УДК 62-503.5
DOI: 10.25140/2411-5363-2021-1(23)-118-127

Олександр Похиленко, Павло Катін

ПАТЕРН «СТАН» ДЛЯ ВБУДОВАНИХ СИСТЕМ З МОЖЛИВІСТЮ ДИНАМІЧНОГО СТВОРЕННЯ СТАНІВ

У сучасних технічних системах (ТС), що є вбудованими системами, які поєднують декілька технологій програмування, виникає потреба додавання станів у ході роботи за командою від головного контролеру (ГК) без перезавантаження програмної частини допоміжного контролеру реального часу (ДКРЧ). У статті розглядається вирішення задачі стикування програмної частини ГК, що може мати елементи штучного інтелекту, і ДКРЧ для виконавчих механізмів. Запропоноване рішення задачі з застосуванням патерну Стан, що дозволяє додавати стани в кінцевий автомат без перезавантаження програмної частини ТС реального часу.

Ключові слова: *вбудована система; фінансово-економічна система, система реального часу, мікроконтролер, динамічне додавання станів до патерну Стан, кінцевий автомат.*

Рис.: 3. Табл.: 1. Бібл.: 9.

Актуальність теми дослідження. На теперішній час актуальними й поширеними є вбудовані системи (ВС) на основі складних програмних рішень, що поєднують у собі декілька технологій програмування. Наприклад, програмна частина банківського автомату (Automated teller machine) включає у себе програми управління механічними пристроями (прийом і видача грошей) і програми мережевих технологій (передача даних). До ВС зі складними програмними частинами можна віднести: фінансово-економічні системи, системи управління рухом наземних і повітряних об'єктів, медичні системи тощо.

Складну ВС умовно можна поділити на дві базові складові, одна з яких виконує роль головного контролера, що описано у [1]. Головний контролер (ГК) може вирішувати відносно складні задачі під управлінням операційної системи (ОС), а саме: забезпечувати роботи бази даних (БД), реалізовувати програмну складову штучного інтелекту, обробляти відео і аудіо інформацію, тощо.

Друга частина складної ВС опитує датчики і керує виконавчим механізмами у реальному часі. Будемо назвати цю частину допоміжним контролером реального часу (ДКРЧ) [1]. Апаратно його доцільно реалізовувати на базі мікроконтролера (МК) архітектури Advanced RISC Machine (ARM) Cortex-M [1; 2], доступні інші апаратні рішення. Програма даних МК може бути побудована у вигляді шаблона (патерна) Стан. Він реалізує роботу математичної моделі кінцевого автомату (КА). До найбільш поширених практичних рішень цього КА на теперішній час належать: нескінченні цикли опитування (Polled Loop Systems), програми на основі операційних систем реального часу (ОСРЧ) системи переривань мікроконтролера (Interrupt Driven), програми на базі простих ОСРЧ з підтримкою багатозадачності (Multi-tasking).

Потужність сучасних мінікомп'ютерів дає можливість реалізовувати функції ГК і ДКРЧ на борту одного мінікомп'ютера [3]. Апаратно він може бути реалізований на базі архітектури ARM, а саме Cortex-A. Навіть у цьому випадку програмна частина розподіляється на модулі, що забезпечують програмну складову ГК і ДКРЧ.

Оскільки сучасні вбудовані системи з ГК і ДКРЧ у багатьох випадках поєднують у себе декілька технологій програмування, їх стикування є актуальним питанням. Особливо це актуально, коли ГК виконує роль штучного інтелекту і може виникнути потреба додавання станів у програму ДКРЧ у ході роботи без перезавантаження складної програмної системи загалом.

Постановка проблеми. Як було зазначено вище, програмна частина ДКРЧ може бути реалізована у вигляді патерну Стан, що реалізує математичну модель кінцевого автомату. У стандартному варіанті розробки програми додавання станів у кінцевий автомат відбувається на етапі розробки програмного забезпечення [5-8] і після перепрограмування МК.

На теперішньому стані розвитку апаратної бази ГК і ДКРЧ і програмних технологій складних ВС може виникати потреба додавати у КА додаткові стани динамічно без перезавантаження.

Для вирішення цієї задачі введемо обмеження. Нехай апаратна частина ДКРЧ реалізована у вигляді мікроконтролера архітектури ARM Cortex-M. Для програмної частини ДКРЧ будемо використовувати Cortex Microcontroller Software Interface Standard (CMSIS) і відповідний стиль розробки.

