МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Информационная безопасность систем и технологий»

Отчет

о выполнении лабораторной работы No2

«Таймеры и обработка прерываний микроконтроллеров STM32»

Дисциплина: ПМК

Группа 21ПТ2

Выполнил студент: Юдин Н.М.

Количество баллов:

Дата сдачи:

Принял: Хворостухин С.П.

2023

1 Цель работы: Ознакомиться с программными средствами работы с таймерами и прерываниями микроконтроллеров STM32

2 Задание:

2.1 Создать проект в среде Keil uVision5 для микроконтроллера STM32F103RB. Настроить режим отладки для проекта.

2.2 Разработать сигнальную функцию, которая обеспечивает изменение периода входного сигнала для значений: 2 мс, 4 мс, 6 мс, 12 мс, 20 мс. Смену периода выполнять через фиксированное количество импульсов - 20 и более.

2.3 Разработать программу для микроконтроллера STM32F103RB, реализующую захват входного сигнала, настройку выходного сигнала согласно параметрам входного: длительность сигнала широтно-импульсной модуляции равна периоду входного сигнала, выдачу выходного сигнала широтно-импульсной модуляции с постоянной частотой. Период таймера широтно-импульсной модуляции задать равным 32 мс. Параметры программы выбираются согласно варианту.

2.4 Провести симуляцию разработанной программы.

3 Результаты работы:

3.1 Был создан проект в среде Keil uVision5:

– Был выбран пункт меню «Project - New uVision Project, была создана папка проекта, задано имя проекта;

– Для использования в проекте была выбрана модель микроконтроллера: STM32F103RB;

– Были выбраны программные компоненты: CMSIS/Core, Device/Startup, Device/StdPeriph Drivers/Framework, Device/StdPeriph Drivers/GPIO, Device/StdPeriph Drivers/RCC, Device/StdPeriph Drivers/TIM и было завершено создание проекта.

Была выполнена настройка режима отладки для проекта. Для этого в папке проекта был создан файл MAP.ini со следующим содержанием:

2

MAP 0x40000000, 0x47FFFFF READ WRITE;

Далее была открыта вкладка «Option for Target...» затем была открыта вкладка «Debug», был включен переключатель «UseSimulator». В поле «DialogDLL» было записано DARMSTM.DLL, в поле «Parameter» было записано: - pSTM32F103RB. Был установлен путь к файлу MAP.ini в поле «InitializationFile». Далее была нажата кнопка OK для завершения настройки режима отладки проекта.

3.2 Была разработана сигнальная функция, которая обеспечивает изменение периода входного сигнала для значений: 2 мс, 4 мс, 6 мс, 12 мс, 20 мс. Смена периода выполняется каждые 20 импульсов. Содержание файла сценария приведено в приложении А. Ниже, на рисунке 1, приведен вариант, согласно которому выполнялась работа.

Номер варианта	Таймер захвата сигнала			Таймер ШИМ		
	Номер таймера	Номер канала	Значение отсчета	Номер таймера	Номер канала	Значение отсчета
3	TIM4	CH3	5 мкс	TIM3	CH1	20 мкс

Рисунок 1 - Вариант заданий лабораторной работы

3.3 Была разработана программа для микроконтроллера STM32F103RB, реализующая захват входного сигнала, настройку выходного сигнала согласно параметрам входного: длительность сигнала широтно- импульсной модуляции равна периоду входного сигнала, выдача выходного сигнала широтно-импульсной модуляции с постоянной частотой. Период таймера широтно-импульсной модуляции задан равным 32 мс. Параметры программы выбраны согласно варианту из предыдущего пункта. Реализация программы приведена в приложении Б. Ниже приведено описание алгоритма работы разработанной программы.

Для инициализации портов ввода-вывода была реализована функция

initPort(), не принимающая параметры и имеющая тип void, что означает, что она не возвращает значение. Функция инициализирует ноги PB8 и PA6. Реализация функции initPort() показана ниже на рисунке 2.

```
void initPort() {
	GPIO_InitTypeDef port;
	GPIO_StructInit(&port);

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
	port.GPIO_Mode = GPIO_Mode_IN_FLOATING;
	port.GPIO_Pin = GPIO_Pin_8;
	port.GPIO_Speed = GPIO_Speed_2MHz;
	GPIO_Init(GPIOB, &port);

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
	port.GPIO_Mode = GPIO_Mode_AF_PP;
	port.GPIO_Pin = GPIO_Pin_6;
	port.GPIO_Speed = GPIO_Speed_2MHz;
	GPIO_Init(GPIOA, &port);
}
```

