Gem #4: Контроль позиции в пространстве используя PWM/ШИМ для радиоуправляемых сервосистем

Автор: Michael Gonzalez Harbour, Universidad de Cantabria, Spain

Краткое содержание: Gem Ada #4 – Пример программы на Ada-2005, который демонстрирует практическое применение событий с контролем по времени.

Давайте начнем…

Широтно-импульсная модуляция (PWM/ШИМ) – Общий механизм кодирования аналоговой информации на цифровой линии. ШИМ обычно используется в управляющих сигналах. Он реализуется из генерации импульсов на цифровой линии. Ширина импульса, то есть временной интервал, в течение которого сигнал является высоким (или низким), пропорционален аналоговому значению, которое используется в качестве управляющего сигнала.

Сервоприводы, используемые в системах с радиоуправлением, таких как модельные автомобили или самолеты, обычно основаны на сигнале ШИМ, в котором генерируется периодический импульс с периодом от 10 до 30 мс. Ширина этого импульса изменяется от 1 до 2 миллисекунд и представляет собой угловое положение сервомеханизма. Импульс 1 мс представляет собой одну крайность углового положения, 2 мс представляет другую крайность, а 1,5 мс представляет собой центральное положение. Четыре таких импульса могут выдаваться за каждый период для управления четырьмя различными сервомеханизмами.

Время импульсов должно быть очень точным. Если у нас есть диапазон вращения сервопривода между -120 и 120 градусами, и нам нужна точность, скажем, 1 градус, это означает, что точность в времени импульса должна составлять 1 мс x 1/240 = 4,1 мсек. Эта точность может быть достигнута только в том случае, если мы используем систему времени выполнения Ada, которая работает поверх операционной системы реального времени (RTOS) для голого компьютера, такого как ОС MaRTE (http://marte.unican.es)

В нашей системе у нас есть два сервомотора, управляющих двумя осями и использующих отдельные цифровые сигналы. Мы хотим контролировать положение этих сервоприводов с компьютера, на котором запущена программа Ada 2005, и генерировать импульсы через цифровой выходной порт.

Поскольку это пример приложения, предназначенный для отображения функций реального времени в Ada 2005 и для целей тестирования, мы будем реализовывать его, используя реализацию GNAT поверх Linux, используя систему времени выполнения MaRTE. Это означает, что мы не можем получить временное разрешение лучше, чем предлагаемое Linux (по умолчанию около 10 мс). По этой причине мы изменим временные требования системы, умножив все временные интервалы на 100. Это подразумевает период 2000 мс и импульсы длительностью от 100 до 200 мс, с точностью до 24%. Это было бы неприемлемым для реальной реализации (для чего необходима ОСРВ), но это нормально для наших целей тестирования

Цифровой порт вывода

Цифровой выходной порт реализован в пакете Digital_Output, который имеет операцию Set, чтобы установить нужный порт на цифровое значение Low или High и операцию закрытия устройства. Открытие устройства происходит автоматически.

```
package Digital_Output is

   -- Identifier of a digital output line
   type Channel_Id is range 1..2;

   -- Digital output value
   type Digital_State is (Low, High);

   ---------------------------------------------------------------------
   -- Close
   ---------------------------------------------------------------------
   --
   -- This procedure must be called once the package is no longer to
   -- be used, to close the output device
   -- Эта процедура должна вызываться после того, как пакет больше не будет
   -- использоваться, чтобы закрыть устройство вывода

   procedure Close;


   ---------------------------------------------------------------------
   -- Set
   ---------------------------------------------------------------------
   --
   -- This procedure sets a particular digital channel to the given
   -- digital value
   -- Эта процедура устанавливает конкретный цифровой канал для данного
   -- цифрового значения

   procedure Set (Channel : Channel_Id; To : Digital_State);

   ---------------------------------------------------------------------
   -- Get
   ---------------------------------------------------------------------
   --
   -- This function returns the digital value of a particular channel
   -- Эта функция возвращает цифровое значение определенного канала

   function Get (Channel : Channel_Id) return Digital_State;

end Digital_Output;
```

Реализация этого пакета представляет собой смоделированную среду, в которой изменения значений цифровых выходов записываются вместе с временем, в которое они происходят, относительно времени начала эксперимента. Когда устройство закрыто, записанные значения записываются в текстовый файл, содержимое которого затем может отображаться с помощью программы графического представления, например gnuplot.

Архитектура реализации

Реализация механизма сервоуправления осуществляется в пакете PWM. Этот пакет содержит защищенный объект Servo_Control, который сохраняет позицию, необходимую для сервоприводов, и выполняет операции, необходимые для изменения положения, и генерирует синхронизацию импульсов, необходимых для управления сервомашинками с помощью PWM.