Потрібно розробити програмне рішення патерну Стан для програмної частини ДКРЧ, що дозволяє додавати стани динамічно під час роботи без зупинки ВС.

Рішення цієї задачі дозволить забезпечити програмне стикування ГК і ДКРЧ і додавати нові стани у КА програмної частини ДКРЧ динамічно для забезпечення працездатності ВС і керованості системи загалом.

Аналіз останніх досліджень і публікацій. У [1] проведений огляд рішень у вигляді операційних систем реального часу, запропоновано рішення у вигляді кінцевого автомату, але при цьому не було проведено досліджень щодо стикування аналогів ДКРЧ з базовою системою управління. Тобто рішення розглядалося на рівні автономної програми кінцевого автомату.

Автори роботи [2] вирішують завдання побудови програмної частини для архітектури Cortex-M для масштабованих рішень. Звичайно процес масштабування реалізується на етапі розробки ПЗ, а не під час роботи.

У [3] проведений огляд рішень на основі високопродуктивних систем, при цьому програмна система не упорядковувалася у вигляді кінцевого автомату, реалізація відбувалася на рівні стандартного фреймворку.

У роботі [4] подано реалізацію програми на основі шаблону Стан та описано дослідження щодо збільшення швидкості обмеження, але це зроблено на основі модернізації відомих рішень. У [5] описано результати дослідження програмної реалізації шаблону Стан в ООП для малопотужних МК в узагальненому вигляді, де неможливо додати новий стан під час виконання програми.

У [6-8] наведено результати дослідження і розробки програмної реалізації кінцевого автомату в різних модифікаціях програмного рішення, проте додати новий стан під час виконання роботи також неможливо.

Отже, за результатом останніх досліджень і публікацій можна зробити загальний висновок щодо відсутності готового рішення задачі додавання станів у КА на основі МК архітектури Cortex-M динамічно.

Виділення недосліджених частин загальної проблеми. Узагальнюючи вищесказане можна зробити висновок, що дана стаття присвячена вирішенню нової перспективної задачі вдосконалення програмної частини сучасних складних ВС, що мають головний контролер і ДКРЧ. Програмна частина ДКРЧ реалізована у вигляді патерну Стан. Задача полягає у розробці програмного прототипу, що дозволить додавати нові стани у КА динамічно без перезавантаження системи у цілому.

Мета статті. Як було вищезазначено, складна ВС має головний контролер реального часу, що може вирішувати відносно складні задачі, а саме: обробку і накопичення даних у бази даних (БД), реалізацію штучного інтелекту, обробку великого обсягу інформації (відео, аудіо), тощо. Для керованості системи загалом програмна частина ГК і ДКРЧ має бути поєднана у одну цілу систему. У результаті роботи програмної частини ГК може виникнути потреба створення нового стану для програмної частини ДКРЧ. Саме вона управляє виконавчими механізмами технічної системи або здійснює опитування датчиків у режимі реального часу.

На теперішній час для додавання нового стану потрібно переробляти програму контролера, записувати її у ДКРЧ і перезавантажувати систему.

Метою статті є розробка й опис програмного рішення, що дає можливість додати новий стан у програмну частину ДКРЧ, під час роботи, динамічно, за командою, що надходить з ГК.

Для вирішення поставлених завдань необхідно наступне:

- розробити та протестувати прототип програми ДКРЧ у вигляді шаблону КА для мікроконтролера архітектури Cortex-M на рівні системного програмування у вигляді програмного рішення шаблону Стан, що дозволяє додавати стани динамчно;

- протестувати це рішення на практичному мікроконтролері Cortex-M.

Виклад основного матеріалу. У таблиці наведено результат розробки шаблона КА для мікроконтролера архітектури Cortex-M на рівні системного програмування у вигляді програмного рішення шаблону Стан. На відміну від відомих рішень [1-8], він дозволяє додавати стани динамічно та формувати переходи між станами під час виконання програми. Розроблене рішення можна адаптувати для реальної складної ВС. Це потрібно реалізувати так, щоб додавання нового стану у КА допоміжного контролеру реального часу виконувалося за сигналом з ГК. Такий сигнал може генерувати ГК складної ВС у разі потреби створення нового стану для ДКРЧ. Це може бути зроблено, наприклад, у ході роботи програмної частини ГК, що реалізує функції штучного інтелекту. Розроблений код являє собою прототип програмного рішення для складних ВС, що містять головний контролері і ДКРЧ.