Рисунок 2 - Реализация функции initPort()

Для инициализации таймера захвата реализована функция initCaptureTimer() не принимающая параметры и имеющая тип void, что означает, что она не возвращает значение. Данная функция настраивает третий канал таймера под номером 4 и разрешает прерывания. Реализация функции показана ниже на рисунке 3.

```
void initCaptureTimer() {
	//TIM4, CHANNEL 3
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

	TIM_TimeBaseInitTypeDef timer;
	TIM_TimeBaseStructInit(&timer);
	timer.TIM_Prescaler = 540 - 1;
	timer.TIM_Period = 0xFF4F;
	TIM_TimeBaseInit(TIM4, &timer);

	TIM_ICInitTypeDef ic;
	ic.TIM_Channel = TIM_Channel_3;
	ic.TIM_ICPolarity = TIM_ICPolarity_Rising;
	ic.TIM_ICSelection = TIM_ICSelection_DirectTI;
	ic.TIM_ICPrescaler = TIM_ICPSC_DIV1;
	ic.TIM_ICFilter = 0;
	TIM_ICInit(TIM4, &ic);

	TIM_ITConfig(TIM4, TIM_IT_CC3, ENABLE);
	TIM_Cmd(TIM4, ENABLE);
	NVIC_EnableIRQ(TIM4_IRQn);
}
```

Рисунок 3 - Реализация функции initCaptureTimer()

Для данной функции был произведен расчет значения предделителя по формуле, приведенной ниже на рисунке 4.

$$Prescaler = F_{TIM} \times T_{CNT}$$

где:

– F_{TIM} - частота тактового сигнала, Гц;

– T_{CNT} - время одного отсчета таймера, с.

Рисунок 4 - Формула для расчета значения предделителя

Согласно данной формуле было получено значение 540, но значение предделителя, записываемое в регистр таймера, должно быть на 1 меньше

рассчитанного. Поэтому в регистр записывается 539.

Для обработки прерывания была реализована функция TIM4_IRQHandler() не принимающая параметры и имеющая тип void, что означает, что она не возвращает значение. В данной функции происходит запись захваченного значения таймера. Реализация функции TIM4_IRQHandler() показана ниже на рисунке 5.

```
void TIM4_IRQHandler() {
    if(TIM_GetITStatus(TIM4, TIM_IT_CC3) != RESET) {
        TIM_ClearITPendingBit(TIM4, TIM_IT_CC3);
        capture1 = capture2;
        capture2 = TIM_GetCapture3(TIM4);

        if(!first_capture)
            ready_capture = 1;

        first_capture = 0;
    }
}
```

Рисунок 5 - Реализация функции TIM4_IRQHandler()

Для установки состояния выходного порта была реализована функция diffTime принимающая 2 параметра типа uint16_t: a и b; и имеющая тип uint16_t. Данная функция вычисляет разницу во времени между значениями a и b. Реализация функции difftime(uint16_t a, uint16_t b) показана ниже на рисунке 6.

```
uint16_t diffTime(uint16_t a, uint16_t b) {
    return ( a >>  b ) ? ( a - b ) : (UINT16_MAX - b + a);
}
```

Рисунок 6 - Реализация функции difftime(uint16_t a, uint16_t b)

Для инициализации таймера ШИМ реализована функция initPWMTimer не принимающая параметры и имеющая тип void, что означает, что она не возвращает значение. Данная функция настраивает первый канал таймера под номером 3 и разрешает прерывания. Реализация функции показана ниже на

рисунке 7.

```
void initPWMTimer() {
	//TIM3, CHANNEL 1
	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

	TIM_TimeBaseInitTypeDef PWM_timer;
	TIM_TimeBaseStructInit(&PWM_timer);
	// 21600 = 108000000 * 0.00020
	PWM_timer.TIM_Prescaler = 21600 - 1;
	PWM_timer.TIM_Period = 200;
	TIM_TimeBaseInit(TIM3, &PWM_timer);

	TIM_OCInitTypeDef oc;
	TIM_OCStructInit(&oc);
	oc.TIM_OCMode = TIM_OCMode_PWM1;
	oc.TIM_OutputState= TIM_OutputState_Enable;
	oc.TIM_Pulse = 0xFF;
	TIM_OC1Init(TIM3, &oc);

	TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);
	TIM_Cmd(TIM3, ENABLE);
	NVIC_EnableIRQ(TIM3_IRQn);
}
```

Рисунок 7 - Реализация функции initPWMTimer()

Значение предделителя для данного таймера определяется по формуле, которая используется в функции initCaptureTimer(). Было получено значение

21600, в регистр записывается 21600 — 1.

