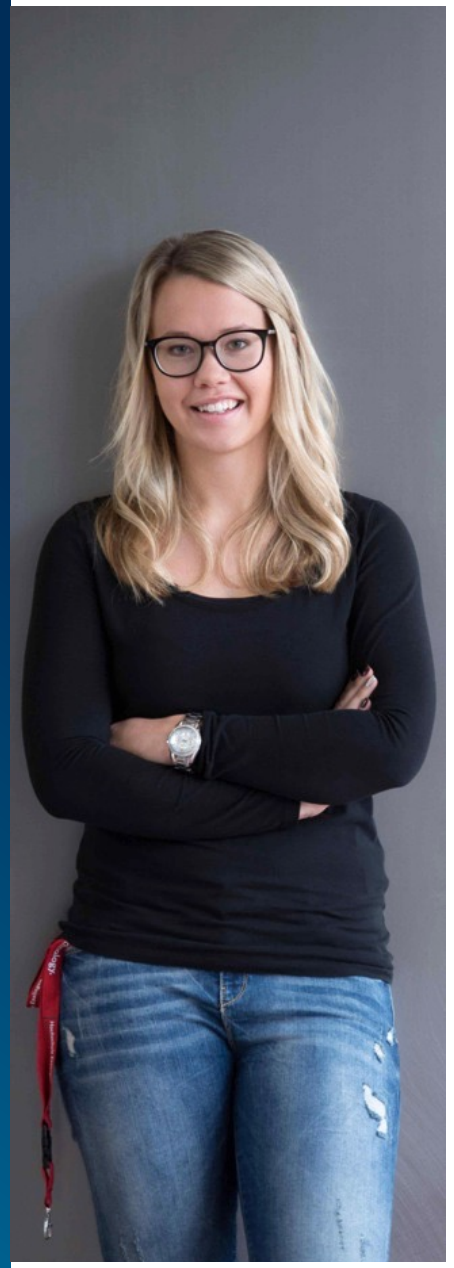
HOCHSCHULE
ESSLINGEN



LECTURE COMPUTER ARCHITECTURE

PERIPHERALS, DIGITAL/ ANALOG CONVERSION AND TIMED PERIPHERALS

RAINER KELLER

HOCHSCHULE
ESSLINGEN

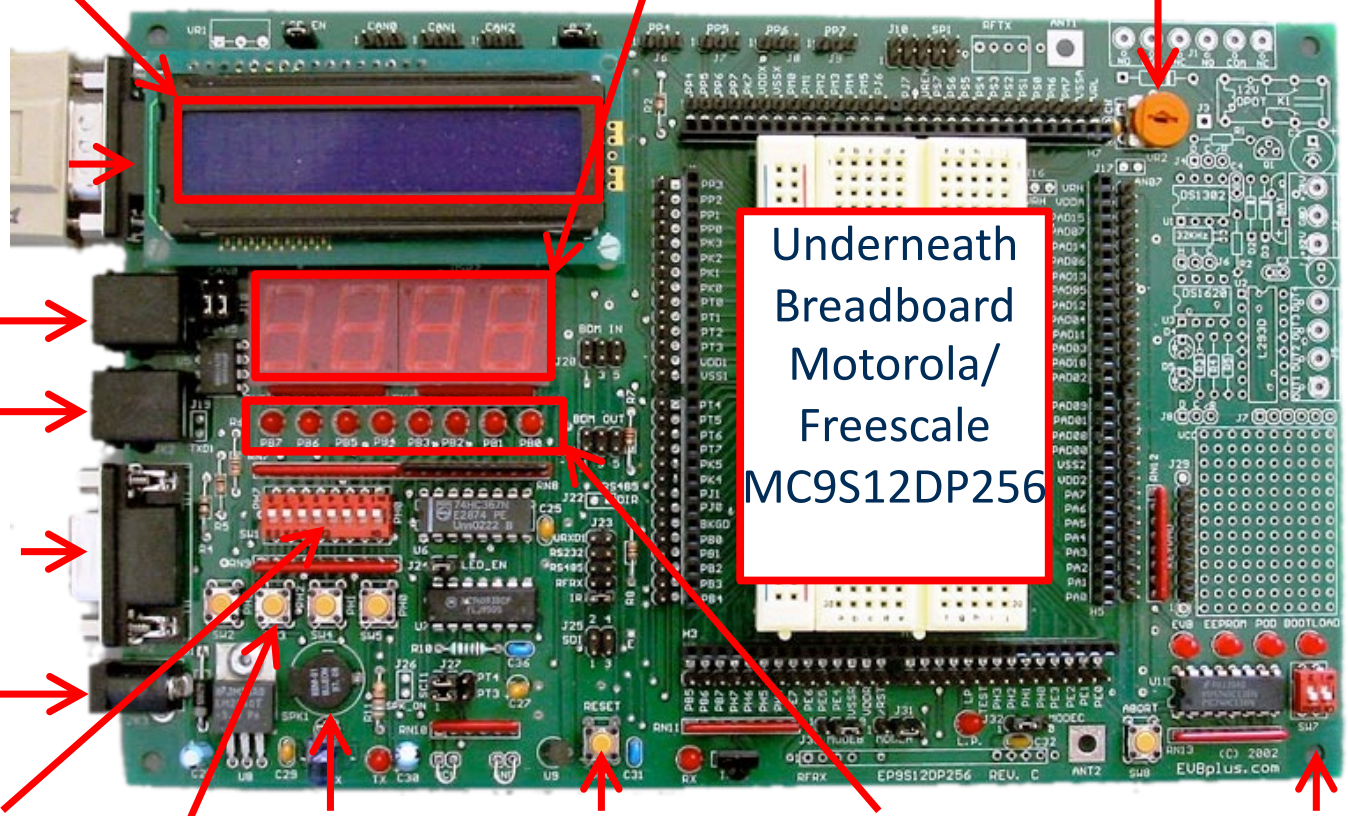
CONTENT

GOALS FOR TODAY

HOCHSCHULE ESSLINGEN

- Understand Digital I/O
- Understand Interrupts in C and ASM
- Being able to handle timers / timing

REPEAT: DRAGON-12 EVALUATION BOARD



