ENIAC-Demonstrator

Rainer Glaschick, Paderborn
2014-07-02

Introduction

World's first electronic computer for daily use was the ENIAC. It had only 20 accumulators which were the only read-write memory, roughly equivalent to 200 bytes, and was programmed by plugging cables, so the user did the microprogamming in today's terms. While programming by plugging was never abondoned for analog computers (and probably was their fate), no successor machine had been built, as programming using commands in memory was clearly superior.

To explain the way the ENIAC worked is not easy, in particular as we nowadays think in terms of sequential programs. So in order to have a real object to demonstrate the way the ENIAC was used, a demonstrator is build in hardware to show the basics of programming the ENIAC.

There are several simulators for the ENIAC; easily found are three of them: Zoppke2004, Hansen2003 and Riley2013. All try to model the ENIAC rather completely; and they helped much with this project.

On the other hand, the current demonstrator aims just to provide a hand-on experience for the basics; and thus there will be no units for multiplication, division and square root. In particular, it can be build with moderate effort and for a low price. Internally, modern integrated circuits, mostly the CMOS 4000 series, are used. A variant with tubes would not be a technical problem, but significantly more expensive, and the high voltages used would inhibit public use, as these are present on the plugs.

Architecture

The demonstrator uses 5 digits plus sign, while the original ENIAC had 10 digits per accumulator and could join two for a double precision number. This is merely a question of cost and effort; expansion to 10 digits is easy and does not require substantial changes.

Digit trunks

The digit trunks of the ENIAC are the means over which the digits of the numbers are transmitted.

It is a true bus in a modern sense [Goldstine1946]:

> A transmitter consists essentially of an inverter tube
> whose output is fed to the grids of 2 amplifying tubes
> which have their plates connected in parallel.
> The cathodes of the amplifier tubes are connected in parallel
> to ground through a resistor and the output of the transmitter
> is taken off between cathode and ground.
> As previously mentioned, varying numbers of output transmitters
> can be connected to the same program line or digit trunk
> since a load resistor is not built into each transmitter
> but is instead plugged into the trunk line.

The demonstrator uses substantially the same architecture, that requires termination resistors to ground to be plugged to the digit trunk. The temination plugs may contain LEDs powered by the bus driver to make a digit pulse visible, in particular if the machine is run

with reduced speed (100Hz instead of 100kHz).

As the demonstrator uses 5 digits plus sign, six lines plus ground are required. Using 8p8c RJ-45 plugs allows two ground lines, but is suitable for a labatory test setup only. Instead of DB-9, the bulky DIN 41622 connectors with 12 contacts are close to the original in appearance; they have two rows of knife contacts. Also, they are at least 50 years in use (the latest (!) standard was published 1965).

The upto six digit trunks had just one connector per panel, which could be either connected to one of five input or to the one output of an adder unit. For this reason, more than one digit trunk is required in the demonstrator to be mechanically close. As Goldstine1946 writes, simultaneous transmissions of different numbers were anticipated. However, all examples developed so far us only a single thread of program control as well as a single data transmission for that program step. Thus, a single digit bus is sufficient for the prototype.

Program trunks

To transmit the program pulses, mostly single connections are used. As opposed to the direct interconnection used by analog computers until their decline, the ENIAC used a trunk with a large number of parallel lines, that are contacted with short cables from the calculation panels.

It has not yet been found out which plugs were used in the ENIAC; possible choices are 4mm Banana plugs, quarter inch phone connectors, or RCA (cinch) coaxial connectors, the latter introduced in the 1940s, and thus are not historically excluded.

The quarter inch tip-ring connectors were well known at that time from telephone switchboards, and are not only good from haptic feeling; short cables are available from stock as audio patch cables, although for relatively high prices (5€ each), as they are professional stage equipment.

If price is an issue, RCA cinch plugs would be an alternative, as audio cables are cheap and fairly reliable. However, vertical receptables for PCB use, being handy for the test setup, are not very common.

Note that repeat loops require another source of the program pulse to be connected to the bus, thus it is a true bus that must be terminated in the same was as the digit bus.

Clock and power supply

The clock unit provides 5 clock signals which have to be supplied to all computing units, of which only a few are needed for a demonstration.

Power for the units can be supplied by the same cable. Needed are 0.15A for the digit busses due to their low impedance and parallel use; so a supply for 1A per digit bus is required.

Each panel thus should have two plugs connected in parallel, so power and clocks can be chainde. one to d

Thus, DB-9 connectors could be used too, but in this case chained from unit to unit. If cables are used for the connection, these should be male/female mixed, the output of the clock unit being female, the input of a calculating unit being male.

However, this depends highly on the mechanical arrangement, which is not yet clear.