Для обработки прерывания была реализована функция TIM3_IRQHandler() не принимающая параметры и имеющая тип void, что

означает, что она не возвращает значение. В данной функции происходит

7

установка длительности выходного сигнала ШИМ. Реализация функции TIM3_IRQHandler() показана ниже на рисунке 8.

```
void TIM3_IRQHandler() {
      if(TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET) {
            TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
            period_capture = period_capture / 38;
            TIM_SetCompare1(TIM3, period_capture);
      }
}
```

Рисунок 8 - Реализация функции TIM3_IRQHandler()

В теле программы происходит вызов функций инициализации портов, инициализации таймера захвата, инициализация таймера ШИМ. Затем идет бесконечный цикл while, в котором, если есть значения для обработки, происходит закрытие прерываний, вычисление длительности сигнала и открытие прерываний. Тело программы показано ниже на рисунке 9.

```
int main() {
      initPort();
      initCaptureTimer();
      initPWMTimer();
      while(1) {
            if(ready_capture) {
                  NVIC_DisableIRQ(TIM4_IRQn);
                  ready_capture = 0;
                  period_capture = diffTime(capture2, capture1);
                  NVIC_EnableIRQ(TIM4_IRQn);
            }
      }
      return 0;
}
```

Рисунок 9 - Тело программы

Таким образом, была завершена разработка программы.

3.4 Была проведена симуляция разработанной программы. Результаты симуляции программы приведены в приложении В.
Для симуляции программы был включен режим отладки, в консоли отладчика был подключен файл сценария и была вызвана его сигнальная функция затем в окно анализатора были добавлены порты PB8 и PA6, используемые по варианту. Так же в окно анализатора были добавлены переменные period_capture, capture1 и capture2.

4 Вывод: Было произведено ознакомление с программными средствами работы с таймерами и прерываниями микроконтроллеров STM32.

Приложение А

Содержание файла сценария

(обязательное)

```
signal void formationSignal() {
     float delay;

     int i;
     while(1) {
     delay = 0.001;
     for(i = 0; i < 20; i++) {

          PORTB |= 0x0100;
          swatch(delay);
          PORTB &= ~0x0100;
          swatch(delay);

     }
     delay = 0.002;
     for(i = 0; i < 20; i++) {

          PORTB |= 0x0100;
          swatch(delay);
          PORTB &= ~0x0100;
          swatch(delay);

     }
     delay = 0.003;
     for(i = 0; i < 20; i++) {

          PORTB |= 0x0100;
          swatch(delay);
          PORTB &= ~0x0100;
          swatch(delay);

     }
     delay = 0.006;

 for(i = 0; i < 20; i++) {

         PORTB |= 0x0100;

         swatch(delay);
         PORTB &= ~0x0100;
         swatch(delay);

    }
    delay = 0.010;
    for(i = 0; i < 20; i++) {
```

```
        PORTB |= 0x0100;
        swatch(delay);
        PORTB &= ~0x0100;
        swatch(delay);

}

} }
```

Приложение Б

(обязательное)

Реализация программы

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#include "stm32f10x_tim.h"

uint16_t capture1 = 0, capture2 = 0;
uint16_t first_capture = 1, ready_capture = 0;
uint16_t period_capture = 0;

void initPort() {
	GPIO_InitTypeDef port;
	GPIO_StructInit(&port);

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
	port.GPIO_Mode = GPIO_Mode_IN_FLOATING;
	port.GPIO_Pin = GPIO_Pin_8;
	port.GPIO_Speed = GPIO_Speed_2MHz;

	GPIO_Init(GPIOB, &port);

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
	port.GPIO_Mode = GPIO_Mode_AF_PP;
	port.GPIO_Pin = GPIO_Pin_6;
	port.GPIO_Speed = GPIO_Speed_2MHz;

	GPIO_Init(GPIOA, &port);
}

void initCaptureTimer() {
	//TIM4, CHANNEL 3

	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

	TIM_TimeBaseInitTypeDef timer;
	TIM_TimeBaseStructInit(&timer);
	timer.TIM_Prescaler = 540 - 1;
	timer.TIM_Period = 0xFF4F;
	TIM_TimeBaseInit(TIM4, &timer);