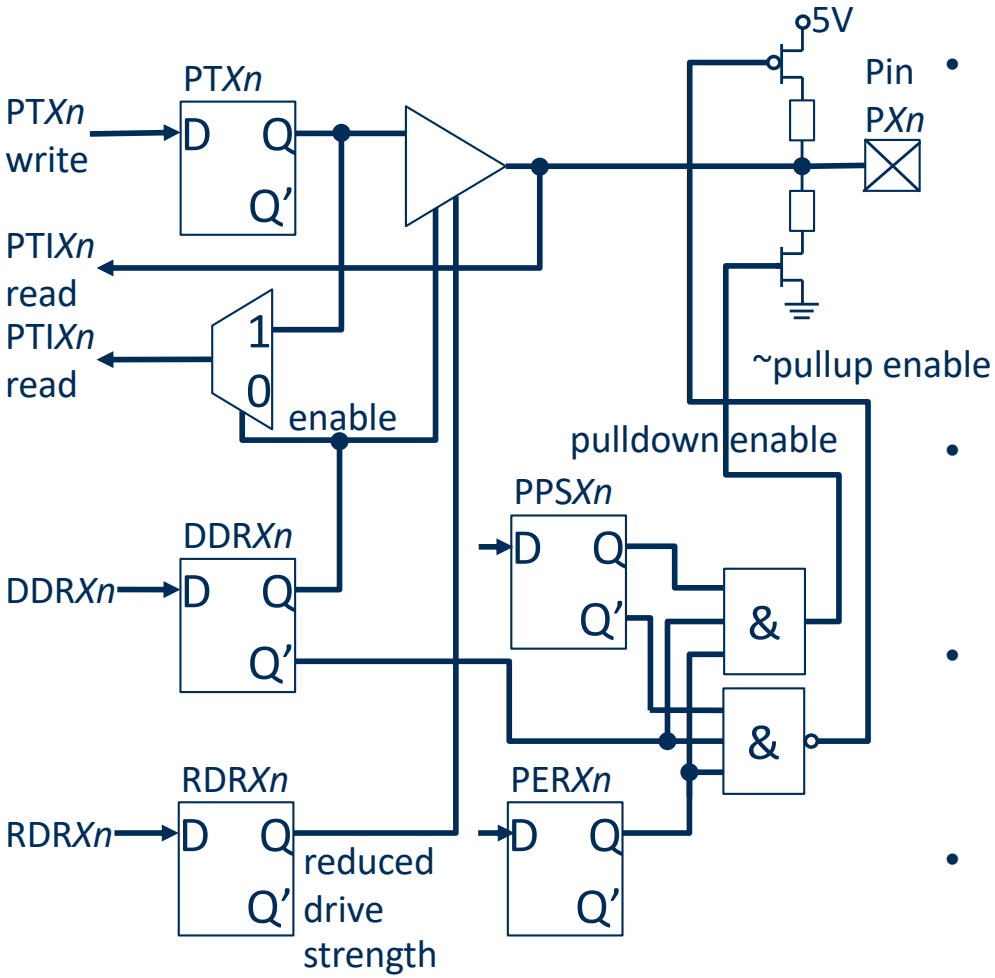
HOCHSCHULE ESSLINGEN

Digital Input and Output and Interrupts

DIGITAL INPUT AND OUTPUT 1/2

HOCHSCHULE ESSLINGEN

See 010-S12MEBIV3.pdf, & S12MEBIV3_Multiplexed_External_Bus_Interface_MEBI.pdf



Circuit diagram of a single digital-in-/output pin X.n
(DDR**X**.**n** = 1 → Pin **n** of Port **X** is configured as output with **n**=0…7 and **X**= A, B, E, H, J, K, M, P, S, T)