Время генерации импульсов контролируется с помощью событий времени Ada 2005. Каждый сервопривод имеет свое собственное событие синхронизации, которое запрограммировано на запуск в момент начала импульса, а затем, когда он должен остановиться. Положение сервопривода разделяется между обработчиком события

синхронизации и задачей пользователя или задачами, которые записывают нужные позиции. Совместно использование корректно за счет использования защищенного объекта Servo_Control.

Спецификация пакета PWM

```
with System;
with Ada.Real_Time;
with Ada.Real_Time.Timing_Events;

package PWM is

   package Timing_Events renames Ada.Real_Time.Timing_Events;

   ----------------------------------------------------------------
   -- Configuration constants
   ----------------------------------------------------------------

   Angular_Range       : constant:= 240.0;         --degrees
   Min_Pulse_Width     : constant Duration:=0.100; --seconds
   Max_Pulse_Width     : constant Duration:=0.200; --seconds
   Central_Pulse_Width : constant Duration:=
     (Max_Pulse_Width+Min_Pulse_Width)/2.0;
   Pulse_Period        : constant Duration:=2.000; --seconds

   ----------------------------------------------------------------
   -- Identifier of each of the two servo mechanisms
   -- Идентификатор каждого из двух серво механизмов
   ----------------------------------------------------------------

   type Servo_Id is (Steering, Throttle);

   ----------------------------------------------------------------
   -- Angular position of a servo mechanism (in degrees)
   -- Угловое положение сервомеханизма (в градусах)
   ----------------------------------------------------------------

   type Position is digits 5
     range -Angular_Range/2.0 .. Angular_Range/2.0;

   ----------------------------------------------------------------
   -- Extension of the Timing_Event tagged type to store an
   -- additional attribute with the servo Id
   -- Расширение типа тегов Timing_Event для хранения дополнительного
   -- атрибута с помощью идентификатора сервопривода
   ----------------------------------------------------------------

   type Servo_Timing_Event is new Timing_Events.Timing_Event with record
      Id : Servo_Id;
   end record;

   ----------------------------------------------------------------
   -- Values of this type represent the positions of several servos
   -- Значения этого типа представляют позиции нескольких сервоприводов
   ----------------------------------------------------------------

   type Servo_Position is array (Servo_Id) of Position;

   ----------------------------------------------------------------
   -- Values of this type represent time values associated with
   -- the servos
   -- Значения этого типа представляют значения времени, связанные с
   -- сервомоторами
```

```
   ------------------------------------------------------------

   type Servo_Time is array (Servo_Id) of Ada.Real_Time.Time;

   ------------------------------------------------------------
   -- Protected object used to store the desired poritions for the
   -- servos
   -- Защищенный объект, используемый для хранения желаемых позиций для
   -- сервоприводов
   ------------------------------------------------------------

   protected Servo_Control is

      ------------------------------------------------------------
      -- Set the position of a given servo to the given value
      -- Установите положение данного сервопривода на заданное значение
      ------------------------------------------------------------

      procedure Set_Position(Id : Servo_Id; Pos : Position);

      ------------------------------------------------------------
      -- Initiate the control actions on a servo
      -- Инициировать действия управления сервоприводом
      ------------------------------------------------------------

      procedure Init (Id : Servo_Id);

      pragma Priority(System.Interrupt_Priority'Last);

   private

      Commanded_Position : Servo_Position:=(0.0,0.0);
      Pulse_Start_Time : Servo_Time;

      ------------------------------------------------------------
      -- Start the pulse on one of the servo outputs
      -- Начать импульс на одном из сервоусилителей
      ------------------------------------------------------------

      procedure Start_Pulse (Event : in out Timing_Events.Timing_Event);

      ------------------------------------------------------------
      -- Finish the pulse on one of the servo outputs
      -- Завершите импульс на одном из выходов сервопривода
      ------------------------------------------------------------

      procedure End_Pulse (Event : in out Timing_Events.Timing_Event);

   end Servo_Control;

end PWM;
```

Мы видим, что мы определили длительность импульса и период с настраиваемыми константами. Мы добавили тегированный тип Timing_Event, чтобы добавить информацию, необходимую для обработчиков. Эта информация является идентификатором сервопривода, который контролируется. Защищенный объект Servo_Control имеет два открытых действия. Один для установки желаемой позиции, а другой - для запуска управления ШИМ на нужном сервоприводе. Потолок защищенного объекта устанавливается на уровень приоритета прерывания, так как события синхронизации выполняются с таким приоритетом.

Приватная часть защищенного объекта содержит общее состояние, состоящее из желаемых позиций и времени, в которое запускается импульс каждого сервопривода, который необходим для запуска события периодической синхронизации. Он также содержит операции, которые будут использоваться в качестве обработчиков для событий синхронизации. Один для запуска импульса на данном сервоприводе, а другой – для его остановки. Конкретный идентификатор сервопривода хранится в расширенной части объекта Timing_Event (для этой цели должен быть передан объект Servo_Timed_Event).