	TIM_ICInitTypeDef ic;
	ic.TIM_Channel = TIM_Channel_3;
	ic.TIM_ICPolarity = TIM_ICPolarity_Rising;
	ic.TIM_ICSelection = TIM_ICSelection_DirectTI;
	ic.TIM_ICPrescaler = TIM_ICPSC_DIV1;
	ic.TIM_ICFilter = 0;
```

```
	TIM_ICInit(TIM4, &ic);

	TIM_ITConfig(TIM4, TIM_IT_CC3, ENABLE);
	TIM_Cmd(TIM4, ENABLE);
	NVIC_EnableIRQ(TIM4_IRQn);

}

void TIM4_IRQHandler() {
	if(TIM_GetITStatus(TIM4, TIM_IT_CC3) != RESET) {

		TIM_ClearITPendingBit(TIM4, TIM_IT_CC3);
		capture1 = capture2;
		capture2 = TIM_GetCapture3(TIM4);

		if(!first_capture)
			ready_capture = 1;

		first_capture = 0;

} }

uint16_t diffTime(uint16_t a, uint16_t b) {
	return ( a >>  b ) ? ( a - b ) : (UINT16_MAX - b + a);

}

void initPWMTimer() {
	//TIM3, CHANNEL 1

	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

	TIM_TimeBaseInitTypeDef PWM_timer;
	TIM_TimeBaseStructInit(&PWM_timer);
	// 21600 = 108000000 * 0.00020
	PWM_timer.TIM_Prescaler = 21600 - 1;
	PWM_timer.TIM_Period = 200;
	TIM_TimeBaseInit(TIM3, &PWM_timer);

	TIM_OCInitTypeDef oc;
	TIM_OCStructInit(&oc);
	oc.TIM_OCMode = TIM_OCMode_PWM1;
	oc.TIM_OutputState= TIM_OutputState_Enable;
	oc.TIM_Pulse = 0xFF;

	TIM_OC1Init(TIM3, &oc);

	TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);
	TIM_Cmd(TIM3, ENABLE);
	NVIC_EnableIRQ(TIM3_IRQn);
```

```
}

void TIM3_IRQHandler() {
     if(TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET) {

          TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
          period_capture = period_capture / 38;
          TIM_SetCompare1(TIM3, period_capture);

} }

int main() {
     initPort();

     initCaptureTimer();
     initPWMTimer();
     while(1) {

          if(ready_capture) {
               NVIC_DisableIRQ(TIM4_IRQn);
               ready_capture = 0;
               period_capture = diffTime(capture2, capture1);
               NVIC_EnableIRQ(TIM4_IRQn);

