Tentamen

Enchipsdatorer TEDK18 7,5 hp

Datum:	2021 - 05 - 30
Tid:	14:00 - 19:00
Plats:	
Antal sidor inklusive försättsblad:	9
Bilaga:	Sammanfattning av HAL-funktioner och utdrag från datablad till STM32WB55RG
Tillåtna hjälpmedel:	Valfri räknedosa, linjal
Examinator:	Andreas Axelsson
Examinator besöker tentamen:	nej
Examinator nås under tentamen:	0709-467760
Betyg:	Tentamen ger maximalt 50p För betyg 3 krävs 20p För betyg 4 krävs 30p För betyg 5 krävs 40p

OBS: Skriv namn på alla inlämnade blad.

LYCKA TILL!

Denna sida är avsiktligt tom

1) Besvara följande frågor om mikrodatorsystem: (totalt 20p)

a) Till vad används registret OTYPER? (2p)

Svar: Bestämmer om en GPIO-output är av typen push-pull eller open-drain.

b) Hur många bytes ryms i nedanstående minne? (2p)

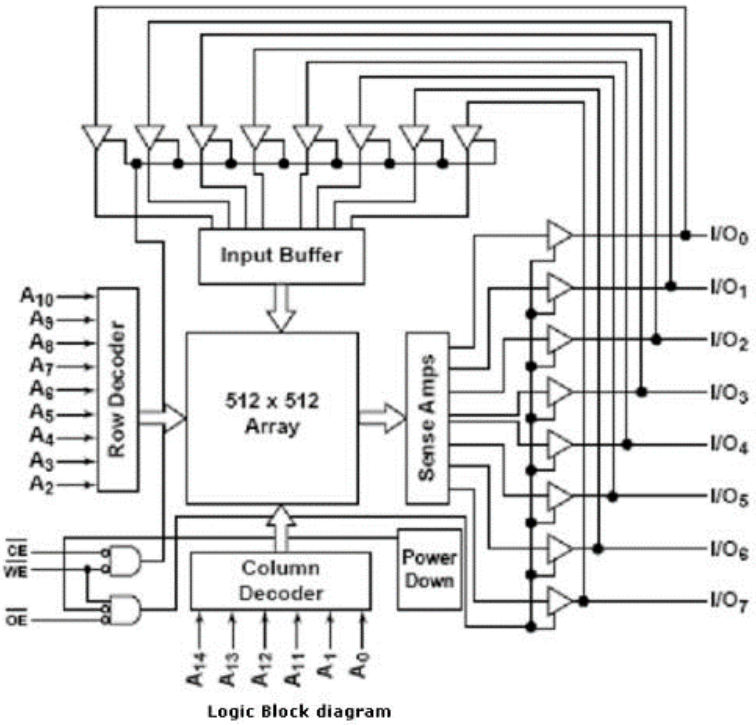
Svar: 2^{15} unika adresser med 1 byte på varje ger 32768 bytes

c) Vilken ordlängd har minnet? (1p)

Svar: Databussen har 8 pinnar, dvs 8 bitars ordlängd (1 byte)

d) Om mikrokontrollern har en 16-bitars databuss D15-D0, hur kan man koppla in två sådana minnen till den? Både avseende adress- och databuss. (3p)

Svar: Om man kopplar in adress-bussen (A0-A14) rakt av till båda minnena och sedan kopplar in $I/0_7$-I/O_0 för ena minnet till D15-D8 och andra minnet till D7-D0.



Logic Block diagram

e) Register R0 innehåller talet 0x57221077, och skrivs ner till minnet på adress 0x20004320. Vad ligger på minnesadresserna 0x20004320-0x20004323 om processorn är big endian, respektive little endian. (2p)

Svar: Big endian: 0x20004320: 0x57 0x22 0x10 0x77, little endian: 0x20004320: 0x77 0x10 0x22 0x57

f) Vad skiljer en RISC- processor från en CISC-processor? Ange en fördel och en nackdel med RISC-processorn jämfört med CISC-processorn (2p)

Svar: RISC (Reduced Instruction Set Computer) använder färre enklare instruktioner som sätts ihop för att göra mer komplexa beräkningar. Varje instruktion gör bara en liten del av helheten, vilket gör att man kan köra på hög klockfrekvens (då varje instruktion går snabbt) och man kan använda pipe-lining. Kräver oftast mer minne.
CISC (Complex Instruction Set Computer) använder sig av mer komplexa instruktioner

som tar längre tid, men som gör flera saker. Krävs oftast mindre minne, men är svårare att köras med hög klockfrekvens eller pipe-linas.

