Building Interactive Devices and Objects

Prof. Dr. Michael Rohs, Dipl.-Inform. Sven Kratz
michael.rohs@ifi.lmu.de
MHCI Lab, LMU München
Books


• Alexander Bierbaum, Alexander Piaseczki, Joachim Schröder, Pedram Azad, Tilo Gockel, Rüdiger Dillmann: Embedded Robotics - Das Praxisbuch, Elektor-Verlag, 2005
## Schedule

<table>
<thead>
<tr>
<th>#</th>
<th>Date</th>
<th>Topic</th>
<th>Group Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19.4.2012</td>
<td>Session 1: Introduction</td>
<td>Team building</td>
</tr>
<tr>
<td>2</td>
<td>26.4.2012</td>
<td>Session 2: Microcontrollers &amp; Electronics</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3.5.2012</td>
<td>Session 3: Sensors</td>
<td>Concept development</td>
</tr>
<tr>
<td>4</td>
<td>10.5.2012</td>
<td>CHI</td>
<td>Concept development</td>
</tr>
<tr>
<td>5</td>
<td>17.5.2012</td>
<td>Christi Himmelfahrt</td>
<td>Concept development</td>
</tr>
<tr>
<td>6</td>
<td>24.5.2012</td>
<td>Session 4: Actuators</td>
<td>Concept presentation, Hardware requ.</td>
</tr>
<tr>
<td>7</td>
<td>31.5.2012</td>
<td>Session 5: Physical Objects (Sven)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7.6.2012</td>
<td>Frohnleichtnam</td>
<td>Project</td>
</tr>
<tr>
<td>9</td>
<td>14.6.2012</td>
<td></td>
<td>Project</td>
</tr>
<tr>
<td>10</td>
<td>21.6.2012</td>
<td></td>
<td>Project</td>
</tr>
<tr>
<td>11</td>
<td>28.6.2012</td>
<td></td>
<td>Project</td>
</tr>
<tr>
<td>12</td>
<td>5.7.2012</td>
<td></td>
<td>Project</td>
</tr>
<tr>
<td>13</td>
<td>12.7.2012</td>
<td></td>
<td>Evaluation</td>
</tr>
<tr>
<td>14</td>
<td>19.7.2012</td>
<td></td>
<td>Evaluation, Presentation</td>
</tr>
</tbody>
</table>
Sessions 2: Microcontrollers & Electronics

- AVR microcontrollers, LEDs, buttons, transistors

- Exercises
  1. Hello world (LED blinking)
  2. Button debouncing, switching LED
  3. Controlling multiple LEDs
Today

• AVR Eclipse Plugin
• Configuring and uploading
• Microcontrollers
• LEDs and buttons
• Exercise 2
AVR ECLIPSE PLUGIN
AVR Development Toolchain & IDEs

• Free AVR-GCC toolchain
  – GNU C compiler + linker: avr-gcc (gcc.gnu.org)
  – C library: avr-libc (www.nongnu.org/avr-libc/)
  – Down-/uploader: avrdude (www.nongnu.org/avr-libc/)

• Eclipse (cross platform)

• CrossPack for Mac OS X

• WinAVR for Windows

• Atmel AVR Studio
  – http://www.atmel.com
Creating a Project with AVR Eclipse Plugin

- Need to install AVR-GCC toolchain separately
- Documentation
- Create project
  - File | New | Project… |
    C/C++ | C Project |
    AVR Cross Target Application
- AVR Target Hardware
  - MCU Type: ATtiny45
  - MCU Frequency (Hz): 1000000
New C Source File

Create a new source file.