Accumulator

While the ENIAC had 10 decimal digits (plus sign) per accumulator, the demonstrator provides just 5 digits, which will be more than enough for demonstration programs and due to the missing high-speed multiplier. Expanding to 10 digits would be rather easy, increasing

the size of the digit trunks and the number of places in the accumulator.

In the ENIAC, the accumulators were (decimal) ring counters; the engineeres came to the conclusing, that ring counters could be built with less tubes than binary counters with 10 states. Note that the ENIAC does all arithmetic by incrementing, so there is no adder[1].

As the demonstrator will use integrated circuits, it is not significant if the accumulators use ring or binary counters; the CMOS IC 4017 provides a decimal counter with decoded outputs and an reset input and thus is the ideal building block here that makes the rest of the logic fairly simple.

Selectors

The selectors connect either one of the input plugs α ... or the output plug A or S to the accumulator for a given number of clock cycles, once a start pulse has arrived.

A start pulse sets a flipflop, further start pulses until reset are without effect. If the flipflop is set, it routes the end-of-cycle clock pulse to a decimal counter, that is thus for the next clock cycle no longer zero and can activate the following logic. The decimal counter is the same as used in the accumulators and advanced with the end-of-cycle clock until reset.

A manual selector switch selects one of the (decoded) outputs of the counter and, when the selected count is reached, enables to:

- send out the done pulse which might be routed to another unit,
- reset the start flipflop
- reset the decimal counter (buffered by a flipflop to avoid race conditions)

Circuit description

A major decision is design of the digit and program control trunks. In the ENIAC, the busses are driven by cathode followers and are just terminated with resistors to ground. Using inverted pull-down bus drivers has several advantages (see Appendix), in particular for TTL and if high speed beyond 1MHz is desired. As this is not the case here, and the ENIAC architecture uses positive pulses that are counted, a similar approach is used in the demonstrator by using active high pulses with a passive resistive termiator.

As power rail, 5V should be sufficient, see below for power consumption. As CMOS-logic is used, it should equally well run with 5V or 12V; however, if 5V works, it would mean less power for the bus terminations.

The termination resistors for external bussses are choosen as 150Ω. If the bus lines have 1nF capacitance (10m coaxial cable), the time constant is 0.15µs and the time to fall to 1V is 0.33µs, which is uncrititcal for pulses of 2.5µs duration. Moreover, the slow slope is the falling slope, and the clock signals are using the rising slope. A 180Ω resistor can be paralleled with a low power LED in series with 1200Ω resulting in 2mA for the LED, so no power is required to show the status in the terminators.

For an active high output, either a bipolar NPN transistor (2N3904) with open emitter or a p-channel MOS-FET (BS250, ZVP2106A) with open drain can be used as bus drivers. With the bipolar transsistor, however, there is a voltage drop of 0.7V for the transistor, and (with B=100) one of 0.3V for 0.3mA from the output of a standard CMOS gate, resulting in 1V below V_{CC}, which is just the noise margin without saftety. Either npn-transistors with guaranteed B>150 can be used, or NAND Schmitt-trigger gates as bus input gates, which have 3.5V max. upper threshold. Moreover, if the transistor is off and the bus is active, the emitter-base diode is loaded with a negative voltage near the specified limit of 6V, which is a problem not arising with valves.

Thus the MOS-FET is preferred, allowing also a Schottky diode to be inserted for decoupling

(used in the program control).

Internal busses (within an accumulator) are shorter and do not need independent state LEDs, thus the terminator resistor could be 1.2kΩ, and bipolar drivers are possible.

Using CMOS tri-state devices, e.g. the 4502, 4503 or 4016 buffers, the lines still need a termination resistor, to keep them at zero if all transmitter are switched off. However, this depends on an activation signal that does not change while the input is acivte, which is not the case here. A termination resistor of 47k&Omega would have been sufficient in these cases.

The peak current is 33mA for 2.5µs, the time of a single pulse; per 140µs cycle time this is 0.6mA on avarage. Sent are at most 55 digit pulses per cycle on a 5-digit-plus-sign bus (10 for the lowest, 9 for all other digits), thus 32mA average current per sender, i.e. 64mA for an accumulator that sends on both output plugs. A power distribution of 100mA per unit should not give any problems.

The clock unit sends out 25 pulses per cylce, while the cycle time is 56 pulse widths. Thus, the duty cycle is 50% with respect to a single pulse, thus the average current is 15mA.

The program pulse bus receives at most one pulse per 140µs, thus the power requirements are neglectible with 0.6mA.

As the bus termination is passive, just the resistors are needed. In order to visualize the action on the bus, LEDs may be used in the termination plugs powerd by the bus itself. Using standard 20mA LEDs would leave 10mA for the final 1.7V to discharge, which would exclude Schmitt-Trigger inputs or HCT logic. Thus, only a low power LED with 1.5kΩ in series, and 180Ω in parallel should be used.