При розробці і тестуванні рішення використано типовий 32-розрядний МК STM32F103, тактова частота якого дорівнює 72 МГц, напруга живлення 3,3 В, температура в приміщенні під час тестування 27—30°С. Результати дослідження можна розповсюдити на широкий спектр МК архітектури ARM Cortex-M.

Результатом решения вищеописаної проблеми є прототип програми, що показана в таблиці і реалізує ООП варіант програмного рішення КА для ДКРЧ складної ВС. У даному коді показана демонстрація динамічного створення нових станів. Наприклад, у рядку

SignalToggleState *s1 = new SignalToggleState...

показано динамічне створення одного стану s1, подальше його додавання і практичне використання у наступних рядках коду до першого програмного циклу.

Припустимо, що у ході роботи програми виникла потреба додати нові стани динамчіно. Це демонструє наступний приклад динамічного створення нового стану s2, встановлення переходів між новим станом та попереднім, а також подальше використання цих станів у програмі (таблиця).

Таблиця. Вміст файлу main.cpp

main.cpp

```
#include "stm32fsm.h"
AbstractState *currentState;
int main(void) {
  SignalToggleState *s1 = new SignalToggleState(GPIO_B, GPIO_Pin_6,
    GPIO_Mode_Out_PP, GPIO_Speed_50MHz,
    GPIO_Pin_9 | GPIO_Pin_10, GPIO_Mode_IPU, false); // Create state 1

  currentState = s1; // Set initial state
  currentState->HandleEntry();
  for (uint32_t i = 0; i < 0xFF; i++) currentState->HandleDo(); // Do something
```

```
  SignalToggleState *s2 = new SignalToggleState(GPIO_B, GPIO_Pin_7,
    GPIO_Mode_Out_PP, GPIO_Speed_50MHz,
    GPIO_Pin_11 | GPIO_Pin_12, GPIO_Mode_IPU, false); // Create state 2

  s1->PushTransition(0x19, s2); // State 1 to state 2 on signal from pin 9 of GPIOB
  s2->PushTransition(0x1C, s1); // State 2 to state 1 on signal from pin 12 of GPIOB
  for (uint32_t i = 0; i < 0xFF; i++) currentState->HandleDo(); // Do something

  SignalToggleState *s3 = new SignalToggleState(GPIO_B, GPIO_Pin_8,
    GPIO_Mode_Out_PP, GPIO_Speed_50MHz,
    GPIO_Pin_13 | GPIO_Pin_14, GPIO_Mode_IPU, false); // Create state 3
  s1->PushTransition(0x1A, s3); // State 1 to state 3 on signal from pin 10 of GPIOB
  s2->PushTransition(0x1B, s3); // State 2 to state 3 on signal from pin 11 of GPIOB
  s3->PushTransition(0x1D, s1); // State 3 to state 1 on signal from pin 13 of GPIOB
  s3->PushTransition(0x1E, s2); // State 3 to state 2 on signal from pin 14 of GPIOB

  while(1) currentState->HandleDo();
}
```

Далі в коді показаний аналогічний процес динамічного створення ще одного стану s3 і подальше його додавання до КА та практичне використання. Таким чином, у таблиці показаний прототип коду, що дозволяє динамічно створити нові стани з переходами. На рис.1 це показано у вигляді графа. Граф на рис. 1 містить три стани, що були створені поетапно, покроково.

На першому етапі був створений та працював Стан 1 у програмному рішенні для МК і показаний на рис. 1.

На другому етапі був доданий Стан 2, були сформовані відповідні переходи між станами, що показані на графі рис. 1. Кінцевий автомат програмного рішення для МК працював з двома станами.

На третьому етапі доданий Стан 3 з відповідними переходами. Таким чином, вихідний код для МК, що зображений у таблиці, реалізує граф кінцевого автомату, що показаний на рис. 1. Звичайно, що вихідний код таблиці є демонстрацією технології динамічного створення станів у КА і має суттєві спрощення.

Але він може бути застосований у як прототип під час розробки реальної програмної частини ДКРЧ для складної ВС. У цьому випадку до програми ДКРЧ можуть додаватися нові стани динамічно при отриманні команд від ГК. У такий спосіб забезпечуєтеся стикування програмної частини КГ складної ВС і допоміжного контролера реального часу. Зв'язок головного контролера і ДКРЧ можна забезпечити послідовними інтерфейсами МК, у самому простому прикладі це може бути USART.

