TD-Magic: From Pictures of Timing Diagrams To Formal Specifications

Jie He[1], Dejan Ničković[2], Ezio Bartocci[1], Radu Grosu[1]
[1]*Technische Universität Wien*, [2]*AIT Austrian Institute of Technology*
Vienna, Austria
[1]{jie.he, ezio.bartocci, radu.grosu}@tuwien.ac.at, [2]Dejan.Nickovic@ait.ac.at

2023 60th ACM/IEEE Design Automation Conference (DAC) | 979-8-3503-2348-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/DAC56929.2023.10247685

***Abstract*—We introduce TD-Magic, the first neuro-symbolic approach for translating an image of a timing-diagram (TD) to a formal specification. We overcome the lack of labelled data for supervised learning, by first developing a synthetic data generator of labelled TDs. We then use object detection techniques to identify rising and failing edges, OCR to recognise the text, and image processing algorithms to capture synchronisation patterns. Finally, we use semantic interpretation to analyse the extracted features and generate the associated formal specification. Our experiments on industrial TDs show high translation accuracy opening the way to more sophisticated requirements-extraction algorithms from pictures.**

***Index Terms*—Requirements Engineering, Formal Specification, Timing Diagram, Computer Vision**

I. Introduction

Timing diagrams (TDs) play a central role in the design of hardware systems. They provide a visual and diagrammatic specification of the timing behavior, which is intuitive to the engineer and widely adopted in the industrial practice. Figure 1(a) shows a TD D with two digital signals, X and Y. Signal X contains two pulses and signal Y contains one. A pulse is delimited by a rising and a falling edge, characterized by the signal crossing the threshold T_1 from above and T_2 from below, respectively. TD D describes three timing relations: (1) the duration t_1 of the first pulse in X, (2) that the first pulse in X is followed by a pulse in Y after t_2 time, and (3) that the first pulse in X is followed by another pulse in X after time t_3 between the end of the first pulse and the beginning of the second one.

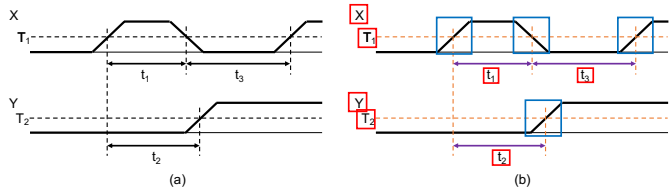

Fig. 1: Timing diagram D: (a) without, and (b) with identified features.

The common practice in the semiconductor industry is to use TDs to exchange the requirements between different stakeholders and as a visual aid to the designer in understanding the system's intended behavior. Typically, an engineer draws a TD by hand and a graphical designer creates a pdf picture (image) of it. *The resulting TD lacks an intermediate formal model and consequently it cannot be used in verification and testing activities*. To address this problem, multiple modelling approaches to formalise TDs have been proposed. Despite these important efforts, the vast majority of TDs found in datasheets of commercial products are still given in the form of images. Unfortunately, the formal specification of a TD from its pictorial representation requires considerable expertise and effort.

Leveraging on recent advances in *ML* (machine learning), *AI* (symbolic artificial intelligence), and *IP* (image processing), we propose TD-Magic, an approach for automatically translating a TD image to a formal representation, as shown in Figure 2.

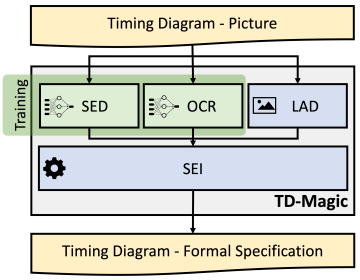

Fig. 2: Overview of TD-Magic

TD-Magic takes as input a bitmap-image of a TD. In the first step, it recognizes the main features in this image: the different signals, their rising/falling edges, the duration, threshold parameters and constants, and the arrows with their associated vertical lines denoting events and their timing relations. To facilitate feature recognition, we developed *SED* (signal-edge detector) and *OCR* (optical character recognition) modules by using state-of-the-art computer vision methods for object detection relying on deep supervised learning. To train these two modules, we needed labeled data and to the best of our knowledge, no public labeled dataset for TDs is available. We therefore devised a procedure to synthetically generate labeled pictures of TDs. The *LAD* (line-and-arrow-detection) module complements the ML modules, with morphology analysis from the field of IP, in order to better identify arrows and horizontal/vertical lines in the picture.

