Anmerkungen

Beim Scheduler haben wir wie bei der Clock letztes mal mit Makros gearbeitet (`threading.a51`). Die Makros werden dann über einen in C# selbst geschriebenen pre-assembler (siehe GitHub cross-platform releases (link)) aufgelöst (`threading-generated.a51`) und das Programm kann anschließend wie gewohnt über `AS51 V4.exe` assembled werden.

Die im Code verwendeten Makros werden dabei 1:1 durch die im Programm oben definierten Werte ersetzt.

Monitoring

	Software	Hardware	Hybrid
Vorteile	TODO		
Nachteile			
Performanz			
Genauigkeit			

11.2

Bei 12 MHz Taktfrequenz zählt Timer 0 mit einer frequenz von $\frac{12}{12}$ MHz = 1 MHz, also $1 \cdot 10^6$ mal pro Sekunde. Jede Stunde werden also $60 \cdot 60 \cdot 1 \cdot 10^6 = 36 \cdot 10^8$ Timer-Ticks gezählt. Wir benötigen also $\lceil \log_2 \left(36 \cdot 10^8\right) \rceil = \lceil 31.7454 \rceil = 32$ Bit um eine Gesamtlaufzeit von einer Stunde zu erfassen.

Für eine Zeit von 6 Monaten = $\frac{365}{2} = 182.5$ Tage $\implies 182.5 * 24 * 60 * 60 =$ 15768000 Sekunden @1MHz $\implies \lceil \log_2 \left(15768000000000\right) \rceil = 44$ Bit, wobei dies in der Praxis mit 48 Bit = 6 Byte umgesetzt werden würde. Mit 48 Bit ließen sich 2^{48} Timer Ticks repräsentieren, was ca. 8.92 Jahren entspricht.

Scheduler

Konzept

Auf Grund der Echtzeit-Anforderung des Reaktionstask, muss dieser mindestens einmal alle 10 ms laufen. In unserer Implementierung wird nach der Initialisierung der Sortier-Task gestartet und der Timer Interrupt freigegeben. Diese Timer Interrupts werden anschließend jedoch nur alle 10 ms verarbeitet, sodass die Vorgabe für den Reaktionstask gerade so eingehalten wird. Ausgehend von einer Abschätzung (10 ms intervall @12MHz CPU frequenz $\Rightarrow$ 120.000 Takte pro Intervall, bei $\varnothing \sim 2$ Takten pro Instruction liefert ca. 60.000 Instructions pro interrupt) kamen wir zu dem Schluss, dass wir somit nacheinander und konfliktfrei während einer solchen Interruptverarbeitung alle drei anderen Tasks einmal

“komplett durchlaufen lassen” können. Somit werden nacheinander der Reaktions, Clock (und alle 10 sekunden Temperatur)-task verarbeitet. Anschließend wird das Sortieren fortgesetzt.

Aufgrund dieser Erkenntnis haben wir ein Single-Stack-Single-Register bank Design gewählt:

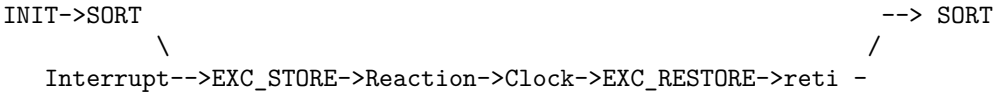
Single Stack Hierbei werden die untersten Stack Frames immer vom Sortierungs Task verwendet (oder der `end` instruction, bzw. stack empty) und dann beim Auftreten eines Interrupts darauf aufbauend die anderen Tasks gepushed. Da diese Tasks nacheinander bis zum `return` durchlaufen, entstehen an dieser Stelle keine Konflikte. Anschließend wird das letzte Stack Frame mit dem “return from interrupt” gepoppt und der Sortier-Task läuft weiter.