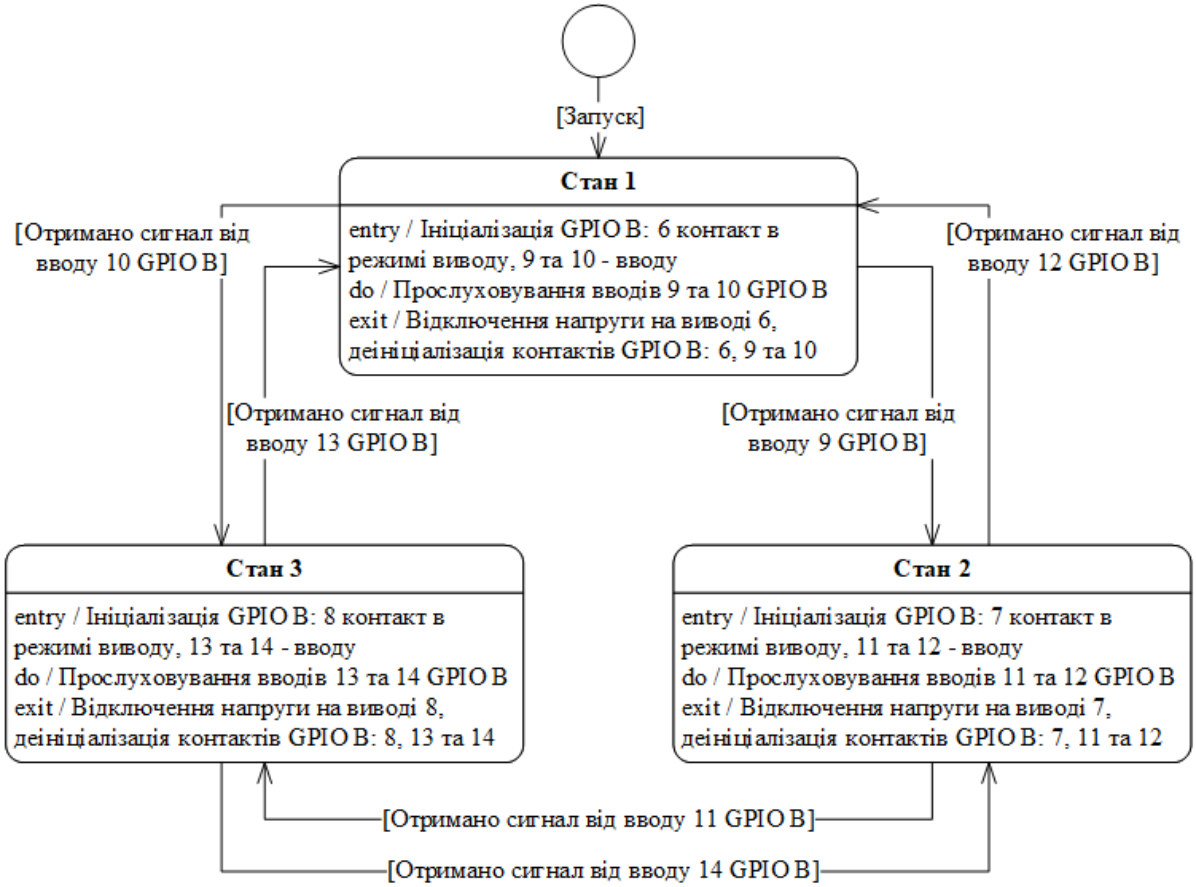


Рис. 1. Діаграма станів для тестового прикладу

Репозиторій з вихідним кодом програми був розміщений на GitHub [9]. Розглянемо програмну реалізацію динамічного додавання нових станів у програмну частину ДКРЧ. Для цього стани КА були реалізовані у вигляді класів, а переходи у вигляді структур мови програмування C++. Схема програмного рішення показана на рис. 2 у вигляді діаграми класів, що відповідає розробленому рішенню.

Для більшої зручності клас стану AbstractState був зроблений абстрактним, але з реалізацією деяких методів, щоб уникнути дублювання коду. Цей клас є базовим для станів КА у програмній частині ДКРЧ.

Структура TransitionInfo відображає переходи між станами. Якщо формалізувати програмне рішення у вигляді графу переходів, то ця структура буде відповідати ребрам орієнтованого графа. Стани будуть відповідати вузлам графа.