- The CPU has several groups of 8 digital input/outputs (ports). Reading and writing of port pins is done via data registers `PTx` or `PORTX` via `MOVB` instructions (8 bit) or via `BSET`/`BCLR` to write & `BRCLR`/`BRTST` to test single pins.
- Via `DDRx` (**D**ata **D**irection **R**egister) each Port's pins can be configured bitwise as input or output.
- Via `RDRx` (**R**educed **D**river **R**egister) the signal strength of the port pin can be reduced, to improve the EMC properties.
- Via `PPSx` (**P**ort **P**olarity **S**elect) a pull-up or pull-down resistor can be selected and with `PERx` (**P**ull **E**nable **R**egister) these resistors will be enabled.

DIGITAL INPUT AND OUTPUT

HOCHSCHULE ESSLINGEN

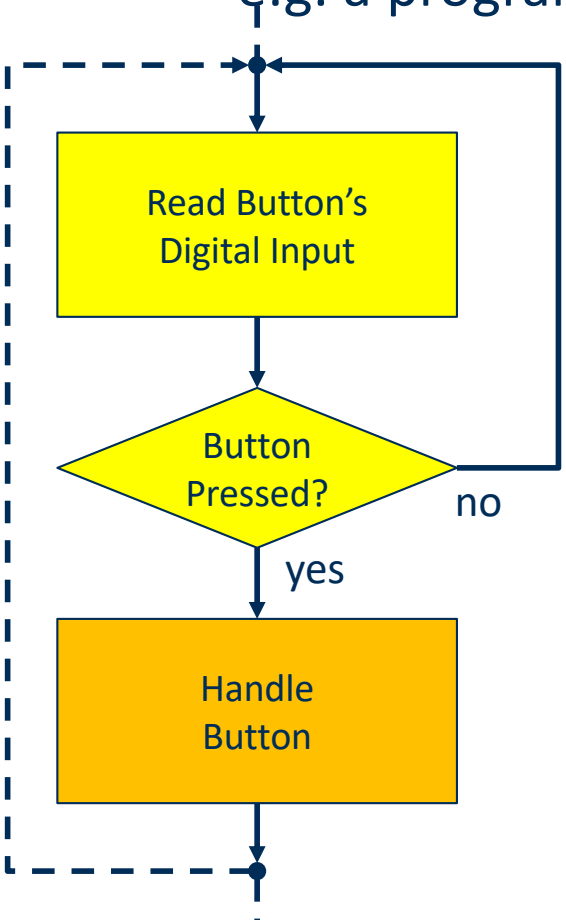
Most ports have alternative functions, e.g. Ports A and B may be used as digital inputs/outputs or external multiplex address/data bus. After Reset all ports are digital inputs. To use a port pin as output, configure it in the port's DDR register. If a port's alternative function is activated, the port (pin) can no longer be used as digital input/output:

Port	Data register/ Data direction	Notes/Restrictions	May trigger interrupt	Alternative Function
A	PORTA / DDRA	Pull-Up resistors and driver strength only for complete port, not bitwise (uses register PUCR rather than PPSx and PERx, register RDRIV instead of RDRx)	No	Multiplex address/data bus
B	PORTB / DDRB			
E	PORTE / DDRE	Like PORTA, pins 0 and 1 only as inputs	No	Control Signals for address data bus
K	PORTK / DDRK	Like PORTA		
H	PTH / DDRH		Yes	SPI interface
J	PTJ / DDRJ	Only 4 bit (J.0, 1, 6, 7) available	Yes	CAN or I^2C
M	PTM / DDRM		No	CAN interface
P	PTP / DDRP		Yes	PWM outputs SPI interface
S	PTS / DDRS		No	Serial interfaces SCI and SPI
T	PTT / DDRT		No	Timer input/output

