (floating-point). Podczas testów MIL weryfikowana jest poprawność implementacji algorytmów, skalowania oraz sprawdzany jest błąd przepełnienia zmiennych (overflow).
2. **Symulacja Software In the Loop** (SIL) w przeciwieństwie do MIL symulacja testuje skompilowany kod uruchomiany w środowisku PC. Obliczenia realizowane są na zmiennych stałoprzecinkowych (fixed-point), co pozwala wykryć błędy kwantyzacji, saturacji i przepełnienia.
3. **Symulacja Processor In the Loop** (PIL) pozwala dokonać ostatecznej weryfikacji oprogramowania skompilowanego docelowym kompilatorem i uruchomionego na docelowym procesorze. Podczas działania symulacji PIL obserwuje się odpowiedzi wygenerowanego kodu na zadane wymuszenia, użycie stosu, pamięci RAM/ROM i wykorzystanie czasu procesora dla różnych ścieżek wykonania. Wyniki uzyskane tą metodą są najbardziej wiarygodne i pozwalają wychwycić znakomitą większość możliwych błędów implementacji.



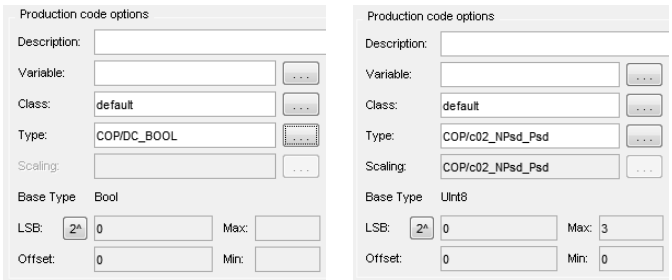
Rys. 9. Przebiegi rezystancji, temperatury oraz statusu w symulacji typu SIL. Rysunek ilustruje różne rodzaje awarii jakie model może zgłosić. Od lewej: LOW, S2G, HIGH, OPL.
Fig. 9. Signal waveforms of resistance, temperature and measurement status during SIL simulation. Following errors can be detected by the model: LOW, S2G, HIGH, OPL

Proces testowania usprawnia się wykorzystując zautomatyzowane środowiska testowe takie jak: TPT firmy PikeTec [5], Reactis firmy Reactive Systems lub Embedded Tester firmy BTC. Narzędzie te pozwalają na przygotowanie zaawansowanych raportów pokrycia wymagań, upraszczają proces tworzenia scenariuszy testowych oraz ich automatyczną generację. Raz utworzony scenariusz testowy może być stosowany do każdego typu symulacji: MIL, SIL, PIL podczas całego cyklu trwania projektu. Ogromną zaletą tworzenia oprogramowania z użyciem modeli jest możliwość testowania funkcjonalności jeszcze przed integracją z całością systemu. Pozwala to na szczegółowe testowanie algorytmów już we wczesnej fazie projektu.

Poniżej na prostym przykładzie przedstawiono błąd, jaki może wystąpić przy automatycznej generacji kodu programu. Założono, że w analizowanym fragmencie modelu obsługującego klawiaturę, występuje 2-bitowy sygnał ParkSw_Psd_LIN przenoszący informację o stanie przycisku: „Niewciśnięty”=0, „Nieużywany”=1, „Wciśnięty”=2, „Sygnał niedostępny”=3. Jego wartość jest przekazywana wprost do odpowiedniego interfejsu wyjściowego. Następnie do poprawnie działającego modelu dodana zostaje funkcjonalność (rys. 9), która ma za zadanie wykrywanie utknięcia przycisku w pozycji „Wciśniety”, która mogłaby oznaczać jego mechaniczne uszkodzenie. W takim przypadku po upływie określonego czasu model powinien ponownie rozpocząć sygnalizowanie pozycji „Niewciśnięty”. W przypadku, gdy jeden z bloków modelu jest niepoprawnie skonfigurowany (rys. 10) opisywana modyfikacja może spowodować pojawienie się poważnych skutków ubocznych tj. utratę przenoszonej informacji.



Rys. 9. Realizacja wykrywania utknięcia przycisku
Fig. 9. Implementation of button stuck detection



Rys. 10. Ustawienia bloku Switch, błędne przypisanie typu Bool (z lewej), prawidłowe ustawienie na typ 2-bitowy (z prawej)
Fig. 10. Switch block settings, incorrect assignment of type Bool (left), correct setting with 2-bit type (right)

Jak widać na rys. 10 (z lewej) w bloku Switch, typ wyjściowy został błędnie ustawiony na Bool. Podczas symulacji typu MIL wszystkie operacje były realizowane na zmiennych zmiennoprzecinkowych, przez co nie było możliwości zauważenia utraty danych. Test w trybie SIL, w którym następuje zawężenie typów, umożliwia wychwycenie błędu. Poprawne rozwiązanie zilustrowano na rys. 10 (z prawej).