Під час динамічного додавання неможливо визначити кількість переходів і вузлів. Для реалізації цього програмного механізму, кожна структура переходу містить у собі покажчик на наступну структуру. Структура TransitionInfo містить в собі інформацію про сигнал, що змушує виконати цей перехід, та наступний стан (покажчик на клас стану), до якого необхідно перейти.

Клас AbstractState при отриманні сигналу в методі MakeTransition виконує перевірку структур переходів, перебираючи всі структури. Також клас AbstractState містить методи HandleEntry, HandleDo та HandleExit, що можуть виконувати певні корисні дії при переході в цей стан. Корисний програмний функціонал передбачений також при знаходженні в цьому стані та при покиданні цього стану. Корисний програмний функціонал передбачений у вигляді методів, що реалізують частини entry, do, exit відповідно до рис. 1.

Оскільки розглядається динамічне додавання станів та переходів, клас AbstractState має методи для роботи з переходами: PushTransition, MergeTransitions та PopTransition – для додавання одного переходу, додавання декількох переходів та видалення останнього переходу.

Здебільшого абстрактного класу AbstractState недостатньо, щоб динамічно додавати нові стани. Користуючись класом AbstractState можна додавати нові стани шляхом створення нових нащадків цього класу. При поточної реалізації на основі класу AbstractState вже є працездатним механізм створення, додавання та видалення переходів між станами КА програмної частини ДКРЧ.

На першому кроці необхідно створити екземпляр структури TransitionInfo і передати необхідні значення через конструктор.

На другому кроці потрібно викликати відповідний метод, щоб додати новий перехід.

Оскільки стани КА програмній частини ДКРЧ створюються динамічно за командою з ГК, потрібно також, щоб стани КА виконували корисні дії для складної ВС. Це потребує додавання до нащадків абстрактного класу AbstractState програмну реалізацію корисних дій у відповідному віртуальному методі. Такими корисними діями можуть бути: видача команд на виконавчі механізми, подача напруги на порти вводу-виводу загального призначення, подача сигналів на послідові інтерфейси МК, прийом сигналів із датчиків, тощо. Для спрощення виконання таких дій були створені додаткові функції [9]. Наприклад, функція деініціалізації, що дозволяє скинути налаштування портів вводу-виводу МК, відключити тактування порту, якщо він більше не використовується тощо. Інші додаткові функції дозволяють керувати портами, не прив’язуючись до деталей реалізації, шляхом використання енуменатору (enum) GPIOs.

На рис. 2 показані два нащадки класу AbstractState: ToggleState та SignalToggleState, що дозволяють динамічно створювати нові стани. Клас ToggleState призначений для керування виводами портів. Клас ToggleState в методі HandleEntry виконує налаштування виводів та керує ними, а в методі HandleExit скидає налаштуваня обраних виводів.

Клас SignalToggleState розширює можливості класу ToggleState. Крім керування виводами, він також може прослуховувати вводи і подавати сигнали для зміни стану. Крім відповідних аналогічних налаштувань в методах HandleEntry та HandleExit дається можливість налаштовуються портів вводу-виводу загального призначення МК для введення і виведення інформації до програмної частини МК.

У методі HandleDo виконується опитування портів вводу-виводу загального призначення та перевірка їхніх станів разом із генерацією відповідного сигналу при зміні стану. Отже, створюючи екземпляри класів SignalToggleState та додаючи переходи між цими станами, можна динамічно збільшувати функціональність програмної реалізації КА допоміжного контролера реального часу складної ВС.