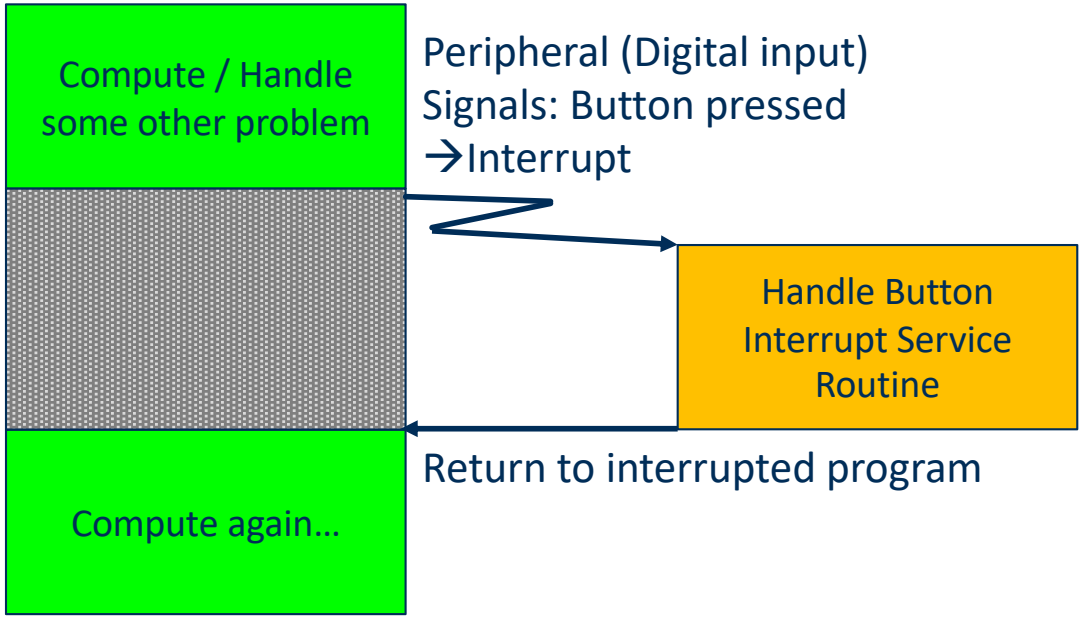
Ports H, J and P may generate edge-triggered interrupts (see chapter 3.2)
Ports are configured in the initialization phase of a program. Afterwards, ports are accessed via their data registers (see example BlinkingLED in chapter 2.2).
For port usage on Dragon12, see chapter 2.1 and Dragon12_getting_started_RevE.pdf.

INTERRUPTS 1/4

Problem: Synchronize a program to an external event,
e.g. a program, which reacts to a button press event.