Source folder: HelloAvr
Source file: main.c
Template: Default C source template

Finish
“µC Hello World”: Blinking LED

#include <avr/io.h>
#include <util/delay.h>

int main() {
    DDRB = 0b010000;
    while (1) {
        PORTB = 0b010000;
        _delay_ms(500);
        PORTB = 0b000000;
        _delay_ms(500);
    }
    return 0;
}
Development Process

Challenge:
Matching hardware and software versions during development process

Source: Gadre, Malhotra: tinyAVR projects
Build Settings

- uncheck “build automatically”

- check “save automatically”
Build Settings

- uncheck EEPROM file
- uncheck Assembler listing
Building the Project

• Right-click project name, “Build Project”

• Console output

```
**** Build of configuration Release for project HelloAvr2 ****

make all
Building file: ../main.c
Invoking: AVR Compiler
Finished building: ../main.c

Building target: HelloAvr2.elf
Invoking: AVR C Linker
avr-gcc -Wl,-Map,HelloAvr2.map -mmcu=attiny45 -o "HelloAvr2.elf" ./.main.o
Finished building target: HelloAvr2.elf

Create Flash image (ihex format)
avr-objcopy -R .eeprom -O hex HelloAvr2.elf "HelloAvr2.hex"
Finished building: HelloAvr2.hex

Invoking: Print Size
avr-size --format=avr --mcu=attiny45 HelloAvr2.elf
AVR Memory Usage
-------------
Device: attiny45

Program: 146 bytes (3.6% Full)
(.text + .data + .bootloader)

Data: 0 bytes (0.0% Full)
(.data + .bss + .noinit)

Finished building: sizedummy

**** Build Finished ****
```

AVR Compiler
avr-gcc main.c → main.o

AVR Linker
avr-gcc main.o → HelloAvr2.elf

Create flash image
avr-objcopy HelloAvr2.elf → HelloAvr2.hex

Print memory usage
avr-size
Build Result

• Generated files
  – Debug folder
  – Release folder

• Build configurations
  – Switch to “release configuration” to install on hardware
AVR-GCC Toolchain Overview

- User’s input files
- GCC
- GNU Binutils
- AVR Libc
- GDB / AVaRICE / Simulavr
- AVRDUDE

Assembly Language

• ATtiny, Atmega
  – simple instruction sets
  – for example: ATtiny13 has 120 instructions
  – reasonably simple to program

• http://avra.sourceforge.net/index.html
CONFIGURING AND UPLOADING
Uploading the Program to the µC

- Serial programming via Serial Peripheral Interface (SPI)
  - MISO, MOSI, SCK
USB Drivers for “mySmartUSB light”

• USB chip CP2102 from Silicon Laboratories

• Windows
  http://shop.myavr.ch/index.php?sp=article.sp.php&artID=200006
  background information, command set, etc.

• Mac OS X, Linux
  VCP Driver Kit
  List serial devices: ls /dev/tty.*
  Java Serial: http://rxtx.qbang.org
Programmer Configuration
Programmer Configuration

Programmer:
Atmel STK500 Version 2.x firmware

Override default port:
/dev/tty.SLAB_USBtoUART
Programming the μC

• Tasks
  – Download/upload program code to/from Flash memory
  – Download/upload data to/from internal EEPROM
  – Configuring the microcontroller (“fuse bits”)

• Programming options
  – Serial programming
    • In-system programming (ISP)
    • High-voltage serial programming (HVSP, only 8-pin controllers)
  – High-voltage parallel programming
    • If RESET pin used as I/O pin: high-voltage programming
  – debugWire on-chip debug system
    • Uses RESET pin for debugging and Flash/EEPROM programming
Configuring the μC

• Configuring the μC = setting “fuse bits”
  – Clock rate, programmability, low-voltage detection, etc.
  – Caution: Wrong fuse bit settings may render chip unusable!

• Only needed once (e.g. when clock rate changes)
• “Fuse bits” described in datasheet
• Simpler with calculators / tools
  – Online fuse calculator
    • http://www.engbedded.com/fusecalc/
  – AVRFuses tool
    • http://www.vonnieda.org/software/avrfuses
Online Fuse Calculator
http://www.engbedded.com/fusecalc/

Device selection
Select the AVR device type you want to configure. When changing this setting, default fuse settings will automatically be applied. Presets (hexadecimal representation of the fuse settings) can be reviewed and even be set in the last form at the bottom of this page.

AVR part name: ATtiny45 Select (141 parts currently listed)

Feature configuration
This allows easy configuration of your AVR device. All changes will be applied instantly.