Driving the bus may be done non-inverting with bipolar NPN-Transistors added to the outputs of non-inverting gates, which would mimic the way the ENIAC was build. However, this scheme drives the line not much above 4V with a moderate output resistance, and would not allow schmitt trigger line receiver inputs reliably. Thus, the preferred way is to use a P-MOS FET like the BS250 as inverting bus drivers driven by a standard CMOS inverter. Using HCT three state buffers is not necessary, even if this would save the extra P-MOS FETs.

Internal busses, in particular those in the accumulator, will not have such large stray capacitances, thus a termination resistor of 1.5kΩ will be sufficient. Polarity is not fixed, as the bus is internal; inverting MOS-FETs are the primary choice to pull the bus line up or down.

Supply bus and ground lines

The primary ground connection is via the power supply connectors.

The clock signals (see below) are also supplied via this connector, and thus use the same as signal ground. Thus, there is no need for another ground connection, as all units are connected to this ground line anyhow and permanently.

There are two more busses, with potential ground connections. Multiple ground connections can establish ground loops and be a source of sporadic failures. While the following considerations may be not be vital due to the relatively slow operating speed, it does not harm to keep to the rules as if a higher speed were necesary.

The control connections do not need additional ground connections and can be plugged by single wires, as they are not very time critical and spikes during transmission are irrelevant, as they set a D-FF anyhow. As the control lines are actually not a bus in the sense that more than one sender sends a pulse, they can be terminated in the sender. If 1/4 inch telephone (tip-ring) plugs are used, the sender should supply it ground, and the receivers leave it open (or connect via a resistor of e.g. 10 Ω). Thus, pseudo-terminators with light indicators can be

used, but are not needed.

The data buses are different, as more than one sender can use the bus, thus the single termination plug is necesary and needs a ground connection, that is not the supply ground for ease of construction. Thus, each sender should ground the bus to its signal ground connected to the power ground, so that a return path for the current from the terminators is supplied.

At the receiver side, no ground connections should be used, or as above via a resistor.

Accumulator

An accumulator contains - one register to store and manipulate (add and send) numbers, - several program controls, including the repeat and clear switches, - one bus coupler that routes the signals to or from the register from or to the digit bus, controlled by signals from the program control.

The bus coupler provides the connection to the power and clock bus, and routes power and clocks to the accumulator and the program controls.

The program controls send gate signals using a *wired or* bus; thus any number of program controls can be used.

The accumulator does not support chaining for double precision numbers, and does not provide an direct unswitched output.

Register

The register is a subunit of the accumulator. It holds a number, can add by counting, and can send its contents. The demonstrator will use 5 digits plus sign; if 10 digits are demanded, 11 lines would be used instead of 6.

It shows the current state per digit with 10 lights each. In the prototype, the accumulator uses LED to indicate the digits. For the demonstrator, miniature neon lamps could be used in connection with a voltage converter. The neon lamp requires 1.5mA at 90V, which would be about 30mA at 5V, at 50% efficieny 60mA per Digit (only one is lit at any time). If a CCFL-Inverter can be used, has to be checked.

The circuit is unchanged, but a high voltage driver MOS-FET like the BUZ60 or IRF540 is to be used. The use of TTL drivers like the ULN400x makes less sense, as there are 7 in a case, so there is a waste of about 1/2 package each.

The interface used is, besides power:

5 Clocksignals:

- `10P`
- `9P`
- `1N`
- `Cy`
- `EoC`

12 Output signals:

- `OutA1` to `OutA6`
- `OutN1` to `OutN6`

9 Input signals:

- `SendA`, `SendN`

- Clr
- In1 to In6

The output signals are, for economy of logic, already split to add and negated pulses, but inverted, so that the bus driver transistors could be on the bus interface. No multiplex is necessary, as there are only single connectors to send the number, one for addition and one for subtraction,

The input signals, however, need a multiplexer on the bus interface. They are just inputs; the bus interface may internally use a bus, or are all local bus signals terminated in the register, driven by the program controls (SendA, SendN, Clr) or switched bus buffers (In1 to In6). Any signals received on the In1 to In6 lines will immediately clock the counters, so there bus buffers must be enabled before the clock pulses arrive, which is the case with the pulses generated by the program controls.

When the Clr control is active, all digits in the register are set to zero at EoC, independend of any other control.

Like the original, each digit has 10 lamps (LEDs) to indicate the number stored; if a number is sent, the lamps are cyclically lit, like in the ENIAC.

So the internal connection from the accumulator to the bus coupler needs at least 26+2 wires, so common 34pol ribbon cable is used.

