Universität Stuttgart
Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen

MEDIS – Module 2

Microcontroller based systems for controlling industrial processes

Chapter 4: Timer and interrupts

M. Seyfarth, Version 0.1

Introduction

- Timing functions are widely used in process automation
  - Start a process 5 seconds after completion of another process.
  - Perform a sequence of steps in a certain timeline.
  - Get a sensor value every 100 ms (e.g. temperature); pass new control data to a motor every 10 ms.
  - Measure the processing time for a task.
  - …

- Example for interrupts
  - Pressing a push button must immediately stop the movement of a robot (collission detection)
  - Calculate the algorithm of a closed loop controller every 1ms.
  - Signal the completion of a task.

Universität Stuttgart

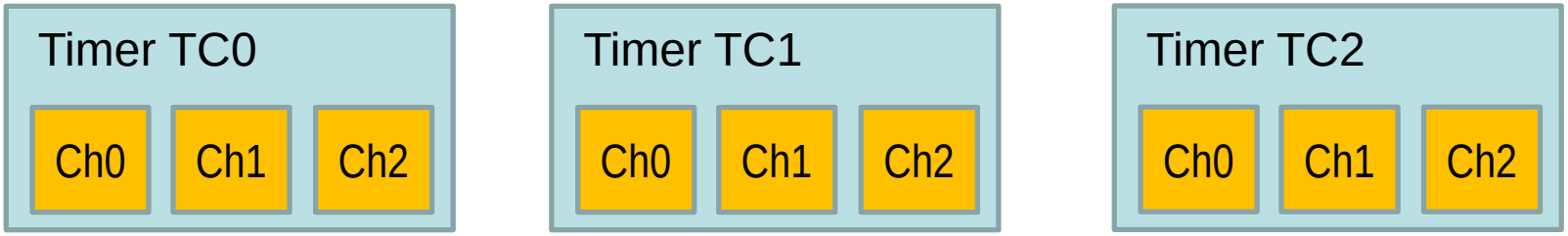
1.1 Timer functions

1.2 Interrupt handling

Timer in microcontroller Arduino Due

Universität Stuttgart

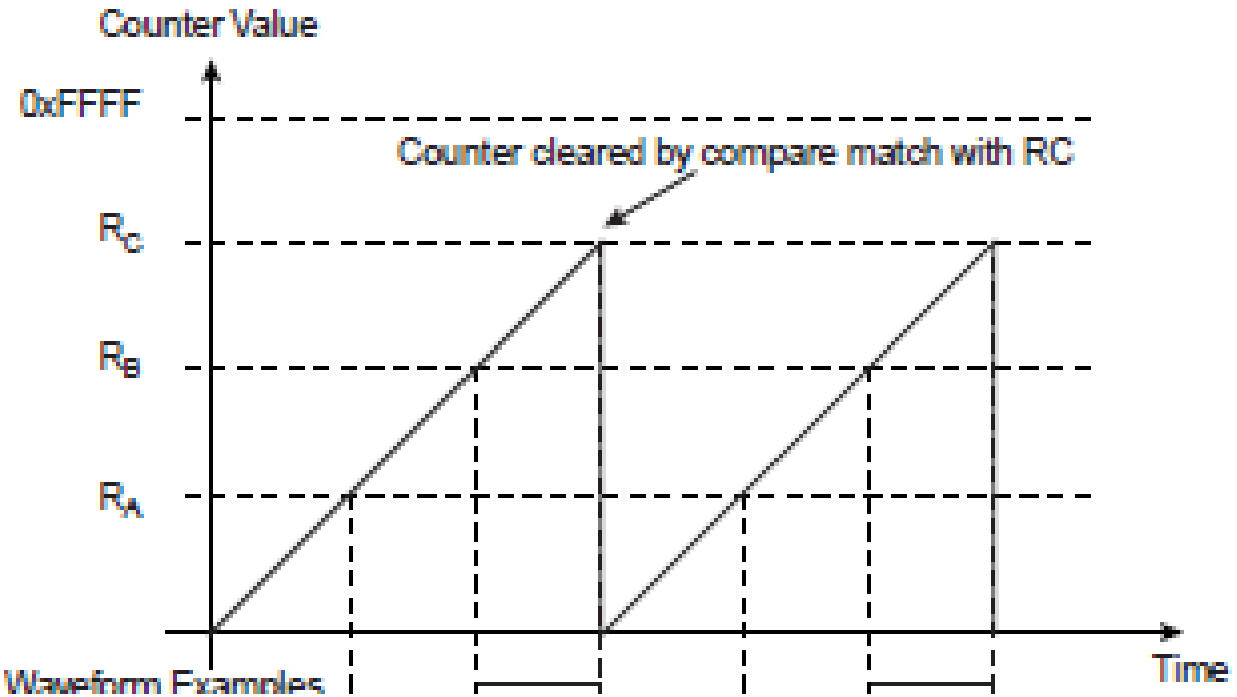
- The microprocessor AT91SAM3X embedds 9 general-purpose 32-bit timers
- Organized in 3 blocks each containing 3 channels



