Exam Cheat Sheet

Eduard Fekete

Character Count: 6903
Word Count: 1100

December 27, 2025

1 Boolean Algebra & Logic Gates

Task: Converting Truth Tables to Expressions and Reducing. **Notation Guide:**

- **NOT:** `!A` (Inverse)
- **AND:** `A * B` (Product)
- **OR:** `A + B` (Sum)
- **XOR:** `A # B` (Exclusive OR)

Truth Table → Expression:

- **Focus on the 1s:** Look ONLY at rows where Output is `T` (1).
- **Create Terms:** For each `T` row, write a product term.
  - If input is 0 → `!A`
  - If input is 1 → `A`
- **Combine:** `OR` (+) all terms together.
  - *Example:* Rows `0 1` and `1 0` are True → `(!A * B) + (A * !B)`.

Circuit → Expression:

- **Trace Left to Right:** Write the sub-expression at the output of *every* gate.
- **Watch for Bubbles:**
  - Bubble at Output = NOT the whole result (NAND/NOR).
  - Bubble at Input = NOT just that wire.

Reduction Rules (De Morgan's):

- `!(A * B) = !A + !B` (Break the bar, change the sign).
- `!(A + B) = !A * !B` (Break the bar, change the sign).
- `A + !A = 1`
- `A * !A = 0`

Combinatorial Logic - Full Adder:

- **Full Adder Inputs:** A, B, and Cin (Carry In).
- **Full Adder Outputs:** Sum and Cout (Carry Out).
- **Logic Check:** If exactly one input is 1, Sum=1. If two are 1, Cout=1. If all three are 1, Sum=1 AND Cout=1.

---

2 Binary Arithmetic & 2's Complement

Task: Negation, Addition, and Overflow. **Calculating 2's Complement (Negative Numbers):**

- **Step 1:** Invert all bits (0 → 1, 1 → 0).
- **Step 2:** Add 1.
- *Example (-111):* `111` is `0110 1111` → Invert `1001 0000` → Add 1 → `1001 1001`.

Detecting Overflow:

- **Rule:** Overflow only occurs when adding numbers of the **same sign** (Pos+Pos or Neg+Neg).
- **Detection:** If `Pos + Pos = Neg` result (MSB is 1) → Overflow.
- **Detection:** If `Neg + Neg = Pos` result (MSB is 0) → Overflow.
- *Note:* Adding a positive and a negative number NEVER causes overflow.

Hex Conversion:

- Group bits in 4s from right to left.
- Memorize: `10 = A`, `11 = B`, `12 = C`, `13 = D`, `14 = E`, `15 = F`.

---

3 The Stack (Atmega2560 Specifics)

Task: Calculating Stack Pointer (SP) value. **Critical Rule:** The stack grows **DOWN** (subtracts from address). **Operation Costs (Atmega2560):**

- **PUSH:** `-1` byte (SP = SP - 1)
- **POP:** `+1` byte (SP = SP + 1)
- **CALL / RCALL:** `-3` bytes (SP = SP - 3)
  - *Note:* Uses 3 bytes because the Program Counter is 22-bits (requires 3 bytes storage).
- **RET / RETI:** `+3` bytes (SP = SP + 3)

Strategy:

1. Start with the initial SP address (e.g., `0x1876`).
2. Walk through code line-by-line.
3. Sum the changes.
4. Subtract/Add total from initial SP.

---

4 Assembly Programming & Control Flow

Task: Writing code, finding bugs, calculating cycles. **Branching Logic:**

- **Signed Numbers (-10, +50):**
  - Use `BRLT` (Branch if Less Than).
  - Use `BRGE` (Branch if Greater/Equal).
- **Unsigned Numbers (0 to 255):**
  - Use `BRLO` (Branch if Lower).
  - Use `BRSH` (Branch if Same/Higher).
- **Equality:**
  - `BREQ` (Equal), `BRNE` (Not Equal).