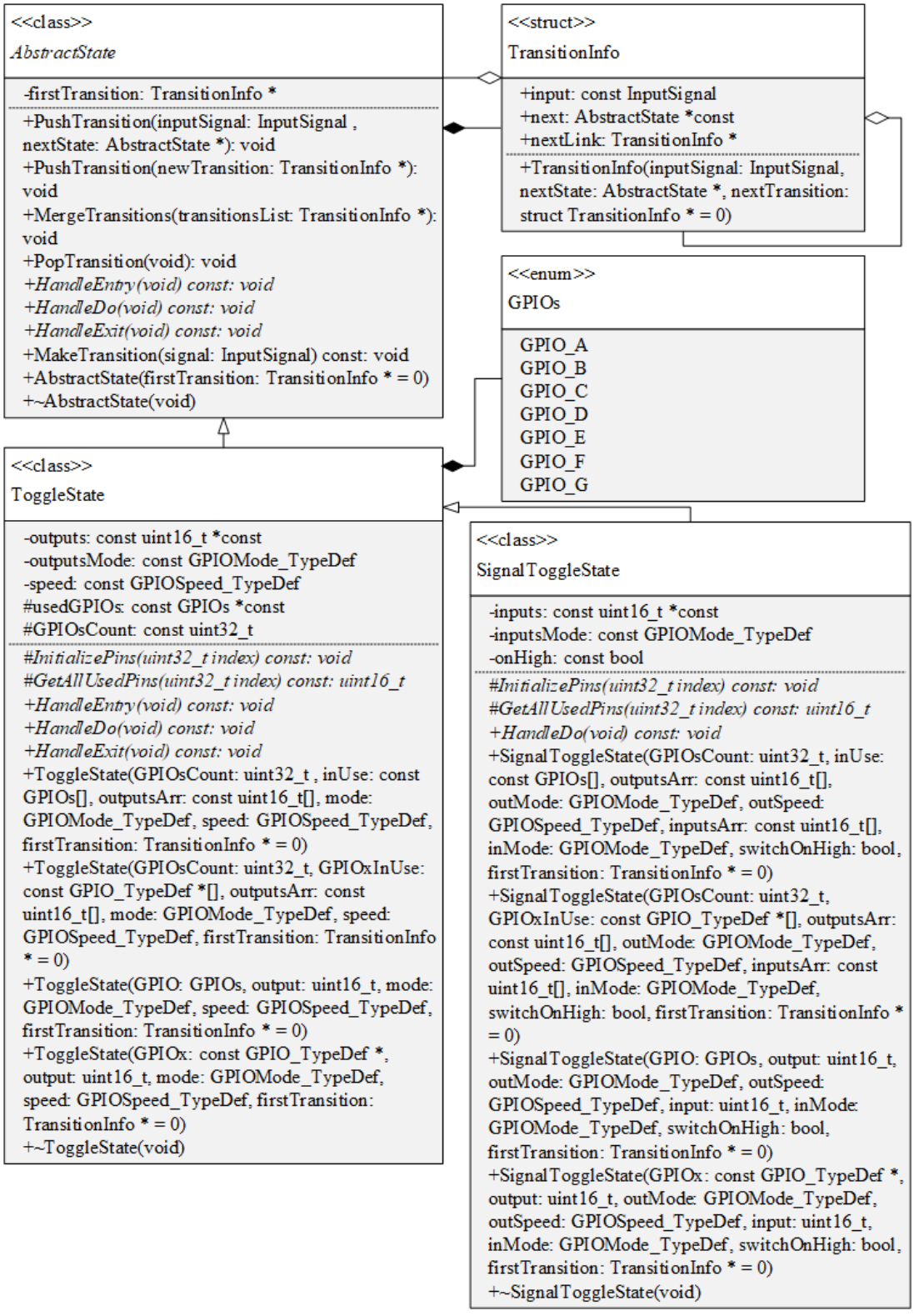
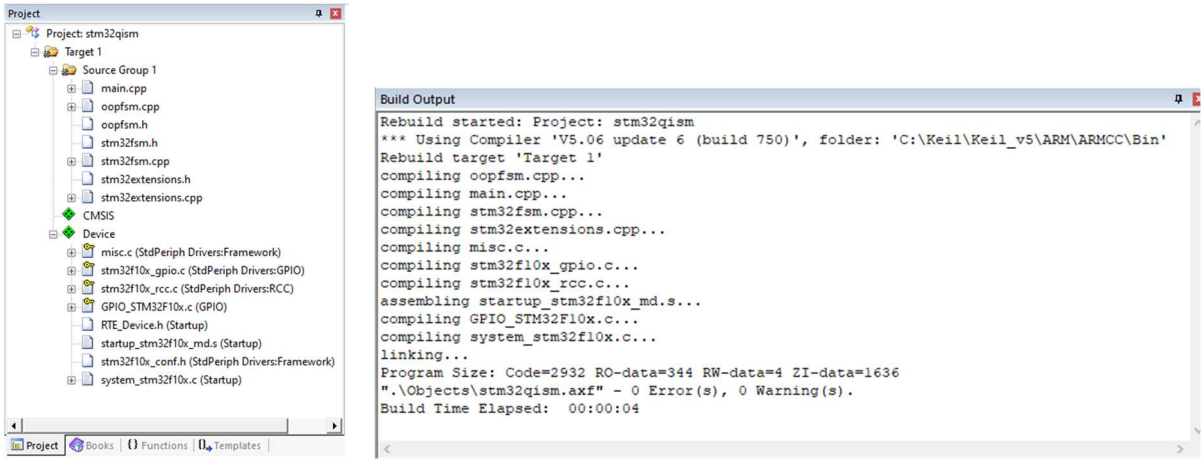


Рис. 2. Діаграма класів розробленого рішення

Працездатність розробленого рішення була перевірена за допомогою тестового коду таблиці. Рішення було успішно зібране без помилок, що підтверджує рис. 3, та перевірене на мікроконтролері Cortex-M, а саме STM32F103.