- Timer are clocked by Master Clock (derived from processors quartz crystal oscillator)
- Functions of timer:
  - Frequency measurement
  - Event counting
  - Interval measurement
  - Pulse Generation
  - Delay timing
  - Pulse Width Modulation
  - …

Timer in microcontroller Arduino Due

- Timer are based on 32-bit counter register
- Value of counter is incremented at each positive edge of Master clock
- Counter throws an interrupt if its abort condition or an overflow occurs
- Abort condition must be calculated from Master Clock an desired duration



Timer functions on Arduino Due

- Different timer functions on Arduino
  - Core functions
  - Timer library DueTimer.h

- Core Functions:
  - `delay(unsigned long ms);`
  - `delayMicroseconds(unsigned int us);`
  - Pause the program for the desired amount of time

  - `unsigned long millis();`
  - Returns the number of milliseconds since the Arduino Due began running the current program. Overflow will happen after approximately 50 days.

  - `unsigned long micros();`
  - Returns the number of microseconds since the Arduino Due began running the current program. Overflow will happen after approximately 70 minutes.

Example timer functions – Blinking LED

- A LED (wired to pin 5) should blink with 1 Hz.

```
void setup()
{   pinMode(5, OUTPUT);
}
void loop()
{   digitalWrite(5, HIGH); // LED on
    delay(500);            // wait 500ms
    digitalWrite(5, LOW);  // LED off
    delay(500);            // wait 500 ms
}
```

Example timer functions – Display sensor values periodically

Universität Stuttgart

- A measured temperature value (sensor wired to analog pin 1) should be displayed every two seconds. In parallel the number of pushes on a button (wired to digital pin 5) should be counted and be displayed, too.

```
int count = 0;     // global variable for counting
void setup()
{   Serial.begin(9600);
    pinMode(5, INPUT);
}
void loop()
{   int temp;
    temp = analogRead(1);  // read analog value
    Serial.print(„Temperature: “);
    Serial.println(temp);
    if (digitalRead(5) == HIGH)
       count++;
    Serial.print(„Pushes on button: “);
    Serial.println(count);
    delay(2000);           // wait 2000ms
}
```

Example timer functions – Blinking LED without pausing program

Universität Stuttgart

- Blink a LED (on pin 5) with 1 Hz and in parallel count pushes on a button (pin 6).

```
int count = 0;        // global variable
int ledState = LOW; // used to store the state of LED
long previousMillis = 0; // last time LED was changed
long interval = 500; // interval to blink
void setup()
{   Serial.begin(9600);
    pinMode(5, OUTPUT);
    pinMode(6, INPUT);
}
void loop()
{   unsigned int currentMillis = millis();
    if (currentMillis – previousMillis > interval)
    {  previousMillis = currentMillis;
       ledState = !ledState // toggle LED state
       digitalWrite(5, ledState) // write LED state to pin
    }
    if (digitalRead(6) == HIGH)
       count++;
}
```

Example timer functions – Debounce push button

Universität Stuttgart

- Mechanical push buttons bounce when pressed or released → multiple detection
- Debounce routine: a signal must be constant for more than 25ms

```
int buttonState = LOW;
int lastButtonState = LOW;
long lastDebounceTime = 0;
long debounceInterval = 25; // length of debounce interval
void setup()
{   pinMode(6, INPUT);
}
void loop()
{   int button = digitalRead(6);
    if (button != lastButtonState)
    { lastDebounceTime = millis();
    }
    if ((millis()- lastDebounceTime) > debounceInterval)
    { buttonState = button; // buttonState is the debounced signal
    }
    lastButtonState = button; // reinitialize lastButtonState
}
```