} }

return 0; }
```

Приложение В

(обязательное)

Результаты симуляции разработанной программы

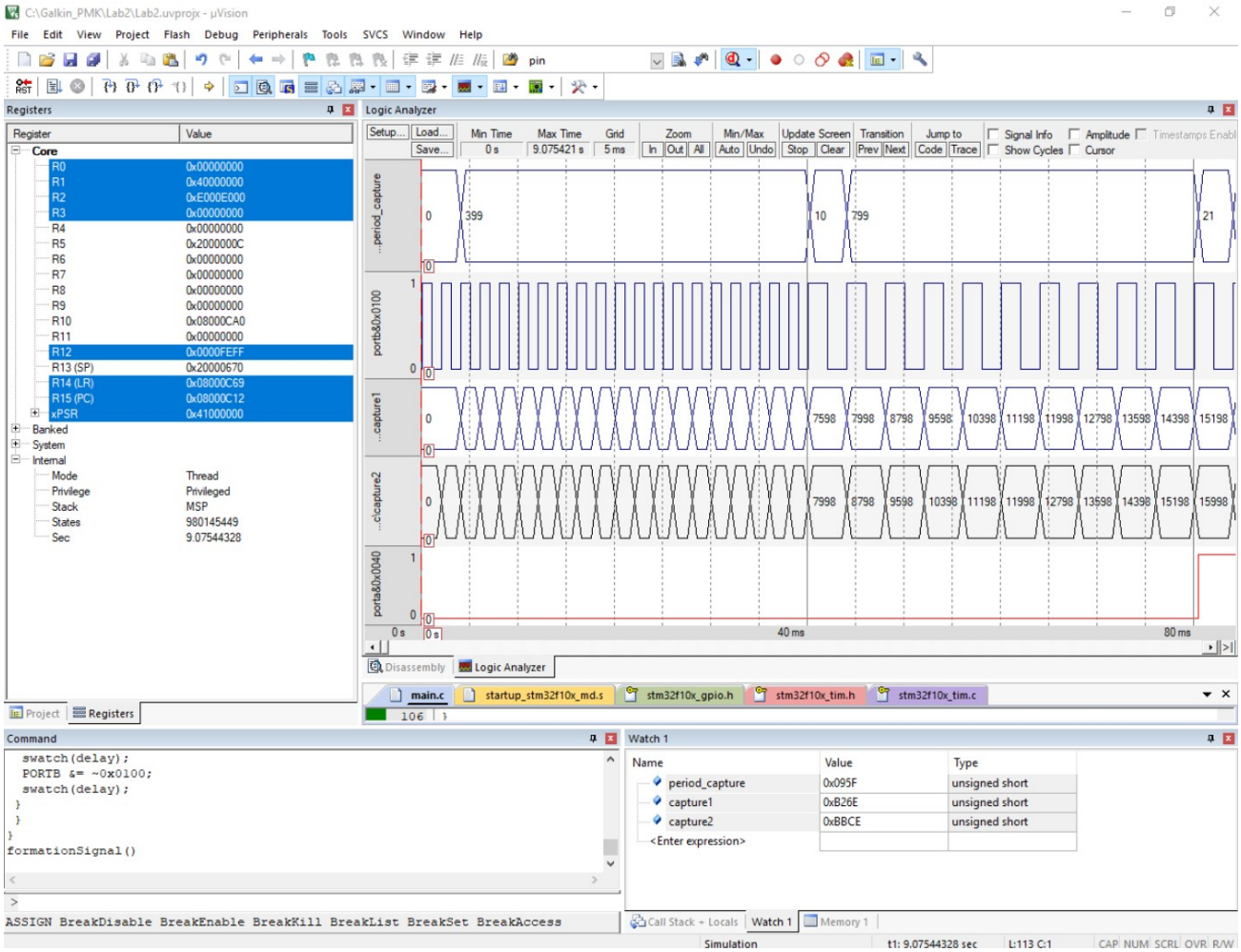