g) Två huvudtyper av arkitekturer finns, Von Neumann och Harvard. Beskriv dessa två (2p)

Svar: Huvudskillnaden är att Von Neumann kör med delad data- och instruktionsbus, medan Harvard har separata bussar.

h) Vad står NVIC för och vad har den för funktion? (2p)

Svar: Nested Vectored Interrupt Controller. Den hanterar avbrottsförfrågningar och tillhörande prioriteter och ser till att rätt avbrott körs på mcu.

i) Beskriv förloppet förloppet från det att mikrokontrollern fått en extern interrupt på en pinne till dess att den utför en callback. (2p)

Svar: Om en GPIO pinne har interrupt aktiverat kommer en förfrågan skickas till NVIC, som kollar om interrupt har tillräckligt hög prioritet att köras. MCU avbryter då sin körning och sparar undan registervärden och återhoppsadress på stacken. Efter det hämtas en pekare i interrupt-vector-tabellen för den aktuella interrupten som pekar på den InterruptHandler-funktion som skall anropas. Använder man HAL (t.ex. genom att man skapat sitt projekt i CubeMX) så finns anrop till HALs drivrutiner i dessa InteruptHandler-funktioner. I drivrutinerna kollas om det finns en user callback, som i så fall anropas.

j) Mikrokontrollerns klockfrekvens är 32 MHz. En räknare som räknar mellan 0-63999 vill man att den ska starta om 50 ggr i sekunden. Vad ska prescalern ha för värde? (2p)

Svar: fclk/(prescaler + 1)/(reload + 1) = 50, dvs 32000000/(prescaler + 1)/64000 = 50. Det ger ett värde på prescaler = 9. Då delar prescalern fclk först med 9+1 = 10 ggr.

2) Du har fått i uppgift att bygga en Sous Vide. Det är en värmare som värmer upp ett vattenbad till en exakt temperatur. Man lägger ner olika matvaror i vattenbadet i vakuumpackade påsar. Sedan ligger de där i flera timmar och blir perfekt tillagade. För att hela vattenbadet ska få samma temperatur finns en liten cirkulationspump som pumpar runt vattnet över ett värmeelement. Temperaturen mäts kontinuerligt av en temperatursensor. Användaren kan ställa in tid och önskad temperatur på en display och tillhörande knappar, UP, DOWN, NEXT och START. Med dessa knappar ökar/minskar man önskad tid och önskad temperatur. Med NEXT växlar man om det är tid eller temperatur man ändrar. Och START startar förloppet. Knapparna är elektriskt avstudsade. Displayen kopplas in via I2C. Funktioner för displayen finns i appendix.

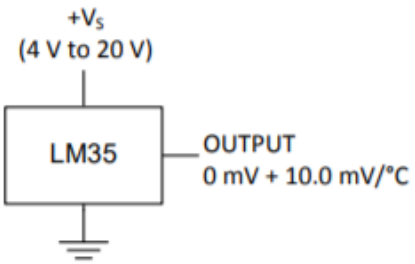
- Temperatursensorn är inkopplad på PC0 (ADC1 ingång IN1)
- Cirkulationspumpens on/off är inkopplad på PA4
- Värmaren styrs av en effekt-transistor på PA8 (TIM1 PWM CH1)
- Displayen är inkopplad via I2C (LCD)
- Knapparna för att mata in är kopplade till PC0, PC1, PC2, PC3 (UP, DOWN, NEXT, START)
- LCD displayen har I2C adress 0x4E och är ansluten till I2C1-bussen.

a) Temperatursensorn är en LM35 (se figur nedan). Den ger en spänning ut som är proportionell mot temperaturen. V_out = 0.010 * Temperatur [V / Grad C]. Skriv en funktion Read_temp() som läser av spänningen med hälp av ADC och returnerar en float med temperaturen i grader Celsius. Full scale för ADC är 4095 vid 3.3V in.