Timer functions on Arduino Due

Universität Stuttgart

- Timer Library DueTimer.h:
  - Comfortable, object-oriented access to timer of Arduino Due
  - Download at: https://github.com/ivanseidel/DueTimer

DueTimer
timer: const unsigned short _frequency[NUM_TIMERS]: static double Timers[NUM_TIMERS]: Timer
uint8_t bestClock(double frequency, uint32_t& retRC) DueTimer getAvailable(void) DueTimer(unsigned short _timer); DueTimer& attachInterrupt(void (*isr)()) DueTimer& detachInterrupt(void) DueTimer& start(long microseconds) DueTimer& stop() DueTimer& setFrequency(double frequency) DueTimer& setPeriod(unsigned long microseconds) Double getFrequency(void) long getPeriod(void)

Timer functions on Arduino Due

Universität Stuttgart

- Timer Library DueTimer.h:
  - Include DueTimer.h
  - Automatically 9 Timer objects are created: Timer0, Timer1, …, Timer8
  - Use of Timer objects:
    - Set function which should be called if timer elapses
    - `Timerx.attachInterrupt(nameOfFunction); // x = 0…8`
    - Start timer with a desired period in microseconds (variable type ‚long')
    - `Timerx.start(microseconds); // x = 0…8`
    - Stop timer if no longer used
    - `Timerx.stop(); // x = 0…8`
    - Example for a function to be called by a timer
    - ```
      HandlerTimerx() // x = 0…8
      { counter++;
      }
      ```
    - Called function should be short!
      ```