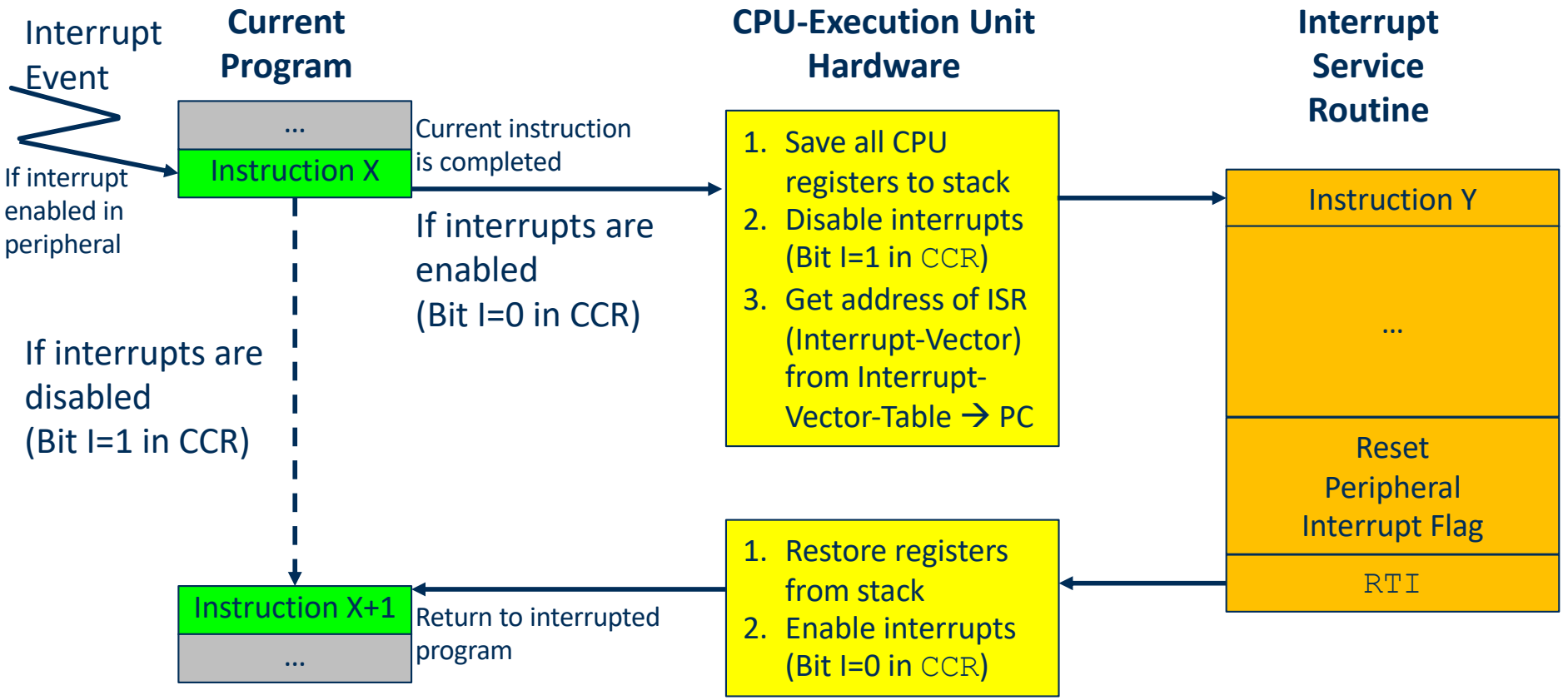


- Periodically reading the input signal (**polling**) in a loop



- Handling another problem (task) while waiting for the event
- Peripheral hardware signals the Interrupt to the CPU, when the button is pressed
- CPU interrupts the current task, handles the interrupt in an Interrupt-Service-Routine ISR and returns to the interrupted task again

INTERRUPTS 2/4



Requirements for an interrupt based program:

- Interrupt mask bit `I` in `CCR` must have been cleared (global interrupt enable)
- Interrupt vector table (see next page) must be an entry of the ISR address
- Peripheral must be configured to trigger an interrupt (peripheral Interrupt Enable IE)
- Interrupt Flag IF of the peripheral must be reset at the end of the ISR.

What kind of problems do You foresee?

INTERRUPTS 3/4

HOCHSCHULE ESSLINGEN

Interrupt-Vector-Table (selected only, for full list see `9S12DP256BDGV2_Device_User_Guide.pdf`, p.75)

No.	Address	Usage	Maskable
0	`$FFFE`	Reset	No *1
2	`$FFFA`	COP: Watchdog interrupt (**C**omputer **O**perating not **P**roperly?)	Yes
3	`$FFF8`	TRAP: Unimplemented opcode (Compiler bug? Stack clobbered?)	No *1
4	`$FFF6`	SWI: Software interrupt, called via instruction `SWI`	No *1
5	`$FFF4`	XIRQ: External, non-maskable interrupt request signal	No *2
6	`$FFF2`	IRQ: External interrupt request signal	Yes
7	`$FFF0`	**RTI**: **R**eal **T**ime **I**nterrupt	Yes
8	`$FFEE`	**Timer Channel 0**	Yes
15	`$FFE0`	**Timer Channel 7**	Yes
20	`$FFD6`	SCI0: First serial interface	Yes
21	`$FFD4`	SCI1: Second serial interface	Yes
22	`$FFD2`	**ATD0**: **A**nalog **T**o **D**igital convertor	Yes
25	`$FFCC`	**PTH**: Digital input port H	Yes
127	…	…	…

*1 “Non-maskable” interrupts are independent from interrupt mask bit I in CCR.

*2 Input XIRQ can be disabled via mask bit X in CCR.

Interrupt Sources

- CPU reset triggered by an external signal, the watchdog or clock generator monitoring. These events use the interrupt-vector-table, but are no real interrupts, because they re- start the CPU and do not return to the interrupted program.
- Hardware interrupts: Most peripherals can generate interrupts
- Software interrupt: Triggered by SWI instruction (used by debugger monitor program)
- Exceptions (traps): Triggered by some errors, e.g. trying to execute invalid opcode.

Interrupt Priorities

- If several interrupt events occur simultaneously, the ISRs are executed in order of entries in the interrupt vector table with lowest first, e.g. RTI before timer channel 0.
- Via register `HPRIO` the priority of one of the ISRs may be increased (not used in the lab). Other processors, e.g. x86, allow to change more priorities.
- If during an ISR other interrupt events shall be handled immediately, the interrupt mask can be cleared via the CLI instruction (nested-interrupts, not recommended)
- For each interrupt source, the latest interrupt event remains stored, until the interrupt flag in the associated peripheral is reset.