16-Bit Math Bugs:

- **Carry Bug:** Using `ADD` for upper bytes instead of `ADC`.
  - *Fix:* Use `ADC` for MSB to include carry from LSB.
- **Pointer Bug:** For 16-bit addresses (X, Y, Z registers), remember that `Z` is `R31:R30`.
  - *Load constant to Z:* `LDI ZH, high(label*2)` and `LDI ZL, low(label*2)`.

Instruction Cycles & Delays:

- **LDI:** 1 cycle.
- **ADD/ADC/SUB:** 1 cycle.
- **DEC:** 1 cycle.
- **BRNE/BREQ:** 1 cycle if false, 2 cycles if true (jumping).
- **CALL:** 4 cycles.
- **RET:** 4 cycles.
- **Loop Timing Calculation:** Total = `LDI` + [Iterations * (`DEC` + `BRNE`)].
  - *Example:* `LDI R20, 10` (1) + 10 * [`DEC` (1) + `BRNE` (2)] = 31 cycles.

I/O Registers:

- **DDRx:** Direction (1=Out, 0=In).
- **PORTx:** Output data (if Out) or Pull-up (if In).
- **PINx:** Read Input data.

Bit Manipulation & Masking:

- **Isolate Bit 6:** `IN R16, PINB` followed by `ANDI R16, 0b01000000`.

- **Toggle Logic:** If `BREQ` follows a mask, it branches if the bit was 0.
- **Skip Instructions:**
  - **SBIC:** Skip next instruction if Bit is Cleared (0).
  - **SBIS:** Skip next instruction if Bit is Set (1).
  - **SBI / CBI:** Set Bit / Clear Bit immediately in I/O register.

---

5 Interrupts (ISR) & External Events

Task: Setup and safety.

Vector Table Setup:

- `.org 0x0000` → `JMP start`
- `.org 0x0002` → `JMP into_isr` (External Interrupt 0).

The Golden Rules of ISRs:

1. **Must Save SREG:** Status flags must be preserved.
   - *Code:* `PUSH R16` → `IN R16, SREG` → `PUSH R16`.
2. **Debouncing in Software:** Add a small delay (e.g., `CALL cao_delay_r16`) inside the ISR or main loop to filter physical button "bounce."
3. **Restore SREG:** `POP R16` → `OUT SREG, R16` → `POP R16`.
4. **End with RETI:** Return from Interrupt.

---

6 EEPROM Operations

Task: Storing non-volatile data.

The "Busy Wait" Pattern:

- Before reading or writing, check if the EEPROM is busy:
  - `sbic EECR, EEPE`
  - `rjmp ...` (loop until clear).

Writing Sequence:

1. Wait for `EEPE` to become 0.
2. Set Address: `OUT EEARH, rH` and `OUT EEARL, rL`.
3. Set Data: `OUT EEDR, rData`.
4. Set Master Write Enable: `SBI EECR, EEMPE`.
5. Set Write Enable (within 4 cycles): `SBI EECR, EEPE`.

---

7 Sequential Logic (Flip-Flops)

Task: Reading Timing Diagrams.

Behavior:

- **Trigger:** D Flip-Flops trigger on the **Rising Edge** (Low → High) of the Clock.
- **Frequency Division:** Connecting `Q-not` back to `D` divides the frequency by 2. Daisy-chaining three of these divides by 8 (2^3).
- **Reading the Graph:**
  1. Draw vertical lines where Clock goes UP.

2. Check `D` state exactly at that line.
3. `Q` (Output) immediately changes to match `D`.
4. `Q` stays flat (latched) until the next rising edge.

---

8 Atmega2560 Architecture Facts

True/False Memorization:

- **CPU:** 8-bit processor.
- **Flash Memory:** 256 KB.
- **SRAM (Data):** 8 KB.
- **EEPROM:** 4 KB.
- **Register Location:** Registers are **Inside** the CPU (Statements saying they are inside the ALU or outside CPU are **FALSE**).
- **Memory Location:** SRAM and Flash are **Inside the chip** but **Outside the CPU** (True).
- **ALU:** Used for calculations (Add, Sub, Mul, Logic), but **cannot** perform division.
- **EEPROM Access:** Not directly mapped in data memory (requires I/O registers to read/write).
- **I/O Access:** Mapped in data memory space (True).