В качестве альтернативы реализации мы могли бы использовать одно событие синхронизации для генерации импульсов для обоих сервомеханизмов, но в этом случае нам пришлось бы реализовать арифметику, необходимую для определения того, какой сервопривод имел более раннее время запуска или остановки. Используя одно временное событие для каждого сервомотора, мы упрощаем код приложения.

Детальная реализация

Далее показано тело пакета PWM которое иллюстрирует использование событий синхронизации.

```
with Digital_Output;

package body PWM is

   use type Ada.Real_Time.Time;

   -- Mapping between servo Id and digital channel
   Digital_Channel : constant array(Servo_Id) of
     Digital_Output.Channel_Id := (Steering => 1, Throttle => 2);

   -- Timed events, one per servo
   Timer : array(Servo_Id) of Servo_Timing_Event;

   -------------------
   -- Servo_Control --
   -------------------


   protected body Servo_Control is

      ------------------
      -- Set_Position --
      ------------------


      procedure Set_Position (Id : Servo_Id; Pos : Position) is
      begin
         Commanded_Position(Id):=Pos;
      end Set_Position;

      ----------
      -- Init --
      ----------


      procedure Init (Id : Servo_Id) is
      begin
         Pulse_Start_Time(Id):=Ada.Real_Time.Clock;
         Timer(Id).Id:=Id;
         Start_Pulse (Timing_Events.Timing_Event(Timer(Id)));
      end Init;
```

```
      -----------------
      -- Start_Pulse --
      -----------------


      procedure Start_Pulse (Event : in out Timing_Events.Timing_Event) is
         Pulse_Stop_Time : Ada.Real_Time.Time;
         Id : Servo_Id:=
           Servo_Timing_Event(Timing_Events.Timing_Event'Class(Event)).Id;
      begin
         -- Start the pulse on the corresponding channel
         Digital_Output.Set(Digital_Channel(Id),Digital_Output.High);

         -- Calculate the stop time
         Pulse_Stop_Time:=Pulse_Start_Time(Id)+
           Ada.Real_Time.To_Time_Span
           (Duration(Float(Commanded_Position(Id)/Angular_Range)*
                     Float(Max_Pulse_Width-Min_Pulse_Width))+
            Central_Pulse_Width);

         -- Program the timed event for the next stop time
         Timing_Events.Set_Handler
           (Event,Pulse_Stop_Time,End_Pulse'Access);
      end Start_Pulse;

      ---------------
      -- End_Pulse --
      ---------------


      procedure End_Pulse (Event : in out Timing_Events.Timing_Event) is
         Id : Servo_Id:=
           Servo_Timing_Event(Timing_Events.Timing_Event'Class(Event)).Id;
      begin
         -- Stop the pulse on the corresponding channel
         Digital_Output.Set(Digital_Channel(Id),Digital_Output.Low);

         -- Program the timed event for the next start time
         Pulse_Start_Time(Id):=Pulse_Start_Time(Id)+
           Ada.Real_Time.To_Time_Span(Pulse_Period);
         Timing_Events.Set_Handler
           (Event,Pulse_Start_Time(Id),Start_Pulse'Access);
      end End_Pulse;

   end Servo_Control;

end PWM;
```

Тело пакета содержит массив констант, который задает сопоставление между идентификатором сервоуправления и связанным с ним цифровым выходным каналом. Он также содержит события синхронизации, по одному для каждого сервопривода, типа Servo_Timing_Event, который содержит дополнительный идентификатор сервопривода, который должен быть установлен с помощью операции Init.

Операции защищенного объекта Servo_Control имеют следующее поведение:

- Set_Position: изменяет желаемое положение указанного сервопривода.
- Init: устанавливает значение Id в событии синхронизации, сохраняет время запуска импульса и вызывает операцию Start_Pulse для события синхронизации, связанного с указанным сервоприводом.

- Start_Pulse: эта операция запускает импульс на цифровом выходе, затем вычисляет время останова для импульса в соответствии с желаемой позицией соответствующего сервопривода и перепрограммирует событие синхронизации для истечения времени остановки, устанавливая операцию End_Pulse в качестве обработчика ,

- End_Pulse: эта операция останавливает импульс на цифровом выходе, затем вычисляет новое время начала импульса, добавляя период к предыдущему, и перепрограммирует событие синхронизации для истечения в начале следующего периода, устанавливая операцию Start_Pulse как обработчик.

Идея состоит в том, что после того, как операция Init вызывается приложением один раз, начинается первый импульс, а затем время события запрограммировано и последовательно вызывает End_Pulse, затем Start_Pulse, затем End_Pulse и т. д. Без дальнейшего вмешательства приложения.