Program Control

A program control is a device that receives a program pulse from a program pulse bus line, and can issue such a pulse. It sets a gate signal for one or more full cycles, and finally emits a program pulse that could be routed to the next program control in the chain. Program controls are used also in the constants unit.

While the original machine used a more stringent scheme for the program pulses, requiring a dummy program control if a conditional jump is required, this demonstrator works slightly different. Except that pulses may not arrive during EoC pulses (including a pulse width time before and after), any pulse that is received sets a flip-flop, and the next EoC pulse starts the gate signal that lasts one or more cycles and ends when another EoC pulse arrives. Any further pulses received before the output pulse is sent are ignored.

The output program pulse for the program sequence trunk is actually derived from the 1N pulse.

Gate output is for a wired or bus, so that all program controls can share the lines to the register and the switches that connect the input plugs to the register, with a bus too. As mentioned, there is no Add gate; the input selection gates are sufficient.

Program control input comes directly from the program control trunk, and the emitted program pulse must use a bus driver as it is connected to a program trunk line.

The *clear correct* switch in the original ENIAC is, if not unclear, a bit complicated. For reasons of economy, a slightly different solution is used. The *clear* switch has three positions:

- neutral: nothing done
- Clear before, by clearing with the Pin pulse.
- Clear after, by clearing with the Pout pulse.

Obviously, *Clear before* is only useful with additions, and *Clear after* with send out. Pin is gated off after EoC, so any pulses arriving will not clear between repetitions.

Constant Generator

The ENIAC's constant generator supports (per panel) four 5-digit constants set per switches (including sign), and the punched card reader. For the switched constants, six program controls are available, that can either activate 5-digit numbers, or join the two 5-digit numbers in the lower row to one 10-digit number. Only one connector to the digit trunk is provided.

Correspondingly, the demonstrator provides two rows of constants set per switches, each row having a 2-digit and a 3-digit constant. The program controls thus have 5 positions:

- left upper (2-digit) constant
- left lower
- right upper (3-digit) constant
- right lower
- lower row joined as 5-digit constant

The program controls have not repeat (like in the ENIAC) and no output (unlike the ENIAC), as it seems useless to send out a constant without a register to receive the number. (Maybe the program outputs were used with the program control inputs in the accumulator that had no outputs.)

Bus Interface

The bus interface connects the accumulator to the digit bus, controlled by the program control.

For each of the five input sockets, a switch connects the 6 digit lines to the register while a signal from a program control is active. This switch internally uses a bus The latter 5 lines are a *wired or* bus, thus a single pole mechanical switch in the program control can provide the pulse.

so that a simple 1-pole switch in a program control unit could activate it.

For each of the input plugs, there is a 6 line switch that transmits the input pulses to the internal digit bus to the register, activated by a gate signal from a program control.

Clock Generator

The clock generator is quite similar to the original one, nevertheless some deviations were considered useful.

The basic clock speed is 100kHz as with the original, but it may be reduced down to 10Hz for better visbility; then, one cycle will last 1.4 seconds.

With full speed, a 10µs clock period is used derived from a two phase non overlapping clock of 400kHz by using a counter with four stages. Thus, the basic pulse width is 2.5µs, and the space between pulses 7.5µs.

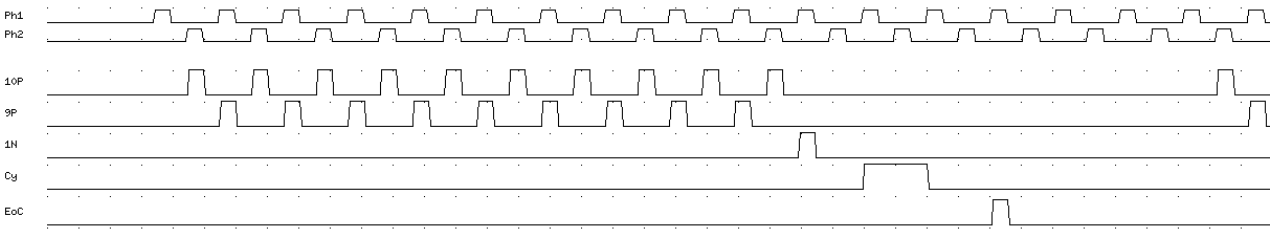
The clock cylcle is substantially the same as in ENIAC, but not all pulses are generated, and some names have been changed together with their intended use:

- `10P`: 10 pulses surrounding `9P`
- `9P`: 9 pulses to send out digit pulses
- `1N`: sent out as 10th pulse for negative numbers
- `Cy`: apply saved carry bits
- `EoC`: end-of-cycle, e.g. clear carry flipflops

ENIAC used the `CPP` (central program pulse) instead of `EoC`, which was followed by more pulses not necessary in this demonstrator.