(3p)

Svar: Ansätt att ADC är rätt uppsatt i CubeMX. Uppgiften kan lösas på många sätt. Nedan visas ett sätt med pollning. Med Interrupt eller DMA är två andra sätt.

```
float Read_temp()
{
   HAL_ADC_Start(&hadc1);
   if (HAL_ADC_PollForConversion(&hadc1, 1000) != HAL_OK)
   {
      return -1000.0; // Error. Return impossible temperature as fault code
   }
   uint32_t value = HAL_ADC_GetValue(&hadc1);
   return (value * 3.3 / 4095) / 0.01;
}
```



b) Skriv två funktioner. Pump_on() och Pump_off() som slår på, respektive slår av cirkulationspumpen. (2p)

Svar:

void Pump_on()

```
{
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
}

void Pump_off()
{
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
}
```

c) Värmaren drar mycket ström och kan därför inte drivas direkt av en I/O-pinne, utan drivs via en MOSFET (se figur nedan). För att styra effekten på värme-elementet används PWM. Dvs när man har 100% duty cycle går den på full effekt, och när man t.ex. har 50% duty cycle går den på halv effekt. TIM1 används till detta som har en 16-bitars räkare.

i) Om MCU kör på 32 MHz, och vi vill ha en PWM-frekvens på 5 Hz. Ange lämpliga värden på pre-scaler och counter reload. (vi vill kunna sätta duty cycle i minst 100 steg) (2p)

Svar: T.ex. 32000000/6400/1000, ger prescaler = 6399 och reload = 999. Då får man en PWM med duty cycle 1000 steg.

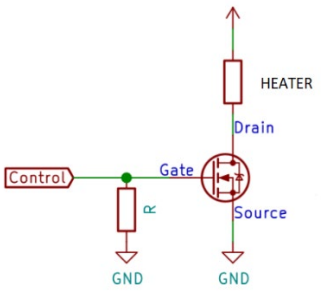
ii) Skriv en funktion som tar ett argument, duty: Heater_on(float duty) som ställer in rätt värden på timerns PWM. (2p)

Svar:

```
void Heater_on(float duty) // duty går mellan 0.0 – 1.0 (100%)
{
   __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 1000 * duty);
}
```

iii) Hur stor ström kan typiskt en I/O-pinne normalt driva på en STM32? (1p)

Svar: en I/O pinne för STM32 kan typiskt driva +-20mA. Obs. Det brukar även finnas en total ström som alla GPIO-pinnarna kan driva samtidigt.



d) Gör en funktion Control_temp() som reglerar temperaturen i vattenbehållaren, dvs den läser av temperaturen och ställer sedan lämplig effekt på värmaren. Använd funktionerna ni gjort tidigare.

Diff_Temp = Önskad temperatur - faktisk temperatur. Om:

1. Diff_Temp > 10 så vill vi ha 100% effekt på värmaren
2. Diff_Temp > 5 så vill vi ha 50% effekt på värmaren
3. Diff_Temp > 0 grader vill vi ha 25% effekt på värmaren
4. Diff_Temp < 0 grader vill vi ha 0% effekt på värmare

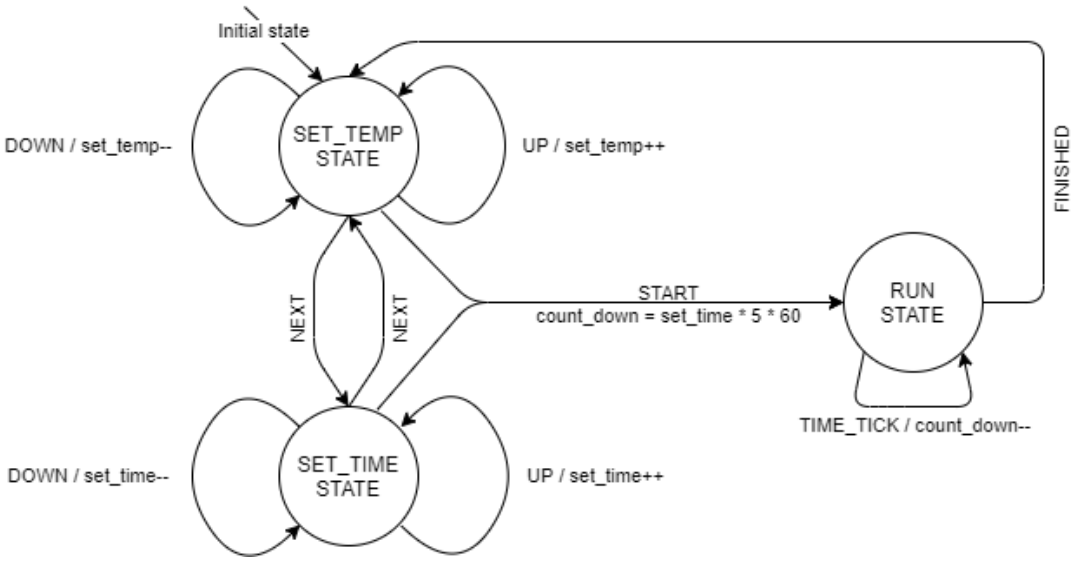
(3p)