Communication between ISR and other parts of a program

- Interrupt service routines are called by hardware asynchronously to other parts of a pro- gram. So an ISR does not have any call and return parameters (with the exception of SWI). If data exchange is required, global variables must be used.

INTERRUPTS EXAMPLE IN C

HOCHSCHULE ESSLINGEN

Trigger an interrupt by pressing button SW5 on the Dragon12
(positive edge on CPU port H, bit 0) See `ButtonInterrupt.mcp`

```
void main (void) {
    EnableInterrupts;// Allow Interrupts
    DDRB  = 0xff;       // Port B.7..0 as outputs (LEDs)
    PORTB = 0x55;       // Turn on every other LED
    DDRH  = 0x00;       // Configure Port H.7..0 as inputs
    PPSH  = 0x01;       // trigger interrupt on pos. edge of H.0
    PIEH  = 0x01;       // enables interrupt for input port H.0
    for (;;) {}    // Endless loop
}
interrupt 25 void ButtonISR(void) {// ISR for int 25 (port H)
    PORTB = ~PORTB;     // Toggle LEDs on Port B
    PIFH  = 0x01;       // 1 resets the interrupt flag;
}
```

Important registers for port H (see `003-S12DP256PIMV2-Port Integration Module.pdf`, ch. 3.3.5)

`DDRH` … Data Direction Register

`PTH` … Data register

`PIEH` … Port Interrupt Enable register (1 = enable, configure bitwise)

`PPSH` … Port Polarity Select register (0 = negative edge, 1 = positive edge)

`PIFH` … Port Interrupt Flag register (1 = int. triggered, cleared by writing 1)

Note: In a practical application, buttons **must be debounced**. This should be done by polling rather than interrupts.

INTERRUPTS EXAMPLE IN ASSEMBLER

HOCHSCHULE ESSLINGEN

Requirements are the same as for C; See `ButtonInterruptAsm.mcp`

```
.vect:  SECTION             ; ROM: Interrupt vector section--------------
        ORG $FFCC
        DC.W isr25          ; Interrupt vector for interrupt 25 (Port H)
.init:  SECTION             ; ROM: Code section--------------------------
main:                       ; Begin of program
Entry:
    LDS #__SEG_END_SSTACK ; Initialize stack pointer
    CLI                     ; Enable interrupts
    …
    MOVB #$FF, DDRB     ; $FF -> DDRB: Port B.7…0 as outputs (LEDs)
    MOVB #$55, PORTB    ; $55 -> PORTB: Turn on every other LED
    MOVB #$00, DDRH     ; Configure port H.7…0 as inputs
    MOVB #$01, PPSH     ; trigger interrupt on positive edge
    MOVB #$01, PIEH     ; Enable interrupt for input port H.0
loop:
    BRA  loop           ; Endless loop
; Interrupt Service Routine (ISR) for interrupt 25
isr25:
    COM PORTB           ; Complement Port B: Toggle LEDs
    BSET PIFH, #1       ; Clear Interrupt Flag
    RTI                 ; Return from ISR (do not use opcode RTS)
```

Note: In a practical application, buttons **must be debounced**. This should be done by polling rather than interrupts.