A machine cycle uses 14 clock periods, but can be reconfigured to 20 like the ENIAC, and

also the Cy pulse is longer (10µs) to allow ample time for carry propagation, although the simple pulse width should have been sufficient. All clock pulses are on the 10µs raster, except 10P, which is offset by 5µs:



The clock generator has four pushbuttons to change the operation mode:

- Reset: Immediately halts and resets the clock, resets all accumulators to zero and clears all program links.
- Single clock: issues the next clock step, i.e. halts after one clock cycle.
- Halt / Single cycle: runs one clock cycle, i.e. halts after EoC has been issued.
- Run: resumes at the next clock step and runs the clock cycles continuously until one of the other buttons is pressed.

Thus, to halt during a run, press Halt / Single cycle to stop after the current cycle. Single clock stops after the current clock cycle.

In order to start a program, a single pulse has to be generated. This is done by a small unit which could be located in any module, as it only uses the normal clock chain and sends the 1N pulse once. For convenience, it is contained in the clock module.

A second similar device is also contained in the clock module, that was not present in the ENIAC, but is useful for demonstrations. It has the usual input and output for the program pulse and a touch button with a lamp. When an input pulse is received, the lamp is lit, and if button is pressed, the output pulse is generated. This allows to interrupt any chain of program pulses and to wait until the results are fully observed. While this could replace the inital start button, by activating the lamp upon reset, two buttons are easier to handle.

Constant transmitter

The ENIAC's constant transmitter had 4 switch fields for 5-digits plus a sign for each, and 6 program controls to transmit the constants. If more than five digits were needed, two switch fields could be joined, which is not needed in the demonstrator.

The ENIAC also had a punched card reader connected, and the remaining 24 program controls could transmit numbers from punched cards. This in particular allowed to run the calculations for a set of arguments, e.g. for ballistic tables, and then restart with a new card.

For a demonstrator, these features are not needed.

While the ENIAC clock generator provided additional clock signals for the generation of constants (just to save digital logic), this demonstrator uses internally a decimal counter and circuitry similar to that used in the accumulator.

Card punch

The ENIAC had a card punch where results could be made externally available, e.g. to be printed by a tabulating printer.

In the demonstrator, a serial interface, when activated via a program control, sends out each number received on the digit bus to a serial line, followed by a carriage return and a line

feed, and then sends the continuation program control.

If the transmission speed can be choosen by configuration switches, either a teleprinter or a computer could be connected.

Function Tables

The detached movable units shown on most photos were used to supply parameters during a calculation. On each side, there were about 50 rows of 10 switches each, thus the tables could provide 100 numbers during the calculation.

Tables were a significant part of the ENIAC architecture, because they receive their address from the digit bus, but the will not be available in the demonstrator, at least currently.

If supplied, the number of cells will be significantly smaller; but there are still no nice examples available.

Input and Output

The ENIAC could read data from a card punch and write to a printer or a card punch. Depending on the code used and the interface to the card punch, this could be quite a challenge with LSI circuits.

A serial (TTY) interface with a microcontroller is cost- and labor-effective, but it needs as many inputs and outputs as there are digits, which is not possible for the cheap ones. It might be considered to use one tiny microcontroller with 8 pins to collect the numbers on the lines, and then transfer them via a digital bus like IIC to another one doing the serial communications.

For some examples, such a device would be useful, e.g. for printing out prime numbers.

Special cables

The most important use — and necessity of — special cables are program flow branches.

Most common is a branch depending on the sign. Here, two identical cables are needed that have at one end a connector for the digit bus, and on the other end a connector for the programme control bus, connected to the line that transmits the sign.

Moreover, a dedicated accumulator is required, where both output, the positive and the negative one, are used. If a branch depending on the sign is required, a program control sends out the number via both, the positive and the negative plug. If the number was negative, nine pulses will be sent on the sign line of the positive digit bus plug, otherwise on the negative one. Depending on this, the next progamme step is activated accordingly.

Examples

Most of the examples to follow can be demonstrated on a fairly small machine.

[Hansen2003] and [Riley2013] has proposed symbolic notations of the connection arrangement, but both do not allow a compact notation along the the execution logic; they are more or less an external display of the internal structure.

The notation used here is fairly simple and line oriented. Each line contains tokens (without white space) separated by white space. Special charcters may freely be used for better readability. Capital and small letters are considered equal.

A semicolon or a vertical bar starts a comment until the end of the line. Empty lines are ignored.

A line starts with a token indicating the target to be set; entries are accumulative. The lines can be in any order; recommended is to first set the programme trunk, then the digit trunk, then the program controls used in this step.