Testy modelu typu Software In the Loop są w stanie wykryć większość możliwych nieprawidłowości pojawiających się w automatycznie generowanym kodzie. Dlatego bardzo ważne jest, aby od samego początku budowania modelu, tworzone były przypadki testowe pokrywające wszystkie realizowane wymagania. Po wprowadzeniu każdej nowej zmiany wysoce zalecane jest ich ponowne uruchomienie (w trybach MIL/SIL/PIL).

5. Wnioski

Obecnie nowoczesne samochody są wyposażane w coraz większą ilość czujników oraz systemów elektronicznych, z coraz to nowszymi funkcjonalnościami. Wiąże się to ze wzrostem skomplikowania implementowanych algorytmów ale także znacząco powiększa się rozmiar kodu źródłowego, który należy utrzymywać i modyfikować. W związku z tym coraz częściej korzysta się z automatycznej generacji kodu na podstawie modeli. Największą zaletą tego rozwiązania jest możliwość wielokrotnego wykorzystywania modelu realizującego daną funkcjonalność, niezależnie od używanego systemu operacyjnego czy mikrokontrolera. Jedynym warunkiem jest by tworzony system był zgodny ze standardem AUTOSAR. Nie bez znaczenia jest również znacząca ilość czasu oszczędzana w projekcie, która byłaby potrzebna każdorazowo na implementację funkcjonalności realizowanej przez modele.

Należy być jednak świadomym, że automatyczna generacja kodu nie wszędzie znajduje zastosowanie. Dużo prościej i szybciej jest bowiem zaimplementować ręcznie sterownik do np. konfiguracji Serial Peripheral Interface (SPI) w mikrokontrolerze niż go zamodelować ze wszystkimi rejestrami. Nie bez znaczenia jest również wysoki koszt zakupu niezbędnych licencji dla narzędzi oraz utrzymanie wsparcia technicznego.

6. Literatura

[1] Automotive SPICE, http://www.automotivespice.com/
[2] Scott P. Duncan: Making sense of ISO 15504 (and Spice), Quality/ Process Improvement Consultant SoftQual Consulting 6029 Flat Rock Rd #437, Columbus, GA 31907 (706) 565-9468.
[3] Murray C.: Automakers Opting for Model-Based Design,
[4] AUTomotive Open System Architecture, http://www.autosar.org
[5] TPT – Time Partition Testing, http://www.piketec.com/
[6] dSPACE, TargetLink Production Code Generation Guide, 2009.
[7] dSPACE, TargetLink Advanced Practices Guide, 2009.
[8] dSPACE, TargetLink Modeling Guidelines, http://www.dspace.com/shared/TargetLink/files?dl=ModelingGuidelines.pdf, 2010.
[9] MATLAB, manual: http://www.mathworks.nl/help/matlab/
[10] SIMULINK, manual: http://www.mathworks.nl/help/simulink/

otrzymano / received: 19.05.2014
przyjęto do druku / accepted: 01.07.2014 *artykuł recenzowany / revised paper*

INFORMACJE

Newsletter PAK

Wydawnictwo PAK wysyła drogą e-mailową do osób zainteresowanych Newsletter PAK, w którym są zamieszczane:
- spis treści aktualnego numeru miesięcznika PAK,
- kalendarz imprez branżowych,
- ważniejsze informacje o działalności Wydawnictwa PAK.

Newsletter jest wysyłany co miesiąc do osób, które w jakikolwiek sposób współpracują z Wydawnictwem PAK (autorzy prac opublikowanych w miesięczniku PAK, recenzenci, członkowie Rady Programowej, osoby które zgłosiły chęć otrzymywania Newslettera).

Celem inicjatywy jest umocnienie w środowisku pozycji miesięcznika PAK jako ważnego i aktualnego źródła informacji naukowo-technicznej.

Do newslettera można zapisać się za pośrednictwem:
- strony internetowej: www.pak.info.pl, po dodaniu swojego adresu mailowego do subskrypcji,
- adresu mailowego: wydawnictwo@pak.info.pl, wysyłając swoje zgłoszenie.

Otrzymywanie Newslettera nie powoduje żadnych zobowiązań ze strony adresatów. W każdej chwili można zrezygnować z otrzymywania Newslettera.

Tadeusz SKUBIS
Redaktor naczelny Wydawnictwa PAK