Svar:

```
void Control_temp()
```

```
{
   float actual_temp = Read_temp();
   if (actual_temp < 0)
   {
      return;
   }
   float diff_temp = set_temp – actual_temp;
   if (diff_temp > 10.0)
   {
      Heater_on(1.0);
   }
   else if (diff_temp > 5.0)
   {
      Heater_on(0.50);
   }
   else if (diff_temp > 0.0)
   {
      Heater_on(0.25);
   }
   else
   {
      Heater_on(0.0);
   }
}
```

e) Gör ett tillståndsdiagram över er design, med de tillstånd som ni hittar, och de events (händelser) som hoppar mellan tillstånden. (3p)
Svar:



f) Skapa en tillståndsvariabel med en enum typ som namnger era tillstånd. (2p)
Svar:

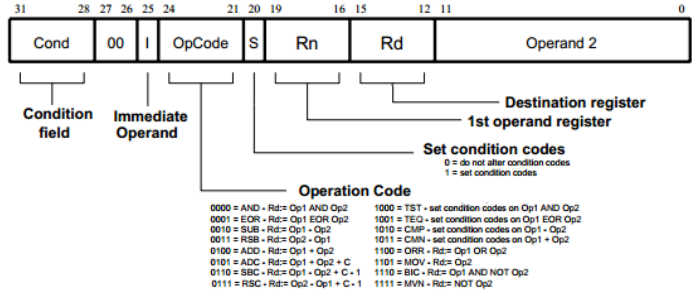
```
typedef enum {
   SET_TEMP_STATE, SET_TIME_STATE, RUN_STATE
} StateType;
StateType state = SET_TEMP_STATE; // Initial state is SET_TEMP_STATE
```

g) Skriv resterande program (initiering och huvud-loop) och eventuella andra hjälpfunktion som gör att användaren kan ställa in önskad tid och temperatur, och att sedan temperatur-

regleringen körs. När tiden är nere på noll skall pump och värmare stängas av. Implementera som en State Machine, med fördel enligt det tillståndsdiagram ni tog fram i uppgift e). (7p)

Svar: Se sista sidan för svar.

3) Svara på nedanstående frågor
   a) Ibland kan har man inte elektriskt avstudsade knappar. Hur kan man avstudsa knapparna i mjukvara? (2p)
   Svar: Man kan vänta tills knapp-ingången ändrar sig (antingen via polling eller interrupt). Sedan läser man av ingången en viss tid senare (där denna tid är längre än tiden studsarna tar) och ser om pinnen är hög eller låg. Den signalen är då studsfri då knappen hunnit stabilisera sig.
   b) Beskriv hur en maskinkodsinstruktion är uppbyggd (2p)
   Svar: Olika bitfält i maskinkodsinstruktionen (som vanligtvis är 32 eller 16 bitar lång), bestämmer vad som ska göras, vilka flaggor som vilka register som ska användas.



   c) Hur många 16-bitars Timers har STM32WB55RG? (1p)
   Svar: Det finns 5st 16-bitars timers.
   - 1x 16-bit, four channels advanced timer
   - 2x 16-bit, two channels timer
   - 1x 32-bit, four channels timer
   - 2x 16-bit ultra-low-power timer

APPENDIX: LCD-SUBRUTINER

<table>
<tr><th>Gäller PIC:</th><th>Gäller ARM:</th></tr>
<tr><td colspan="2">Definitioner för att underlätta läsning av koden:</td></tr>
<tr><td>#define LCD_E RD7
#define LCD_RW RD6
#define LCD_RS RD5
#define LCD_FCN_SET 0x38
#define LCD_ENTRY_MODE 0x06
#define LCD_DISPLAY_ON 0x0E
#define LCD_DISPLAY_CLEAR 0x01
#define LCD_CURSOR_HOME 0x80
#define LCD_CURSOR_LINE2 0xC0
#define LCD_COMMAND 0
#define LCD_DATA 1
#define HIGH 1
#define LOW 0</td><td><pre>typedef struct {
    I2C_HanfleTypeDef *hi2c;
    Uint8_t DevAddress;
    Uint8_t data;