Device designations are (digits represent arbitrary numbers):

- P5: Programme trunk 5
- D3: Digit trunk 3
- A2: Adder 2
- C1: Constant unit 1

A digit trunk line contains the terminals connected to the given digit trunk. Digit input terminals are designated by the (small) letters `a`, `b`, `c`, `d` or `e` (e.g. `A1a`); the glyps `α` to `ε` may also be used. Because there is no case distinction with letters, the digit output terminals are denoted by `P` and `N` instead of *A* and *S* (e.g. `A1P`); `P` may be read as *plain*, and `N` as *nines complement* or *negated*. If *positive* and *negative* are considered as mnemonics, note that `P` for a negative number still sends a negative number. Instead of `P` and `N`, `+` and `-` may be used.

A typical digit trunk line is:

```
D2 A1P A2a  | connect plain output of A1 to first input of A2
```

A programme trunk line contains programme control inputs and outputs. Inputs are designated by the number following the device, usually separated by a point, e.g. `A2.1`. As usually there is one output connected to two or more inputs, the first program control is always an output, without any additional signs or characters. The start button is designated by the letter `G` (for *go*). Thus, a typical programme control line might read:

```
P1 G A1.1 A2.1      | start activating A1 and A2
```

To make a demonstration easier, two (or more) GO program pulse geneators are provided, written as `G1`, `G2` etc, so that a program could be continued at another place if stopped.

If more than one outputs are connected to a programme trunk, additional lines must be used. If it is desired to write a line without an output, and it is not sensible to re-use an output, the special device `N` may be used.

Program controls start with the program control indication, i.e. the device name followed by a number, usually separated by a point. It follows the terminal letter (or symbol) and optionally the repeat number (repeat is 1 if omitted). Thus, a typical program control line reads:

```
A1.1 a 2            | add input a twice
```

The clear switch is indicated by the letter z (anywhere in the line):

```
A1.1 za 2       | zero and add
A2.1 Pz 1       | send and zero
```

For constants, the rest of the line contains the (signed) number.

Tabulating squares by pushing the start button for each number thus reads:

```
| add count in A2 to squares in A1 twice
P1 G A1.1 A2.1      | go pulse activates first round
D1 A1P A2a          | positive output goes to a input
A1.1 a 2            | add input a twice
A2.1 P 2            | send plain value twice
| increment square accumulator A1
```

```
P2 A1.1 C1.1 A1.2
D1 C1P A1a
C1.1 1
A1.2 a
| increment count in A2
P3 A1.2 C1.2 A2.2
D1 C1P A2a
C1.2 1              | send value 1
A2.2 a              | add input a
| machine stops
```

Note that for the real machine as well for the demonstrator, an accumultor needs to use as many digit bus trunks as inputs and output are used, because there is only one connector for each digit bus trunk per unit, i.e. Accumulator.

As the constants unit has only one output plug, data bus trunks must be allocated starting with the constants.

Using Integer Numbers

Squares and Cubes

According to the well known binominal formula, the squares of the integer numers could be enumerated by adding twice the argument plus one:

$$(x+1)^2 = x^2 + 2 \cdot x + 1$$

Two accumulators and one constant unit are required.

The program is fairly easy and requires of two program controls:

- add twice the contents of the first accumulator containing the argument to the second one
- add the constant 1 to both accumulators once.
- stop, i.e. do not continue; restart without clear

This reads symbolically (same example as above):

```
| argument in A1, square in A2
| add count in A1 to squares in A2 twice
P1 G A1.1 A2.1      | go pulse activates first round
D1 A1P A2a          | positive output goes to a input
A1.1 P 2            | send plain value twice
A2.1 a 2            | add input a twice
| increment both, A1 and A2
P2 A1.1 C1.1 A1.2 A2.2
D1 C1P A1a A2a
C1.1 1
A1.2 a
A2.2 a
| machine stops, continue via GO without reset
```

Before the first round, the accumulators are cleared by a global reset. (It has to be checked if this was available in the original machine). The first round then does nothing in the first part, and increments both registers to 1 in the second part.

If a third accumulator is available, the cubes can be calculated simultaneously without increasing the run time, in contrast to all later machines[2], using the formulas:

$$(x+1)^3 = x^3 + 3 \cdot x^2 + 3 \cdot x + 1$$

$$(x+1)^2 = x^2 + 2 \cdot x + 1$$

Let the argument be in A1, the squares in A2, and the cubes in A3, then the straightforward program might read:

```
| Add the square from A2 thrice to A3
P1 G A2.1 A3.1
D1 A2P A3a
A2.1 P 3
A3.1 a 3
| Add the argument A1 twice to A2 and thrice to A3
P2 A2.1 A1.2 A2.2 A3.2
D1 A1P A2a A3a
A1.2 P 3
A2.2 a 2
A3.2 a 3
| Increment A1, A2 and A3
P1 A1.2 C1.1 A1.3 A2.3 A3.3
D1 C1P A1a A2a A3a
C1.1 1
A2.3 a
A3.3 a
| stop
```

Note that in the second step, the third number sent by A1 is ignored by A2 as the count is exhausted.