Universität Stuttgart

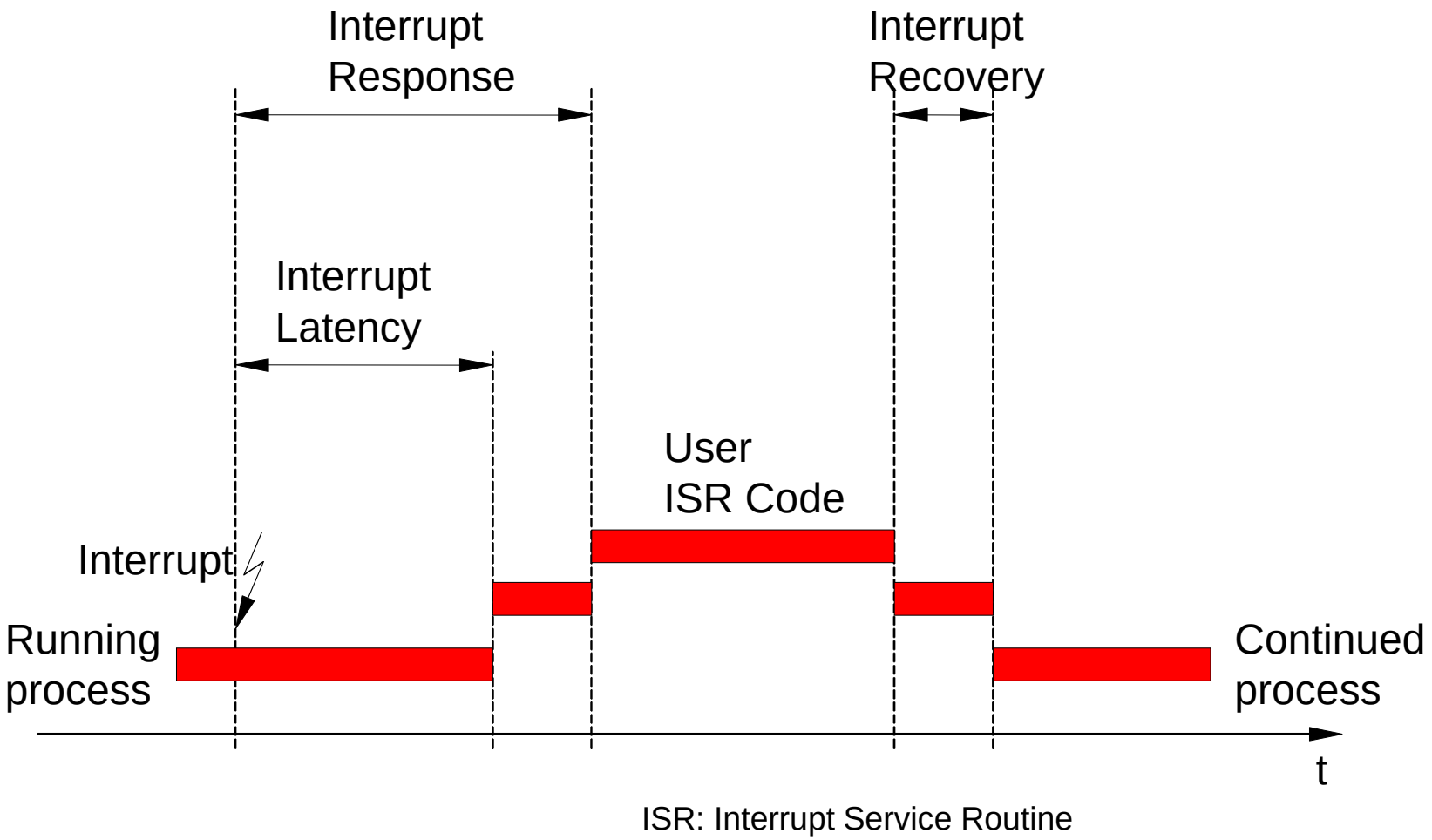
1.1 Timer functions

1.2 Interrupt handling

Interrupts

- Interrupt is a request for a pause in a running process, to perform another (time critical) task. The paused process will be resumed after completion of the interrupt task.
- IRQ: Interrupt request, the triggering event
- ISR: Interrupt Service Routine, tasks to be done when interrupt occured
- Immediate reaction on event
- No need for cyclic polling

Timing of interrupts



ISR: Interrupt Service Routine

Interrupt Sources

- External sources, digital pins
  - Push button or rotary encoder at a digital pin
  - Bussystem or I/O-system
- Timer/Counter
  - Overflow
  - Counting value is reached
- Requirements for Interrupt Service Routines (ISR)
  - As short as possible, as other interrupts and other processes are blocked
  - Maximum duration as long as period of interrupt event, otherwise interrupts may be lost
  - No/little writing access to variables of the interrupted process, changes may be lost when returning from ISR

Interrupt use on Arduino Due

- Arduino Due contains Core functions for interrupt handling
- Facilitates easy use

- Notes
  - Inside the ISR `delay()` won't work, `delayMicroseconds()` will work as expected
  - Inside the ISR the value of `millis` will not increment.
  - Serial data received while in the ISR may be lost
  - An ISR can have no parameters and should return no values → use global variables!
  - Declare variables used in ISR as `volatile` to ensure they are updated correctly
  - An ISR blocks other processes and other interrupts .

Interrupt use on Arduino Due

- Core Functions:
  - `attachInterrupt(pin, ISR, mode);`
    - pin: number of pin where interrupt source is wired (e.g. push button)
    - ISR: name of function for interrupt service routine to be called
    - mode: defines when the interrupt should be triggered. Four constants are predefined as valid values:
      - LOW to trigger the interrupt whenever the pin is low,
      - CHANGE to trigger the interrupt whenever the pin changes value
      - RISING to trigger when the pin goes from low to high,
      - FALLING for when the pin goes from high to low.
      - HIGH to trigger the interrupt whenever the pin is high.
  - Example: blinks LED on pin 13 when button on pin 4 is pressed

```
volatile int state = LOW;
void setup()
{ pinMode(13, OUTPUT);
  attachInterrupt(4, blink, CHANGE);
}
void loop()
{ digitalWrite(pin, state);
}
void blink()
{ state = !state;
}
```

Interrupt use on Arduino Due

- Core Functions:
  - `detachInterrupt(pin);`
    - pin: the pin number of the the interrupt to disable
  - `noInterrupts();`
    - Disables all interrupts (also system interrupts for delay, millis, PWM, serial communication, …)
    - Use very carefully only for extreme critical sections that may not be interrupted
  - `Interrupts();`
    - Re-enables interrupts again
  - Example:

```
void setup() {}

void loop()
{
  noInterrupts();
  // critical, time-sensitive code here
  interrupts();
  // other code here
}
```