Features
- Int. RC Osc. 8 MHz; Start-up time PWRDWN/RESET: 6 CK/14 CK + 64 ms; [CKSEL=0010 SUT=10]; default value
- Clock output on PORTB4; [CKOUT=0]
- Divide clock by 8 internally; [CKDIV8=0]
- Brown-out detection disabled; [BODLEVEL=111]
- Preserve EEPROM memory through the Chip Erase cycle; [EESAVE=0]
- Watch-dog Timer always on; [WDTON=0]
- Serial program downloading (SPI) enabled; [SPIEN=0]
- Debug Wire enable; [DWEN=0]
- Reset Disabled (Enable PB5 as i/o pin); [RSTDISBL=0]
- Self Programming enable; [SELFPRGEN=0]

Low
0x 62

High
0x DF

Extended
0x FF
AVR Clock Options

• Clock frequency can be chosen
  – Application requirements, power consumption
  – Clock prescaler register (divide clock by factor)
  – Component clocks can be disabled to reduce power consumption

• Clock source can be chosen
  – Internal resistor capacitor (RC) oscillator
    • Convenient, but not precise (temperature, operating voltage)
    • ATtiny13: 4.8MHz, 9.6MHz (at 3V and 25°C), 128kHz (low power)
  – External crystal oscillator
    • Highly precise, requires external quartz

• Clock source distributed to modules
  – $\text{CLK}_{\text{CPU}}$, $\text{CLK}_{\text{I/O}}$, $\text{CLK}_{\text{flash}}$, $\text{CLK}_{\text{ADC}}$
  – $\text{CLK}_{\text{ADC}}$ allows switching off other clocks during ADC conversion
AVRFuses Tool (optional)

Caution: Wrong fuse bit settings may render chip unusable!
Tool: AVRFuses (www.vonnieda.org/AVRFuses/)
Configuring AVRFuses for the Programmer and USB Port

**mySmartUSB light:**

- Path to avrdude: `avrdude`  
- Programmer: `stk500v2`  
- Port: `/dev/cu.SLAB_USBtoUART`  
- Baud Rate: [Default]

**USBasp:**

- Path to avrdude: `/usr/local/CrossPack`  
- Programmer: `usbasp`  
- Port: `usb`  
- Baud Rate: [Default]

Additional resources:

- [http://www.fischl.de/usbasp/](http://www.fischl.de/usbasp/)
• Fuses show factory configuration of ATtiny13
  – reset when Vcc below level
• Brown-out detection
  – use reset pin as I/O pin: dangerous!
• Reset disabled
  – use reset pin as I/O pin: dangerous!
• Start-up time
  – delay until conditions are stable
Setting Fuses with Eclipse: ATtiny45

for ATtiny45:
AVRDude Settings

- **Uploader configuration**
  - Programmer
  - Flash / EEPROM
  - Fuses
  - Lockbits
  - Advanced
  - Other

- **Upload Flash Memory Image**
  - **Do not upload flash memory image**
  - **From build**

- **Upload EEPROM Image**
  - **Do not upload eeprom image**

- **Upload Lockbits**
  - **Do not set lockbits**
Uploading Program with Eclipse

- Console output

Launching /usr/local/CrossPack-AVR/bin/avrdude -p t45 -cusbasp -Uflash:w:HelloAvr2.hex:a

Output:

avrdude: AVR device initialized and ready to accept instructions
Reading | #-----------------------------------------------------| 100% 0.02s
avrdude: Device signature = 0x1e9206
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "HelloAvr2.hex"
avrdude: input file HelloAvr2.hex auto detected as Intel Hex
avrdude: writing flash (146 bytes):
Writing | #-----------------------------------------------------| 100% 1.20s
avrdude: 146 bytes of flash written
avrdude: verifying flash memory against HelloAvr2.hex:
...
Reading | #-----------------------------------------------------| 100% 0.82s
avrdude: verifying ...
avrdude: 146 bytes of flash verified
avrdude done. Thank you.
avrdude finished
AVR Eclipse Plugin – Advanced Settings

• Target Hardware: Specify target microcontroller
  – MCU Type: ATtiny45 (later will also use ATmega8)
  – MCU Clock Frequency: typical values are 1 MHz (internal), 8 MHz (external quartz), 16 MHz (external quartz)