Unfortunately, no subtraction is used here.

Fibonacci numbers

These are caluclated with the recursion fourmula

$$f_{n+1} = f_{n-1} + f_n$$

If there are only two accumulators, the numbers must be produced using alternate accumulators, according to:

```
x = 0
y = 1
repeat
    x =+ y
    y =+ x
```

Thus, the programme might be:

```
| produce Fibonacci number alternating in A1 and A2
| initialize accumulators
P1 G1 C1.1 A1.1 A2.1
D1 C1P A2a
C1.1 1      | send 1
A1.1 za     | clear
A2.1 za
| first step: add A2 to A1
P2 C1.1 A2.2 A1.2   | constant triggers next step
```

```
D2 A2P A1a      | need another bus for output
A2.2 P
A1.2 a
| second step: Add A1 to A2
P3 A2.2 A2.3 A1.3
D3 A1P A2.b
A1.3 P
A2.3 b
| For the next step, use 2nd GO button instead of C1
P2 G4
```

If a third accumulator and enough programme controls, the three accumulators could be shifted circularily at each round.

Greatest common divisor

Because the demonstrator does neither have the multiplication nor the division unit of the ENIAC, calculating the greatest common divisor is a lengthy run, as the division is done by subtraction as long as the result is not negative. Fortunateley, only the remainder is relevant, which is what is left over after the last subtraction is reverted.

The GCD program reads in its recursive version:

```
gdc(x,y):
    assert x > y and y >= 0
    if y > 0
            return gcd(y, x % y)  -- % is the modulus
    return x
```

Replacing the tail recursion by a loop:

```
gcd(x,y):
    assert x > y and y >= 0
    while y > 0
        z = x % y
        y = x
        x = z
    return x
```

As we will replace the remainder by continuous subtraction, and using additions to a variable only, the program will be:

```
gcd(x,y):
    while y > 0
        z =+ x
        while z >= 0
            z =- y
        z =+ y; y = 0
        y =+ x; x = 0
        x =+ z; z = 0
    return x
```

Let us have `x` in `A1`, `y` in `A2` and `z` in A3, the initial values in `C1` and `C2`, using 20 and 15 as example. For the conditional branches, another accumulator is needed.

Here is the program:

```
Init > P1
```

```
# transfer initial values
P1 > C1.1 > C2.1 > A1.1 > A2.1
C1.1: !20
C2.1: !15
A1.1: +a
A2.1: +a
C1_+ = A1_a = D1
C2_+ = A2_a = D2
C1.1 > P2
# copy x to z
P2 > A1.2 > A3.2
A1.2: !+
A3.2: +a
A1_+ = A3.a = D1
A3.1 > P3
# subtract y from z
P3 > A2.3 > A3.3
A2.3: !-
A3.3: +a
A2_- = A3_a = D1
A3.2 > P4
# conditional branch on z (dummy transmission without receiver)
P4 = A3.4
A3.4: !+-
A3_+ = D1
A3_- = D2
D2 = P3     -- repeat if negative number is negative
D1 = P5     -- else continue
# restore remainder: add y to z, clear y
P5 > A2.5 > A3.5
A2.5: !+ 0
A3.5: +a
A2_+ = A3_a = D1
A2.5 > P6
# move x to y
P6 > A1.6 > A2.6
A1.6: !+ 0
A2.6: +a
A1_+ = A2.a = D2
A1.6 > P7
# move z to x
P7 > A3.7 > A1.7
A3.7: !+ 0
A1.7: +a
A3_+ = A1.a = D1
# loop back if y>0
P8 > A2.8
A2.8: !-
A2_- = P2
# otherwise, stop
```

Square and Cubic Root

A simple and quite unefficient method to determine a square root uses the fact that the sum of odd numbers is the square of its count:

$$\sum_{k=1}^{n} (2k-1) = n^2$$

Instaed of tabulating the square numbers with the difference method until the nearest (or lower or upper) square is found, an accumulator can be saved if the odd numbers are subtracted from the argument until the result is negative:

```
y = 1
while x > 0
    x =- y
    y =+ 2
```

This will round up the result, because the comparison to y instead of 0 is not simple, as the ENIAC can branch only on negative or not negative, not on zero[3].

So this example is probably the shortest one to demonstrate conditional loop repeat, but the constants unit should be able to supply two numbers, the value to be rooted and the constant 1.