INTERRUPTS CPU STATE AFTER RESET

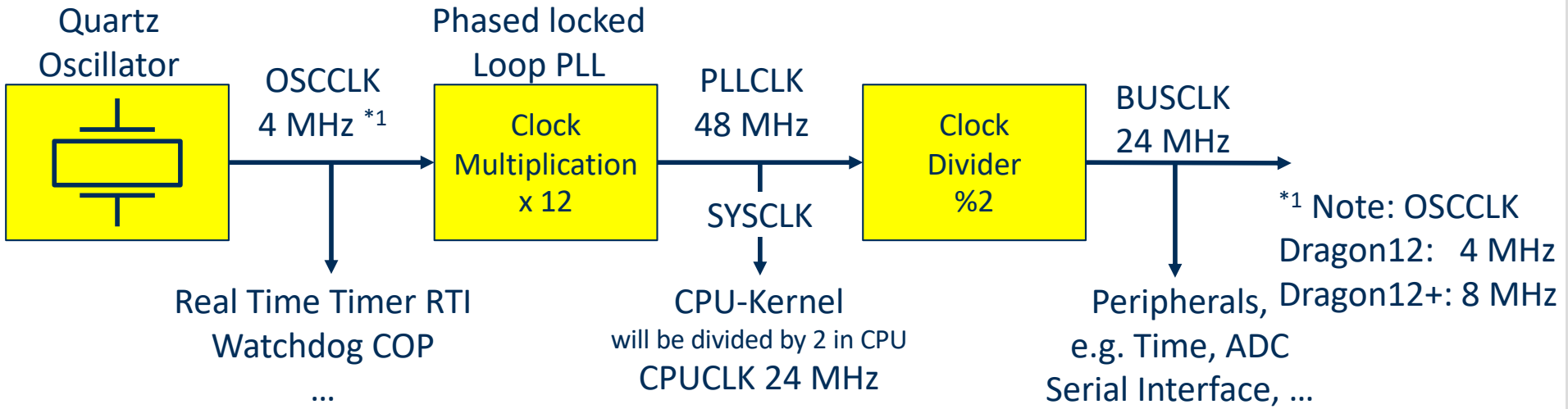
HOCHSCHULE ESSLINGEN

CPU State after Reset:

- `PC` loaded with reset interrupt vector (points to Dragon12 board's debugger monitor program, which inits the CPU's clock generator)
- `I=X=S=1` in `CCR` register, i.e. interrupts masked (disabled), STOP instruction disabled
- Other registers, e.g. `SP` (!) undefined
- Peripherals turned off, i.e. the clock generator's PLL, the watchdog (COP) or the Real Time Interrupt (RTI)
- All digital input/output pins configured as inputs

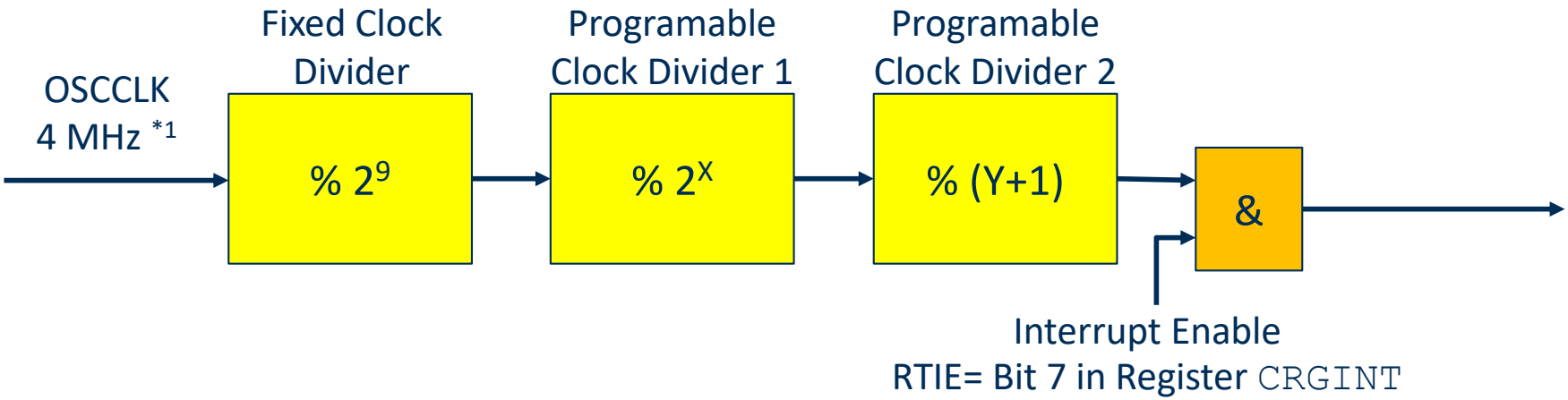
Clock Generator CRG and periodical Real Time Interrupts (RTI):

- Clock Signals (see Clock & Reset Generator Module CRG [3.4] & Appendix)
  The HCS12 has a complex programmable clock generator, configured by the debugger's monitoring program on the Dragon12 board as follows:



INTERRUPTS – REAL-TIME INTERRUPT

- Real Time Interrupts (RTI) (see Clock & Reset Generator Module CRG in [3.4])
  Periodical interrupts can be generated using the RTI unit:



The interrupt frequency is defined via clock divider settings X=1..7 and Y:

Register `RTICTL`

Bit	7	6	5	4	3	2	1	0
	0	X (3 bit)			Y (4 bit)			

$$f_{RTI} = \frac{f_{OSCCLK}}{2^{9+X} * (Y+1)}$$

Register `CRGINT`: To enable the RTI interrupt (RTI Enable, short RTIE), set bit 7 in `CRGINT` to 1. The **other bits** must not be changed!

Register `CRGFLG`: At the end of the ISR, the RTI Interrupt Flag (RTIF), i.e. bit 7 in `CRGFLG` must be reset by writing a 1. The other bits must not be changed!