• AVRDude: Install program on microcontroller
  – Programmer: Atmel STK500 Version 2.x firmware
  – Override default port: /dev/tty.usbserial-A100OXPZ
Command Line Without Eclipse

• Compiling
  – avr-gcc -Os -mmcu=attiny45 main.c

• Format conversion
  – avr-objcopy -R .eeprom -O ihex a.out a.hex

• Uploading
  – avrdude -pt45 -cstk500v2 -P/dev/ttyUSB0 -Uflash:w:a.hex:a

Source: Mikołaj Dądela
MICROCONTROLLERS
Microcontrollers

• Integrates processor, memory, I/O peripherals, and sensors on a single chip
  – Replaces many traditional hardware components in a single chip
  – Lower cost, fewer additional components, smaller circuit board
  – Very memory efficient (sleep modes)
  – Software flexibility through software

• Memory types
  – Flash: program
  – RAM: working memory (stack, heap)
  – EEPROM: non-volatile memory

• Interrupt-driven I/O
  – Sources: signal changes, timer overflow, ADC conversion done
  – Interrupts can wake microcontroller from low-power sleep state
Microcontrollers

Source: Gadre, Malhotra: tinyAVR projects
Microcontrollers

• I/O Pins
  – Used as input or output (controlled by software)
  – Serial communications (UART, I²C, SPI)
  – Signal generation (PWM, timers)
  – Analog input (ADC conversion)

• Development
  – In-circuit programming and debugging, field update of firmware
  – Programming in assembly language or C

• Selectable clock frequencies
  – Lower clock rate \(\rightarrow\) less energy

• No floating point unit (typically)
Atmel AVR: ATtiny, ATmega

• 8-bit RISC chip, Harvard architecture

• ATtiny
  1–8 kB program memory
  6–32-pin package

• ATmega
  4–256 kB program memory
  28–100-pin package
  Extended instruction set
    • Multiply instructions
    • Handling larger program memories

• Large family of devices, specific features
Many types of AVRs: Choose depending on required features

**ATtiny13**
- 6 I/O pins, 1.8-5.5V operation
- 20 MPIS @ 20 MHz (clock rate selectable), internal oscillator
- 64B RAM, 64B EEPROM, 1kB Flash program memory
- 8-bit timer, 2 PWM channels, 10-bit ADC, analog comparator
- Price: €1.15

**ATtiny45**
- 6 I/O pins, 1.8-5.5V operation
- 20 MPIS @ 20 MHz (clock rate selectable), internal oscillator
- 256B RAM, 256B EEPROM, 4kB Flash program memory
- 2 8-bit timers, 4 PWM channels, 10-bit ADC, analog comparator, SPI, TWI, temperature sensor
- Price: €2.05
Many types of AVRs: Choose depending on required features

ATmega8
- 23 I/O pins, 2.7-5.5V operation
- 16 MIPS @ 16 MHz (clock rate selectable), internal oscillator
- 1kB RAM, 512B EEPROM, 8kB Flash program memory
- 2 8-bit timers, 1 16-bit timer, 3 PWM channels, 10-bit ADC, analog cmp., SPI, TWI, USART
- Price: €2.60

ATmega328P
- 23 I/O pins, 1.8-5.5V operation
- 20 MIPS @ 20 MHz (clock rate selectable), internal oscillator
- 2kB RAM, 1kB EEPROM, 4kB Flash program memory
- 2 8-bit timers, 1 16-bit timer, 6 PWM channels, 10-bit ADC, analog cmp., SPI, TWI, USART, temperature sensor
- Price: €3.30
ATtiny45 Data Sheet

- 8 pins, 236 pages datasheet!
Pinout ATtiny45

• Multiplexed pin functions, software configurable
  – Example: Flash/EEPROM programming via SPI:
    MOSI = master out, slave in (from programmer to ATtiny)
    MISO = master in, slave out (from ATtiny to programmer)
    SCK = serial clock
  – Example: ADC1 = ADC input channel 1
  – Example: PCINT3 = pin change interrupt 3
Pinout ATmega8