Collatz Series

The *Collatz Conjecture* claims that the series

$$c(i) = \begin{cases} \frac{i}{2} & \text{if } i \text{ odd} \\ 3i+1 & \text{if } i \text{ not odd} \end{cases}$$

will eventually reach 1.

As there is no divider, division by two has to be done by subtracting 2 until the sign changes, adding one and branching dependent on the sign then. Fortunately, the quotient can be counted in parallel.

If the quotient were not needed, a special cable could first extract the last digit only.

Using fractional numbers

Using fractional numbers, i.e. decimal fractions, is simple. Just assume the decimal point not after the last digit, but before any other digit. For additions (and subtractions), nothing more is necessary, and consequently for multiplication with small integers using the repeat feature of the programme control.

Most convenient are fractions in the range $-9.9999 \le x \le 9.9999$, i.e. the decimal number after the most significant digit, as to avoid problems with intermediate results exceding ±1.

The range $-0.99999 \le x < +0.99999$ can be used by assuming the decimal point before the most significant digit, provided that the evaluation does not exceed the number range.

Multiplications would produce a double precision result. If the decimal point is before the most significant digit, just the upper half is the result; otherwise, the double precision reuslt has to be shifted n decimal places to the right, if the decimal point is befor the n-th digit.

Sine approximation

As shown in my notes on Tschebyscheff Approximations, the $\sin(x)$ function with all numbers in the ±1 range can be nicely approximated as

$$\sin\left(\frac{\pi}{2}x\right) \approx 1.5460x - 0.5464x^3$$

This cubic polynome can easily tabulated using the difference method and requires three accumulators, that keep track of x, x^2 and x^3.

To allow the numbers upto $\frac{\pi}{2} \approx 1.5708$ as arguments, ´x$has \rightarrow bereplacedby$2x/pi $\in thepolynome$:

$$\sin(x) \approx \frac{1.5460}{1.5708}x - \frac{0.5464}{1.5708^3}x^3 = 0.9842x - 0.1410x^3$$

To tabulate with 10° steps, the increment should be $\frac{\pi}{2 \cdot 90} = 0.1745$:

```
d = 0.1745
x = 0
x2 = 0
x3 = -0.5464
for 9 times:
    x3 =+ 0.1745
    x2 =+ x3
    x1 =+ x2
    x1 =+ 1.5460
```

So this is possible with the demonstrator.

Appendix

Open-Collector Bus Characteristics

With the advent of integrated TTL-Logic, the open collector bus became the dominant solution. One reason is the small noise margin of only 0.4V for low inputs. On the high to low transition, the transistor quickly discharges bus capacitors, and the low level magin is quickly reached. The high level input margin is 2.0V, i.e. below the mid-voltage, and so only the standard time constant of the pullup-resistor and the bus capacitance is relevant. The low-to-high slope is significantly slower than the high-to-low slope, thus negative slopes are prefered.

All in all, a negative logic where low is true and high false, would be the natural choice.

References

Zoppke2004:
Till Zoppke: *Simulating the ENIAC as a Java Applet*. Dipolmarbeit FU Berlin, 2004. http://www.zib.de/zuse/Inhalt/Programe/eniac/

Hansen2003:
Peter Hansen: *A Java Simulation of the ENIAC*. Diplomarbeit Univ. Osnabrück, 2003. http://home.arcor.de/-ph/eniac/

Riley2013:
Mike Riley: *RCS ENIAC simulator*. http://historicsimulations.com/eniac.html

VanderSpiegelTauAlailimaAng2000:
Jan Van der Spiegel, James F. Tau, Ttiimaea F. Ala'ilima and Lin Ping Ang: *The ENIAC: History, Operation and Reconstruction in VLSI*.

In: Raul Rojas and Ulf Hashagen: *The First Computers. History and Architectures*, p. 121ff, MIT Press 2000.

Muus2000:
Mike Muuss: *History of Computing Information*.
Online at http://ftp.arl.mil/~mike/comphist/

Goldstine1946:
Adele Goldstine: *A Report on the ENIAC*. Philadelphia, USA (1946)
Online version without schematics: http://ftp.arl.mil/~mike/comphist/46eniac-report/index.html

BrainerdSharpless1948:
J.B. Brainerd, T.K.Sharpless: *The ENIAC* Electrical Enigneering, vol.67, pp. 163-172, 1948
Reprinted in the Proceedings of the IEEE, vol.72, no.9, pp.1203-1212, 1984.
doi:10.1109/PROC.1984.13000

---

[1]It may be interesting to check if the ring counters could be made with one tube per digit, as with thyratrons.

[2]even the ACE and similar machines like the Bendix G3 and IBM's 620 did not have different counts for the sender and the receiver; this was too seldomly useful.

[3]which is not so bad, as robust programming tries to avoid equality tests for loop ends anyhow.