Single Register bank Weiterhin verwenden wir nur eine einzelne Register Bank, die wir beim Start der Interruptverarbeitung in den von uns definierten `SWAP` Bereich im internen RAM sichern, und vor dem “return from interrupt” wieder herstellen. Gesichert werden hierbei: `PSW, A, B, r0-7, DPTR (dpl, dph)`, sowie die beiden selbst-definierten 32 bit “Register” `UINT32_0` und `UINT32_1`, bei denen es sich um spezielle RAM Regionen für 32 bit bit-shifting- und Additions-Operationen handelt (benötigt für das Dividieren der 16 bit Summe durch die Konstante `10` bei der Berechnung des Temperaturdurchschnitts). Diese Division wird ersetzt durch eine Multiplikation (weiterhin ersetzt durch `shift` und `add`) mit `0xcccd` (16 bit, mit overflow in 32 bit), gefolgt von einem Bitshift um 19 bits nach rechts. Theoretisch greifen wir nur an dieser Stelle auf die 32 bit Register zu, halten uns jedoch durch das Sichern in den `SWAP`-Space die Möglichkeit offen, dies auch an anderen Stellen zu tun.

Umsetzung (Überblick)

Der Scheduler läuft alle 10 ms und ruft die Funktion `TasksNofityAll()` auf. Dabei wird zuerst der aktuelle `ExecutionContext` durch Aufrufen der `EXC_STORE` Funktion in den `SWAP` Bereich geschrieben. Anschließend wird der `Reaktions` task aufgerufen. Nachdem der `Reaktions` task beendet wurde, wird weiterhin die `Clock` benachrichtigt, dass 10 ms vergangen sind. Abhängig von dem internen Clock-counter wird dann eine Sekunde inkrementiert, wobei dann weiterhin der `Temperatur` task benachrichtigt wird.

Nachdem alle Tasks benachrichtigt wurden wird der originale `ExecutionContext` wieder aus dem `SWAP` Bereich geladen (`EXC_RESTORE`) und der Interrupt beeendet. Somit läuft der `Sort` task nach kurzer Unterbrechung des Schedulers weiter.



RAM layout

Region	Start	End	Size	Description
RESERVED	0x0	0x8	8	register bank 0
STACK	0x8	0x2f	24	stack
RAM 1	0x30	0x3f	16	RAM for Task 1: Scheduler
RAM 2	-	-	0	RAM for Task 2: Reaction (allocation free)
RAM 3	0x40	0x4f	16	RAM for Task 3: Clock
RAM 3B	0x50	0x5f	16	RAM for Task 3B: Temperature
RAM 4	0x60	0x67	8	RAM for Task 4: Sorting (not used)
SWAP	0x68	0x7f	24	swap area for execution context

Reaction-Task (R-Task)

Der Reaction-Task liest alle 10 ms den Wert aus Port 1 aus und schreibt basierend auf der Größenordnung einen festgelegten Wert in Port 3.

Speicheradresse	Information
Port 1	Der auszulesende Wert
Port 3	Der berechnete Wert

Der Wert in Port 3 wird in den zwei least significant bits folgendermaßen gespeichert:

Wertebereiche (Port 1)	Resultat (Port 3 / XH, XL)
$100<x<200$	0, 0
$x \geq 200$	1, 0
$x<100$	0, 1
$x=100 \vee$ error	1, 1

Tests

Die Funktion des Reaktions-Tasks wird im Folgenden durch Tests veranschaulicht und verifiziert:

Wert Port 1	Wert Port 3
0	0x1
99	0x1
100	0x3
101	0x0
150	0x0
199	0x0

Wert Port 1	Wert Port 3
200	0x2
255	0x2

Berechnungs-Task (C-Task)

Der Berechnungs-Task sortiert den gesamten externen RAM aufsteigend nach Größe. Benutzt wird der Bubble-Sort Algorithmus.

Tests

Um die Funktion nachzuweisen, wurde ein Array mit 256 Werten sortiert.

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0000	00	02	05	05	06	06	07	08	08	0B	0C	0E	0F	10	10	11
0010	11	11	12	12	14	16	16	18	18	19	19	1A	1C	1D	1D	21
0020	22	23	24	25	25	26	28	28	2A	2B	2C	2C	2F	31	31	32
0030	33	33	33	34	34	37	39	39	39	3A	3A	3C	3D	3D	3F	40
0040	41	41	43	44	44	44	44	45	46	46	46	47	48	49	4A	4D
0050	4E	4E	4F	4F	52	53	54	56	59	5B	5B	5B	5C	5C	5C	5D
0060	61	61	62	62	64	64	64	65	65	66	66	68	6A	6A	6B	6B
0070	6D	6E	6E	6E	6F	70	73	73	74	74	75	75	77	7A	7A	7B
0080	7C	7D	80	81	81	82	82	83	84	84	85	85	85	85	87	8A
0090	8C	8C	8E	91	92	92	93	95	98	99	9A	9F	9F	A1	A2	A3
00A0	A6	A7	A8	A8	A9	A9	AA	AA	AB	AC	AD	AE	AE	AF	AF	B1
00B0	B1	B2	B2	B5	B7	B8	B9	B9	BA	BA	BB	BC	BD	BE	BF	BF
00C0	C0	C0	C1	C1	C2	C4	C7	C7	C8	C8	C9	CA	CC	CF	D0	D3
00D0	D4	D4	D5	D6	D6	D8	D8	D9	D9	DC	DE	DF	DF	DF	E0	E0
00E0	E1	E2	E3	E4	E6	E7	E8	EA	EB	EC	ED	ED	EE	EF	F0	F0
00F0	F1	F1	F2	F2	F3	F4	F6	F7	F8	F9	FB	FB	FC	FF	FF	FF