Рисунок 10 - начало симуляции

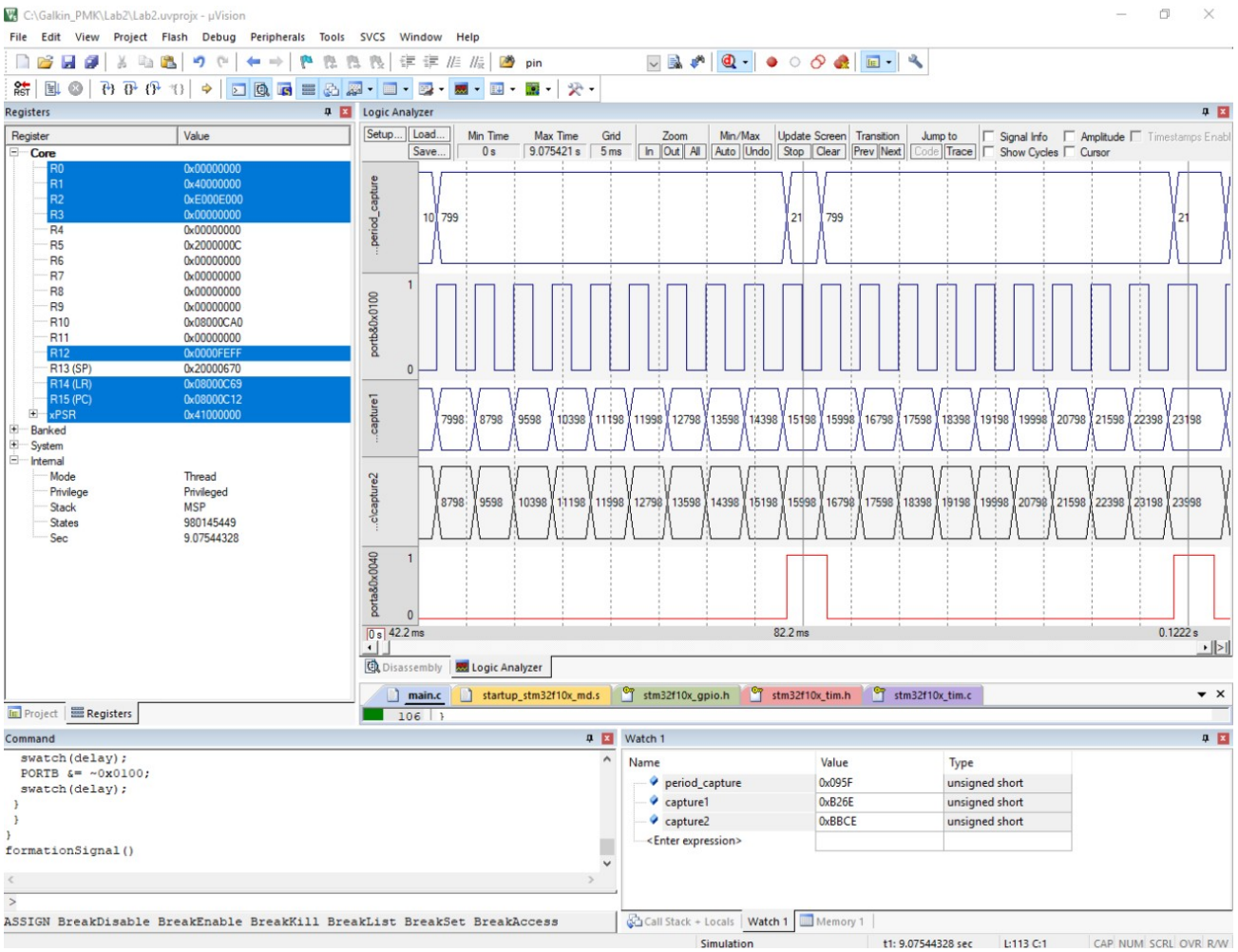

Рисунок 11 - Начало выдачи сигнала ШИМ