Source: Atmel data sheet
AVR Memory Layout

Program Memory

0x0000

internal Flash
word size = 16 bits

end

Data Memory

32 registers

64 I/O registers

internal SRAM
word size = 8 bits

Data Storage

internal EEPROM
word size = 8 bits

end

0x0000

0x0020

0x0060
AVR I/O Ports

• I/O pin either input or output
  – Individually software-controlled

• Pin as output
  – States: low, high
  – Can drive 40mA (→ LED)

• Pin as input
  – Internal pull-up resistor (enabled/disabled in software)
  – high resistance state (high-Z) if pull-up disabled
Accessing the I/O Ports

- Three memory addresses for each I/O port
  - **Data Direction Register:** DDRx
    - 1 = output
    - 0 = input
  - **Data Register:** PORTx
    - if input: 1 = pull-up enabled, 0 = pull-up disabled
    - if output: 1 = PIN driven high, 0 = PIN driven low
  - **Port Input Pins:** PINx
    - read: PIN state (independent of DDRx)
    - write 1: toggles PORTx
### AVR I/O Ports: Pin Control Example

<table>
<thead>
<tr>
<th>PIN</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>in/out</td>
<td>out</td>
<td>out</td>
<td>out</td>
<td>out</td>
<td>in</td>
<td>in</td>
<td>in</td>
<td>in</td>
</tr>
<tr>
<td>value</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>pullup</td>
<td>hi-z</td>
<td>hi-z</td>
<td>hi-z</td>
</tr>
</tbody>
</table>

#### Assembly

```assembly
ldi r16, (1<<PB4) | (1<<PB1) | (1<<PB0)
ldi r17, (1<<DDB3) | (1<<DDB2) | (1<<DDB1) | (1<<DDB0)
out PORTB,r16
out DDRB,r17
nop    // synchronization
in r16,PINB
```

#### C

```c
unsigned char i;
PORTB = (1<<PB4) | (1<<PB1) | (1<<PB0);
DDRB = (1<<DDB3) | (1<<DDB2) | (1<<DDB1) | (1<<DDB0);
__no_operation(); // synchronization
i = PINB;
```
External Clock: Quartz Crystal Oscillators

- More precise than internal oscillators
- Quartz 1..20 MHz
- Ceramic capacitors 12-22pF
- Place quartz and capacitors close to AVR pins
- Change CLKSEL fuse bits
Stabilizing and Decoupling Capacitors (Stütz- und Abblokkondensatoren)

• Sudden fluctuation in current caused by
  – switching on/off LEDs, motors, relays causes
  – changing state of AVR pins

• Power supply alone cannot compensate for these

• Solution: stabilizing capacitors between VCC and ground
  – no current flow through them after charging (if voltage stable)
  – local energy source
  – filter spikes

• Higher frequency ripple requires smaller capacitor
Stabilizing and Decoupling Capacitors (Stütz- und Abblokkondensatoren)

- Larger capacitors (10µF..100µF) as a local energy source
  - Electrolytic, high capacitance, high leakage, not suited for high frequencies, polar (!)

- Smaller capacitors (10nF..100nF) for filtering spikes
  - Ceramic, low capacitances, suited for high frequencies

- Place capacitors between GND and VCC of ICs
  - Place close to IC pins

\[
\begin{align*}
C &= 0.1\mu F \\
\text{with capacitor:} & \quad \begin{array}{c}
5.00V \\
4.99V
\end{array} \\
\text{without capacitor:} & \quad \begin{array}{c}
5.00V \\
4.99V
\end{array}
\end{align*}
\]
AVR ATtiny45 Architecture

Source: Datasheet
AVR ATtiny45
Architecture
LEDS & BUTTONS
LEDs

- Quickly switchable, power-efficient light sources
  - different types covering different parts of the visible spectrum (and beyond: IR LEDs, UV LEDs)
- Anode (long lead) goes to positive potential
- Cathode (short lead) goes to negative potential
- LEDs operate like voltage-controlled switches
  - little current below turn-on voltage (silicon: 0.7V)
  - very high current above → LEDs need current-limiting resistors
- LEDs are diodes: no current in reverse direction
- Typical forward current: 20mA, typical forward voltage 2V
LEDs