Thermometer

Das Thermometer liest alle 10 Sekunden einen Wert aus Port 2 aus.

Speicheradresse	Information
0x50-59	Die 10 letzten Messungen (ausgelesen aus Port 2)
0x5A	Ticks
0x5B	Mittelwert

Speicheradresse	Information
0x5C	Tendenz
0x5D	Pointer auf die aktuelle Adresse ausgehend von 0x50
0x5E-5F	High- und Low-Nibble der Summe für die Mittelwertberechnung

Die Tendenz kann folgende Werte betragen:

Wert	Bedeutung
0x0	Fallend
0x1	Steigend
0xFF	Keine Änderung

Tests

Tests für die Mittelwertberechnung Es wurden 10 Messungen $M_1 \dots M_{10}$ durchgeführt:

M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	Mittelwert
0	0	0	0	0	0	0	0	0	0	0
50d	0	0	0	0	0	0	0	0	0	5d
50d	50d	0	0	0	0	0	0	0	0	10d
50d	50d	50d	0	0	0	0	0	0	0	15d
50d	50d	50d	50d	0	0	0	0	0	0	20d
50d	50d	50d	50d	50d	0	0	0	0	0	25d
50d	50d	50d	50d	50d	50d	0	0	0	0	30d
50d	50d	50d	50d	50d	50d	50d	0	0	0	35d
50d	50d	50d	50d	50d	50d	50d	50d	0	0	40d
50d	50d	50d	50d	50d	50d	50d	50d	50d	0	45d
50d	50d	50d	50d	50d	50d	50d	50d	50d	50d	50d

Tests für die Tendenzberechnung Die Tendenz wird basierend der beiden zuletzt ermittelten Mittelwerte erstellt:

Mittelwert	$\Rightarrow$	Mittelwert	Tendenz
0	$\Rightarrow$	0	0xFF
0	$\Rightarrow$	0xA	0x1
0xA	$\Rightarrow$	0	0

Clock

Speicheradresse	Information
0x40	Stunden
0x41	Minuten
0x42	Sekunden
0x43	Max-Stunden
0x44	Max-Minuten
0x45	Max-Sekunden

Manuelles Stellen der Clock

Das lower nibble des Ports 0 wird genutzt um den Modus der Clock auszuwählen: - Die niedrigeren 2 bit kontrollieren den Modus - Die oberen 2 bit selektieren die zu setzenden Werte

Modus	Beschreibung
0	normal
1	increment
2	decrement
3	invalid

Selektion	Beschreibung
0	hours
1	minutes
2	seconds
3	invalid

Port 0 wird jede Sekunde abgefragt und die jeweilige Operation wird anschließend ausgeführt. Das Stellen der einzelnen Spalten geschieht unabhängig von den anderen. Es werden keine 'carries' erzeugt.

Tests

Die Übergänge unserer Uhr wurden in folgenden Szenarien für den normalen Modus (die zwei least significant bits aus Port 0 = 00) geprüft:

BEACHTE: Auf der linken Seite von "⇒" sind Werte zum Zeitpunkt t angegeben. Auf der rechten Seite von "⇒" sind Werte zum Zeitpunkt t+1 angegeben.

Stunden	Minuten	Sekunden	⇒	Stunden	Minuten	Sekunden
0	0	0	⇒	0	0	1
0	0	59	⇒	0	1	0

Stunden	Minuten	Sekunden	⇒	Stunden	Minuten	Sekunden
0	59	59	⇒	1	0	0
23	59	59	⇒	0	0	0