а *б*

Рис. 3. Середовище розробки Keil µVision:
а – структура проєкту; б – вивід при зборці

Висновки. Сучасні вбудовані системи, у багатьох випадках, поєднують у програмній частини декілька технологій програмування. Складну ВС умовно можна розподілити на дві базові складові: головний контролер (його програмну і апаратну частину) і ДКРЧ (його програмну і апаратну частину).

Актуальною проблемою є забезпечення програмного стикування керувальної програми ГК і програмної частини ДКРЧ. Особливо це викликає ускладнення, коли ГК може ініціалізувати створення нових станів у програмній частині ДКРЧ, що стандартно реалізована у вигляді патерна Стан і може бути формалізована у вигляді математичної моделі кінцевого автомату. Тобто може виникнути потреба додавання нових станів у ході роботи програмної частини ДКРЧ динамічно.

У статті розроблений прототип програми, що дозволяє вирішити цю задачу. Він може бути застосований у якості прототипу під час розробки реальної програмної частини ДКРЧ для складної ВС. Отримане програмне рішення дає можливість додавати нові стани динамічно при отриманні команд від ГК. Зв'язок головного контролера і ДКРЧ можна забезпечити через послідовні інтерфейси МК, у самому простому прикладі це може бути USART.

Під час роботи над статтею розроблений та протестований прототип програми ДКРЧ у вигляді патерну Стан для мікроконтролера архітектури Cortex-M на рівні системного програмування, що дозволяє додавати стани під час виконання програми.

Проведені дослідження і визначена працездатність програми. Побудовані діаграма станів і діаграма класів отриманого рішення. Прототип вихідного коду розташований на ресурсі [9].

Список використаних джерел

1. Chmelov V. O., Katin P. Y., Shemaev V. M. Development of typical "State" software patterns for Cortex-M microcontrollers in real time. *Eastern-European Journal of Enterprise Technologies*. 2020. Vol. 3(9 (105)). Pp. 29-38.

2. Chen Z., Chen J., Zhou S. Embedded electronic scale measuring system based on STM32 single chip microcomputer. *Proceedings – 2019 Chinese Automation Congress* (November 22-24). Hangzhou, China, 2019. Pp. 3062-3065.

3. Zhu W., Wang Z., Zhang Z. Renovation of Automation System Based on Industrial Internet of Things: A Case Study of a Sewage Treatment Plant. *Sensors*. 2020. Vol. 20(8). P. 2175.

4. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. *Addison-Wesley*, 1994.

5. Katin P. Y. Development of variant of software architecture implementation for low-power general purpose microcontrollers by finite state machines. *EUREKA: Physics and Engineering*. 2017. Vol. 3. Pp. 49–55.

6. Dietrich C., Hoffmann M. and Lohmann D. Back to the Roots: Implementing the RTOS as a Specialized State Machine. *OSPERT 2015* (Sweden, July 7, 2015). Lund, Sweden, 2015. Pp. 7-12.

7. Beynon W. On the structure of free finite state machines. *Theoretical Computer Science*. 1980. Vol. 11. Pp. 167-180.

8. Adamczyk P. The Anthology of the Finite State Machine Design Patterns, 2013. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.838&rep=rep1&type=pdf.

9. Репозиторій з вихідним кодом на GitHub. URL: https://github.com/AlexanderPokhilenko/STM32QISM.

References

1. Chmelov, V. O., Katin, P. Y., and Shemaev, V.M. (2020). Development of typical "State" software patterns for Cortex-M microcontrollers in real time. *Eastern-European Journal of Enterprise Technologies, 3*(9(105)), pp. 29-38.

2. Chen, Z., Chen, J. and Zhou, S. (2019). Embedded electronic scale measuring system based on STM32 single chip microcomputer. CA*C 2019, Proceedings – 2019 Chinese Automation Congress, Hangzhou* (pp. 3062-3065).