} TextLCDType;</pre></td></tr>
<tr><td colspan="2">Funktioner:</td></tr>
<tr><td>void lcd_tkn(unsigned char t);</td><td>void TextLCD_Putchar(TextLCDType *lcd, char data);</td></tr>
<tr><td>void display_ascii (char *s);</td><td>void TextLCD_Puts(TextLCDType *lcd, char *string);</td></tr>
<tr><td>void itoa(uint8_t tal, char *siffror);</td><td>void itoa(uint32_t tal, char *siffror);</td></tr>
<tr><td>void lcd_clear(void);</td><td>void TextLCD_Clear(TextLCDType *lcd);</td></tr>
<tr><td>void lcd_home(void);</td><td>void TextLCD_Home (TextLCDType *lcd);</td></tr>
<tr><td>void lcd_ini (void);</td><td>void TextLCD_Init(TextLCDType *lcd, I2C_HandleTypeDef *hi2c, uint8_t DevAddress);</td></tr>
<tr><td></td><td>void TextLCD_Printf (TextLCDType *lcd, char *message, ...);</td></tr>
</table>

STM32 HAL-sammanfattning.

- `void HAL_Delay(uint32_t Delay)`
- `uint32_t HAL_GetTick( void )`
- `void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)`
- `GPIO_PinState HAL_GPIO_ReadPin( GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)`
- `void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)`
- `void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)`
- `HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)`
- `HAL_StatusTypeDef HAL_TIM_PWM_Start(TIM_HandleTypeDef *htim, uint32_t Channel)`
- `__HAL_TIM_SET_COMPARE(__HANDLE__, __CHANNEL__, __COMPARE__)`
- `void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim):`
- `void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)`
- `__HAL_ADC_GET_FLAG(__HANDLE__, __FLAG__)`
- `uint32_t HAL_ADC_GetValue(ADC_HandleTypeDef* hadc)`
- `HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandelTypeDef* hadc)`
- `void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc):`

Svar uppgift 2g:

```
typedef enum {
	SET_TEMP_STATE, SET_TIME_STATE, RUN_STATE
} StateType;

typedef enum {
	NONE, UP, DOWN, NEXT, START, TIME_TICK, FINISHED
} EventType;

// Global variables

StateType state = SET_TEMP_STATE;
EventType event = NONE;

uint32_t set_temp = 0; // contains the desired set temperature in degree Celsius
uint32_t set_time = 0; // contains the number of desired minutes to run the Sous Vide

uint32_t count_down; // contains number of remaining ticks. Tick rate is 5 Hz.

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
	if (htim->Instance == TIM1)
	{
		event = TIME_TICK;
	}
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
	if (GPIO_Pin == BUTTON_UP)
	{
		event = UP;
	}
	if (GPIO_Pin == BUTTON_DOWN)
	{
		event = DOWN;
	}
	if (GPIO_Pin == BUTTON_NEXT)
	{
		event = NEXT;
	}
	if (GPIO_Pin == BUTTON_START)
	{
		event = START;
	}
}

void StateMachine()
{
	switch (state)
	{
	case SET_TEMP_STATE:
		Print_set_temp_on_LCD();
		if (event == NEXT)
		{
			event = NONE;
			state = SET_TIME_STATE;
		}
		if (event == UP)
		{
			event = NONE;
```

```
                set_temp++; // TODO: Add some upper limit on set_temp
            }
            if (event == DOWN)
            {
                event = NONE;
                set_temp--; // TODO: Add some lower limit on set_temp
            }
            if (event == START)
            {
                event = NONE;
                count_down = set_time * 5 * 60;
                Pump_on();
                state = RUN_STATE;
            }
            break;
        case SET_TIME_STATE:
            Print_set_time_on_LCD();
            if (event == NEXT)
            {
                event = NONE;
                state = SET_TEMP_STATE;
            }
            if (event == UP)
            {
                event = NONE;
                set_time++; // TODO: Add some upper limit on set_time
            }
            if (event == DOWN)
            {
                event = NONE;
                set_time--; // TODO: Add some lower limit on set_time
            }
            if (event == START)
            {
                event = NONE;
                count_down = set_time * 5 * 60;
                Pump_on();
                state = RUN_STATE;
            }
            break;
        case RUN_STATE:
            Print_current_temp_and_time_on_LCD();
            Control_temp();
            if (event == TIME_TICK)
            {
                event = NONE;
                count_down--;
                if (count_down <= 0)
                {
                    event = FINISHED; // event FINISHED added just for clarity
                }
            }
            if (event == FINISHED)
            {
                event = NONE;
                Pump_off();
                Heater_on(0.0);
                state = SET_TEMP_STATE;
            }
        }
}

int main(void)
```

```
{
	/* ... */

	HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
	HAL_TIM_Base_Start_IT(&htim1);
	TextLCD_Init(&lcd, &hi2c1, 0x4E);

	while(1)
	{
		StateMachine();
	}
}
```