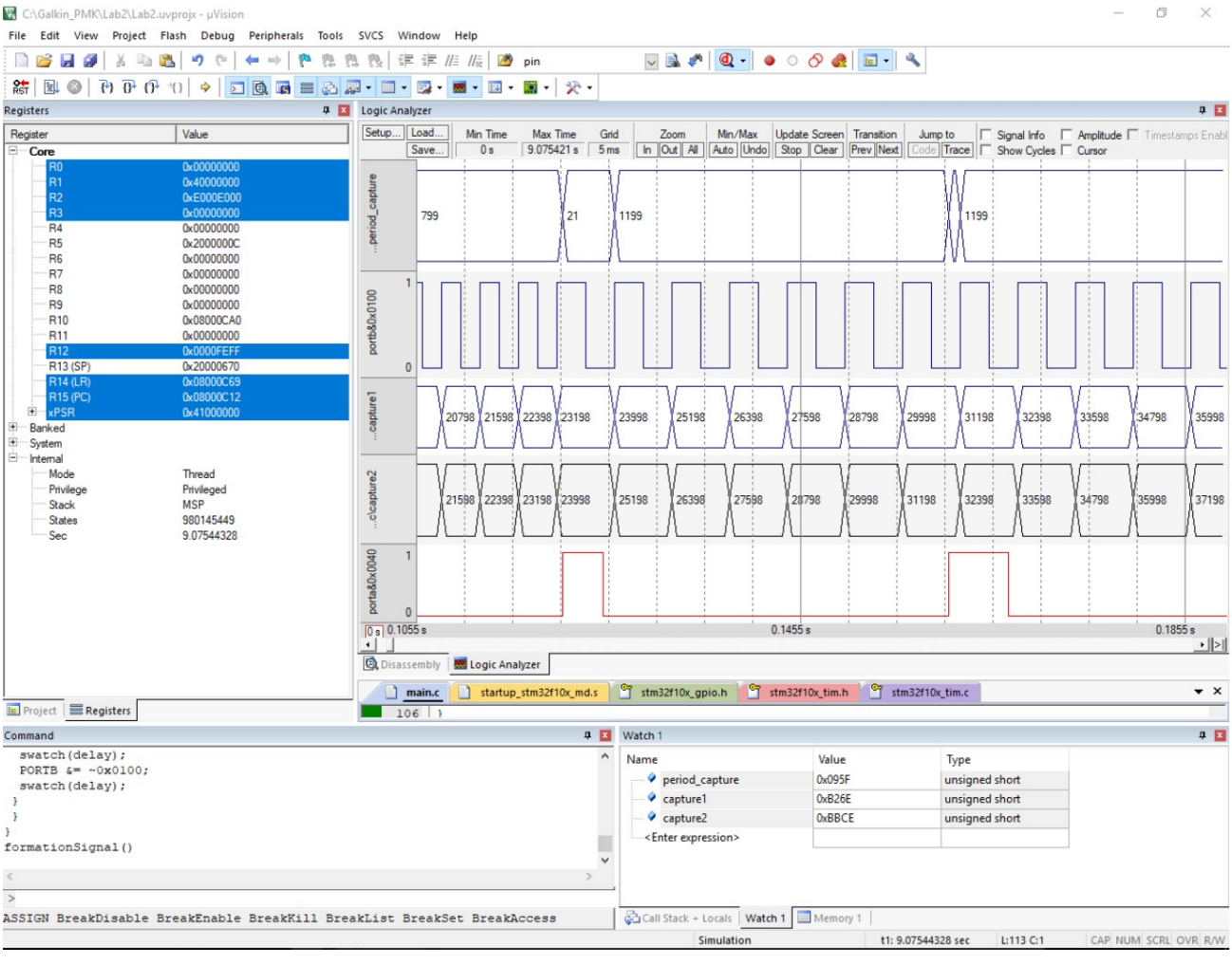

Рисунок 12 - Первое изменение сигнала ШИМ

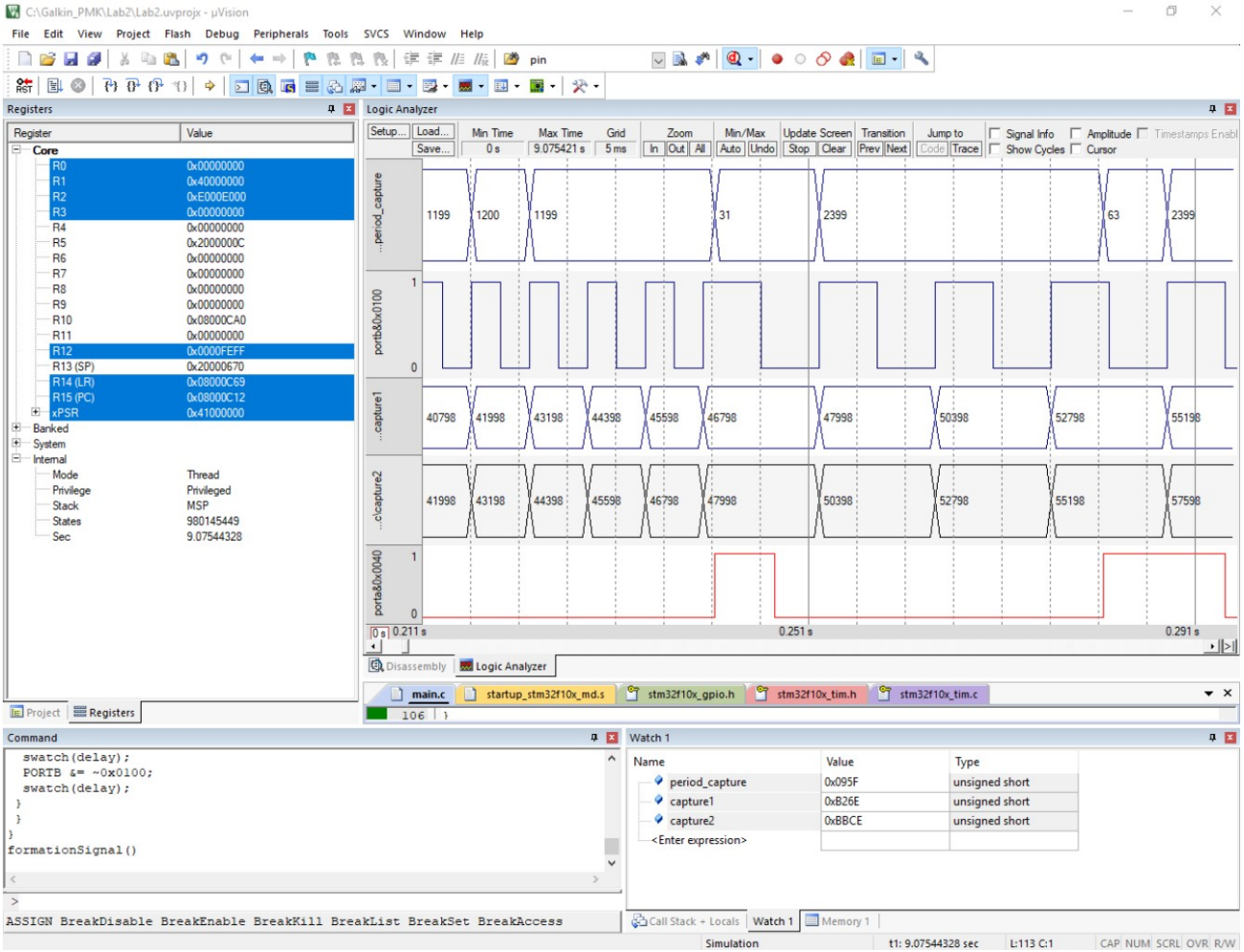