3. Zhu, W., Wang, Z. and Zhang, Z. (2020). Renovation of Automation System Based on Industrial Internet of Things: A Case Study of a Sewage Treatment Plant. *Sensors*, *20*(8), p. 2175.

4. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley. Boston, Massachusetts.

5. Katin, P.Y. (2017). Development of variant of software architecture implementation for low-power general purpose microcontrollers by finite state machines. *EUREKA: Physics and Engineering*, *3*, pp. 49–55.

6. Dietrich, C., Hoffmann, M. and Lohmann, D. (2015). Back to the Roots: Implementing the RTOS as a Specialized State Machine. *Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications, Proceedings of OSPERT 2015* (pp. 7-12).

7. Beynon, W. (1980). On the structure of free finite state machines. *Theoretical Computer Science*, *11*, pp. 167-180.

8. Adamczyk, P. (2003). The Anthology of the Finite State Machine Design Patterns. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.838&rep=rep1&type=pdf.

9. Source code repository on GitHub. https://github.com/AlexanderPokhilenko/STM32QISM.

UDC 62-503.5

Pavlo Katin, Oleksandr Pokhylenko

STATE PATTERN FOR EMBEDDED SYSTEMS WITH THE POSSIBILITY OF DYNAMIC CREATION OF STATES

Nowadays, embedded systems are technical systems that combine several programming technologies in the software part. A complex technical system of this type can be divided into two basic components, first – the main controller (MC). The second part is designed to control the mechanisms and is an auxiliary real-time controller (ARTC).

The urgent task is to provide software connection of the MC software and the software part of the ARTC. This is especially true when the MC plays the role of artificial intelligence and when it may be necessary to add new states during program execution without restarting the ARTC software.

The software part of the ARTC can be implemented in the form of a State pattern, which implements a mathematical model of a finite state machine. To connect the software part of the MC and ARTC and dynamically add new states to the ARTC program, the following tasks must be solved:

- to develop a software solution of the pattern State for ARTC, which will allow to add states dynamically while working without stopping and rebooting on command from the MC;

- test this solution on a practical microcontroller.

According to the results of recent researches and publications, we can draw a general conclusion about the lack of a ready-made solution to the problem of adding states to the finite state machine without rebooting.

This article is devoted to solving a new promising problem of connecting the software part of the MC with elements of artificial intelligence and ARTC for actuators. There is an unexplored part of the overall problem that needs to be solved in the context of the dynamic addition of new states to the finite state machine of the ARTC program.

To solve the tasks, it is necessary to develop and test a prototype of the ARTC program in the form of a State pattern for the microcontroller architecture Cortex-M at the level of system programming, which allows you to add states during program execution dynamically.

A prototype of the ARTC program was developed in the form of a finite state machine for the Cortex-M architecture microcontroller at the level of system programming in the form of a software solution of the State pattern, which allows to add states during program execution. The source code was developed and the formal model of the software decision in the form of the graph was constructed. The research was carried out and the work of the program was demonstrated.

The proposed software solution allows software connection of the MC software and the software part of the ARTC in complex embedded systems. This is necessary when there may be a need to add new states dynamically during execution of software program without restarting the ARTC. Such a need may arise, for example, when the MC plays the role of artificial intelligence and there is a need to add new states during execution dynamically.

Keywords: *embedded system; financial and economic system, real-time system, microcontroller, dynamic addition of states to the State pattern, finite state machine.*

Fig.: 3. Table: 1. References: 9.

Похиленко Олександр Андрійович – студент кафедри автоматики та управління в технічних системах, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» (просп. Перемоги, 37, м. Київ, 03056, Україна).
Oleksandr Pokhylenko – student of Department of automation and Control in Technical Systems, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute" (37 Peremohy Av., 03056 Kyiv, Ukraine).
E-mail: pokhilenko.alex@gmail.com
ORCID: https://orcid.org/0000-0002-1562-2051

Катін Павло Юрійович – кандидат технічних наук, доцент кафедри автоматики та управління в технічних системах, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» (просп. Перемоги, 37, м. Київ, 03056, Україна).
Pavlo Katin — PhD in Technical Sciences, Associate Professor of Department of automation and Control in Technical Systems, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute" (37 Peremohy Av., 03056 Kyiv, Ukraine).
E-mail: p.katin@kpi.ua
ORCID: https://orcid.org/0000-0002-2542-9976

Похиленко О., Катін П. Патерн «стан» для вбудованих систем з можливістю динамічного створення станів. *Технічні науки та технології.* 2021. № 1(23). С. 118-127.