- Intensity of light proportional to current
  - can also use PWM to control brightness
  - light covers narrow spectrum only, except for white LEDs

- Forward voltage drop depends on color, e.g.:

<table>
<thead>
<tr>
<th>Color</th>
<th>Fwd. current</th>
<th>Fwd. voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>20mA</td>
<td>2.0V</td>
</tr>
<tr>
<td>green</td>
<td>20mA</td>
<td>3.5V</td>
</tr>
<tr>
<td>blue</td>
<td>20mA</td>
<td>3.7V</td>
</tr>
<tr>
<td>white</td>
<td>20mA</td>
<td>3.5V</td>
</tr>
</tbody>
</table>

- Can go up to 100mA (peak current)
Example: Blue LED Voltage Drop

- $U_f = 3.2\,\text{V}, I_f = 20\,\text{mA}$
- Current limiting resistor: $U_R = U - U_f = 5 - 3.2 = 1.8\,\text{V}$
  $R = \frac{U_R}{I} = \frac{1.8\,\text{V}}{0.020\,\text{A}} = 90\,\Omega$

\[ R_f = \frac{U_f}{I} \]
LED Displays: 7-Segment, 10-Bar

- 7-segment display (green)
  - $U_f = 2.2V$, $I_f = 20mA$

- 10-bar display (red)
  - $U_f = 2.0V$, $I_f = 20mA$
LED Displays: 5x7-Matrix

- 5x7 dot matrix display (red)
  - \( U_f = 2.25\text{V}, \; I_f = 20\text{mA} \)

- Check total power consumption
  - ATtiny can only drive up to 40mA per pin
  - use transistor if necessary
More LEDs than μC Pins

- 8-bit serial-in, parallel-out shift register 74LS164N

- $Q_A = A$ and $B$, unused input must be $H$
- CLK: low-to-high shifts data one place right
- maximum clock frequency: 25 MHz
- $V_{cc} = 5V$, $I_{CC} = 16mA$
- $I_{OS} = -10..-27.5mA$ (short-circuit output current)
Multiplexing LEDs

Charlieplexing LEDs

- Enables one LED at a time
  - N LEDs, each only on $1/N^{th}$ of the time
- $Z = \text{tri-state (high impedance state, } \text{“no” current)}$

<table>
<thead>
<tr>
<th>LED</th>
<th>Pin1</th>
<th>Pin2</th>
<th>Pin3</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>1</td>
<td>0</td>
<td>Z</td>
</tr>
<tr>
<td>D3</td>
<td>Z</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D6</td>
<td>1</td>
<td>Z</td>
<td>0</td>
</tr>
<tr>
<td>D4</td>
<td>Z</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D2</td>
<td>0</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>D5</td>
<td>0</td>
<td>Z</td>
<td>1</td>
</tr>
</tbody>
</table>
Button De-Bouncing

• Activate pull-up resistor on pin
  – Pull-up puts pin into defined state
  – (see previous slides on pin configurations)

• Connect button to GND
  – Pin will be high until button pressed

• De-Bouncing
  – Button contacts bounce, which generates many spikes
  – Hardware solutions: SR latch, capacitor
  – Software solution:
    – wait for 10-20ms after first event

Source: Wikipedia, Author: Tomoldbury, public domain
74HC138: 3-to-8 Line Decoder/Demultiplexer

- Input \( x = (\text{not}(E1) \land \text{not}(E2) \land E3) \)
- Address lines \( A0, A1, A2 \) to select output \( Y_{A0,A1,A2} \)
- Output \( Y_{A0,A1,A2} = \text{not}(x) \)
- Other outputs: \( Y_i = 1 \)
- \( V_{cc} = 5V \)
- \( I_{OUT} = \pm 25mA \)

Source: Philips Datasheet
74HC154: 4-to-16 Line Decoder/Demultiplexer

- Input $x = (E1 \text{ or } E2)$
- Address lines $A0, A1, A2$ to select output $Y_{A0,A1,A2}$
- Output $Y_{A0,A1,A2} = x$
- other outputs: $Y_i = 1$
- $V_{cc} = 5V$
- $I_{OUT} = \pm 50mA$

Source: Harris Semiconductor Datasheet