INTERRUPTS – FREQUENCY RANGE OF RTI

The Frequency range of the RTI Interrupt

$$f_{RTI} = \frac{f_{OSCCLK}}{2^{9+X} * (Y+1)}$$ with X=1…7, Y=0…15 and f_{OSCCLK}=4MHz

Maximum:

Minimum:

APPENDIX B: CLOCK GENERATOR AND CLOCK DIVIDER

HOCHSCHULE ESSLINGEN

RTI Interrupt:

Interrupt period $T_{RTI} = 2^{9+X} * (Y + 1) / f_{OSCCLK}$. All values in ms @ f_{OSCCLK} = 4 MHz

	Y=0	Y=1	Y=2	Y=3	Y=4	Y=5	Y=6	Y=7	Y=8	Y=9	Y=10	Y=11	Y=12	Y=13	Y=14	Y=15
X=1:	0.256	0.512	0.768	1.024	1.280	1.536	1.792	2.048	2.304	2.560	2.816	3.072	3.324	3.584	3.840	4.096
X=2:	0.512	1.024	1.536	2.048	2.560	3.072	3.584	4.096	4.608	5.120	5.632	6.144	6.656	7.168	7.680	8.192
X=3:	1.024	2.048	3.072	4.096	5.120	6.144	7.168	8.192	9.216	10.240	11.264	12.288	13.312	14.336	15.360	16.384
X=4:	2.048	4.096	6.144	8.192	10.240	12.288	14.336	16.384	18.432	20.480	22.528	24.576	26.624	28.672	30.720	32.768
X=5:	4.096	8.192	12.288	16.384	20.480	24.576	28.672	32.768	36.864	40.960	45.056	49.152	53.248	57.344	51.440	65.536
X=6:	8.192	16.384	24.576	32.768	40.960	49.152	57.344	65.536	73.728	81.920	90.112	98.304	106.496	114.688	122.880	131.072
X=7:	16.384	32.768	49.152	65.356	81.920	98.304	114.68	131.072	147.456	163.840	180.224	196.608	212.992	229.376	245.760	262.144

ECT Timer: @ f_{BUSCLK} = 24 MHz

Clock period $T_{TCNT} = 1/f_{TCNT} = 2^X/f_{BUSCLK}$. In µs

X=0	X=1	X=2	X=3	X=4	X=5	X=6	X=7
0.042	0.083	0.167	0.333	0.667	1.333	2.667	5.333

Clock period $T_P = 2^{16} * TTC_{NT}$ Values in ms

X=0	X=1	X=2	X=3	X=4	X=5	X=6	X=7
2.731	5.461	10.293	21.845	43.691	87.381	174.763	349.525

PWM: @ f_{BUSCLK} = 24 MHz

Clock period of fast clock $T_{A/B} = 2^X/f_{BUSCLK}$. In µs

X=0	X=1	X=2	X=3	X=4	X=5	X=6	X=7
0.042	0.083	0.167	0.333	0.667	1.333	2.667	5.333

Clock period of slow clock $T_{SA/B} = 2*y*T_{A/B}$. In µs

	X=0	X=1	X=2	X=3	X=4	X=5	X=6	X=7
Y=1	0.083	0.167	0.333	0.667	1.333	2.667	5.333	10.667
…	…	…	…					
Y=255	21.250	42.500	85.000	170000	340000	680000	1360000	2720000

Serial Interface: @ f_{BUSCLK} = 24 MHz

Clock Divider Register SCIxBD = $f_{BUSCLK}/(16*f_{BIT})$

f_{BIT}=300 bit/s	600 bit/s	1,2 kbit/s	2,4kbit/s	4,8kbit/s	9,6kbit/s	19,2kbits	38,4kbit/s	57,6kbit/s	115,2kbit/s
5000	2500	1250	625	313	156	78	39	26	13

LECTURE: LITERATURE

HOCHSCHULE ESSLINGEN

According to list of literature:

- Patterson, D., Hennessy, J.: *Computer Organization and Design*, Kaufmann, 2011
  (Deutsche Übersetzung: Rechnerorganisation und –entwurf, Spektrum)
- Hennessy, J., Patterson, D.: *Computer Architecture: A quantitative Approach, **5th edition***, Morgan Kaufmann, 2017
- Tanenbaum, A., Austin, T.: *Structured Computer Organization*, Pearson, 6th edition, 2013
- Tanenbaum, A., Austin, T.: *Rechnerarchitektur: von der digitalen Logik zum Parallelrechner*, Pearson, 6th ed., 2014
- Huang, H.W.: *The HCS12/9S12. An Introduction to the HW and SW interface*, Thomson Learning, 2009
- Beierlein, T.: *Taschenbuch Mikroprozessortechnik*, Carl-Hanser Verl., 2011