Тестовая программа

Мы написали небольшую тестовую программу, которая устанавливает положение сервоприводов и инициирует управление ШИМ для обоих сервоприводов, затем ждет некоторое время и устанавливает позиции на разные значения, а затем ждет некоторое дополнительное время до окончания эксперимента. Результаты эксперимента хранятся в текстовом файле pmw_data.txt, который затем отображается.

Далее следует текст тестовой программы.

```
with Pwm;
with Digital_Output;

procedure Test_Pwm is
   use type Pwm.Position;
begin
   -- Set initial position and start the PWM control
   Pwm.Servo_Control.Set_Position(Pwm.Steering,100.0);
   Pwm.Servo_Control.Set_Position(Pwm.Throttle,-100.0);
   Pwm.Servo_Control.Init(Pwm.Steering);
   Pwm.Servo_Control.Init(Pwm.Throttle);

   -- Change position after some time
   delay 3.0;
   Pwm.Servo_Control.Set_Position(Pwm.Steering,0.0);
   Pwm.Servo_Control.Set_Position(Pwm.Throttle,100.0);

   -- Let the experiment run for some time and then close it
   delay 20.0;
   Digital_Output.Close;
end Test_Pwm;
```

Программа построена путем компиляции основной программы test_pwm.adb с системой времени выполнения MaRTE, которая реализует синхронизированные события

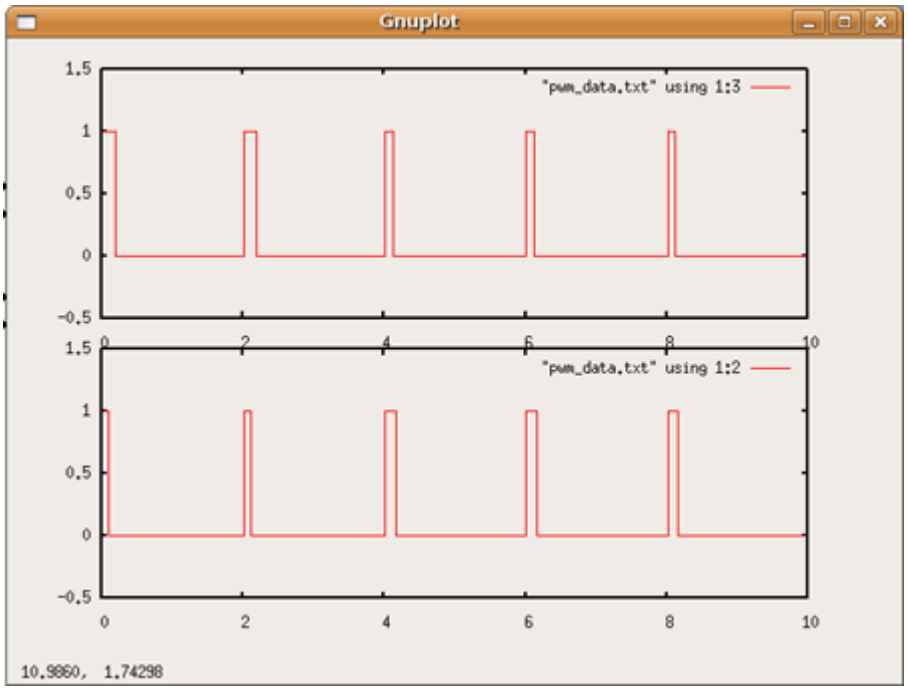
```
gnatmake --RTS=rts-marte test_pwm.adb
```

Результаты

Выполнение программы создает текстовый файл с тремя столбцами, представляющий время с момента запуска программы, состояние цифрового выхода 1 и состояние цифрового выхода 2.

На рисунке ниже показаны результаты выполнения. Мы видим, что для первых двух импульсов ширина первого сервопривода (сверху) больше, чем для второго, что соответствует начальным желаемым позициям. Через 3 секунды позиции меняются, а затем ширина импульса второго сервомашины меньше.

Результаты показывают ожидаемое поведение для событий синхронизации, эффективно реализуя требуемый сервоуправление ШИМ.



Обсуждение…

2 ответов на "Gem # 4: PWM Position Control для радиоуправляемых сервомоторов"

1. 4 июня 2007 года в 20:58

   Ань Во сказал:

   Благодарим вас за публикацию GEM. Это один из моих лучших фаворитов. При нажатии ссылки приложение / x-ada-source показывает ошибку. Кроме того, апостроф, такой как «Приоритет прагмы (System.Interrupt_Priority« Last »),« отображается как » на этой странице.

2. 5 июня 2007 года в 11:35

   Франко Гасперони сказал:

   Спасибо, что указали проблему связи, которая была исправлена.