Рисунок 13 - Второе изменение сигнала ШИМ

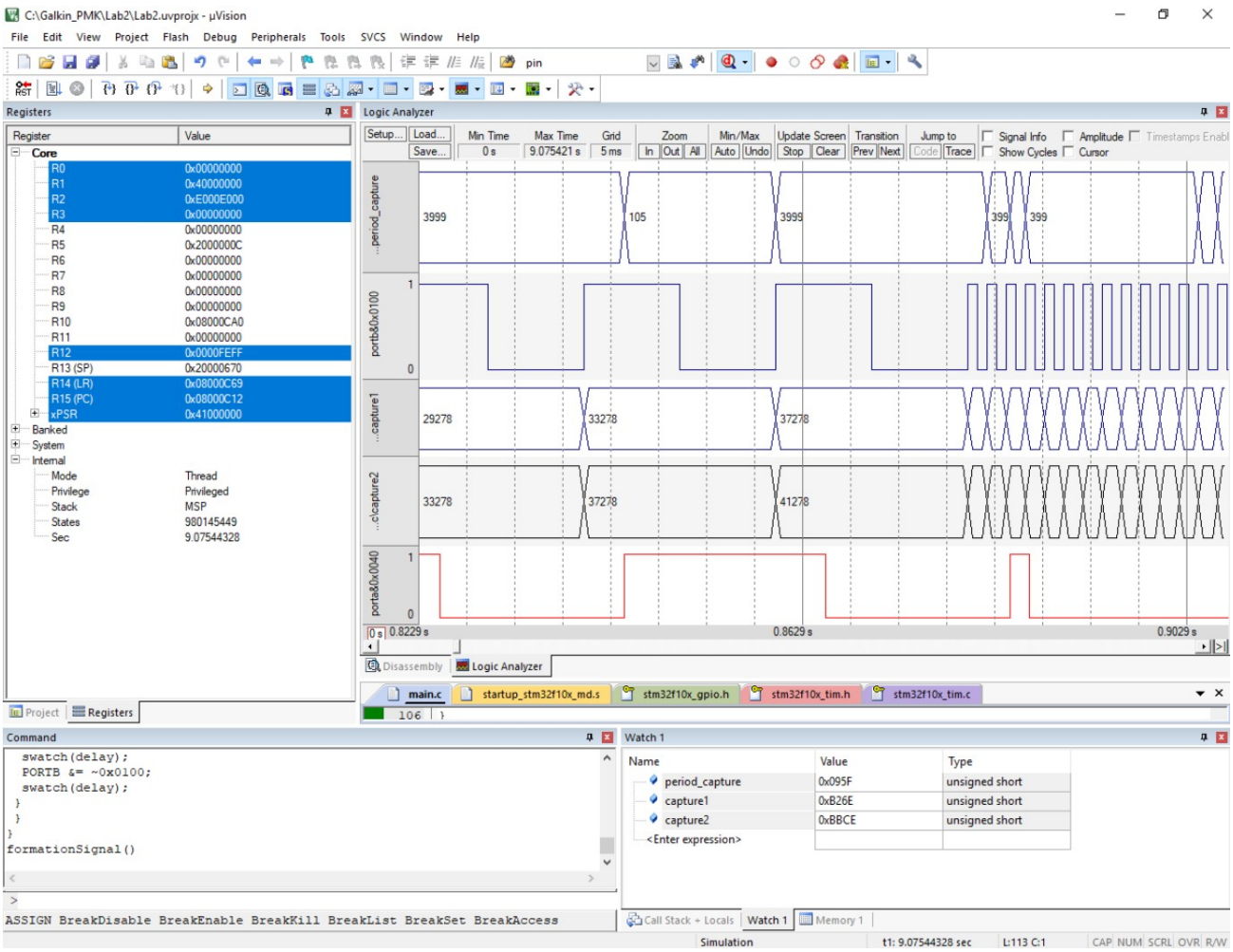

Рисунок 14 - Переход от наибольшего сигнала ШИМ к начальному