Figure 1(b) shows the expected outcome of identifying features in D. SED and OCR output for every feature a *bounding box* in its diagram's coordinates (red for texts, and blue for edges). OCR also associates the recognized text to its bounding box. LAD, aided by *SEI* (semantic interpretation), outputs the detected vertical and horizontal lines (orange dotted) and arrows (violet). Finally, SEI analyses all extracted features and their relations, to generate a formal representation of D in form of an *SPO* (strict partial order), among events. These are annotated with their type (e.g., ramp-up or ramp-down with corresponding height and width, and associated signals, such as, X and Y), and duration constraints between the events (e.g., t_1, t_2, and t_3). Figure 3 graphically depicts the SPO representation of D as a DAG, with nodes as the events, directed arrows as the order relation, and arrow annotations as the timing constraints imposed on the order.

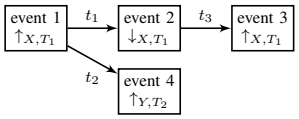

Fig. 3: Formal representation.

We experimentally evaluated the performance of TD-Magic on an extensive set of benchmarks, both synthetic and from the semiconductor industry. Our results show that TD-Magic is very effective in learning from synthetic TDs, and extrapolating to industrial TDs.

To summarize, TD-Magic combines advances from ML, AI, IP, and *FM* (formal methods), to automatically translate digital TDs into their SPO representation. TD-Magic significantly broadens the application of TDs. It bridges the gap between *RE* (requirements engineering) and FM and enables the use of model checking, runtime verification and testing tools with TDs as formal specifications.

II. Related Work

TDs are a very well established visual specification language [1], they are included in the Unified Modelling Language [2], and they capture with waveforms and graphs the evolution of the values of the variables in a system over time, and the casual dependencies of timing events occurring among variables in different subsystems. It found its best application in the design of hardware components.

Motivated by the need to automatically verify, monitor, or test TD requirements against a hardware design and implementation [3], a great deal of past work was devoted to the model-to-model translation of TDs into formal-specification languages such as VHDL [4], Linear Temporal Logic [5] and automata [3]. *All these works assumed to have access to the formal representation of a TDs in the form of an SPO*. However, this textual, formal representation is often not available. For example, in the semiconductor-industry practice, requirements specifications are usually in form of thousands of data sheets, with *several TDs each, and only available as pictures* in a document.

TD-Magic infers the PO by directly observing a TD image. To the best of our knowledge, there are no other similar approaches. The closest topic in the literature somewhat related to our work is the problem of signal digitisation. For example, in the biomedical domain, there has been a great deal of interest in automatically digitizing ECG signals from pdf documents directly [6], [7], using both traditional image processing [8]–[12] and the latest advances deep-learning technology [13], [14]. The major difference between our work and these methods is that we focus on simpler, *"digital" signals*, with rising and failing ramps, that have nevertheless very sophisticated synchronization patterns among them, describing for example, the duration of timed events with horizontal arrows, and the desired timing between events by using vertical bars. We take advantage of recent advances in computer vision using object detection techniques [15], [16]. In particular we employ YOLO [17] to identify rising and failing edges, OCR to recognise the text, and IP to capture the other graphical patterns describing synchronisation and time duration.

III. Preliminaries

Data Collection. We aim to identify the main features present in common industrial TDs of products' data sheets from major semiconductor manufactures. We selected 29 representative industrial TDs from STMicroelectronics™ and Infineon Technologies™. We restricted our attention to TDs with up to three digital or piecewise linear analog signals with maximum five rising/falling edges, and maximum ten timing constraints.

Timing Diagrams Description. We consider timing diagrams with three different categories of signals: digital signals (e.g., signals V_{INA} in Fig. 4), analog signals with *ramp* edges (e.g., signal V_{OUTA} and SCK in Fig. 4) and analog signals with *double-ramp* edges (e.g., signal SI in Fig. 4). These signals may contain five types of edges: *riseStep*, *fallStep*, *riseRamp*, *fallRamp*, *double*. In addition we consider the following graphical and textual annotations that are important for the semantic interpretation of TDs:

1) *Signal names*, often positioned near the y-axis arrow, e.g., V_{INA} and V_{OUTA} in Fig. 4. In the absence of axes, names are usually in the leftmost part of the signal, e.g., SI and SCK in Fig. 4.
2) *Boundary values* describe the maximum and minimum signal values, e.g. V_{CC} gives upper and GND lower bound in Fig. 4 (on the right). Not all TDs are annotated with boundary values (e.g., see Fig. 4). They are often placed to the right hand side of the signal, to the left side of the y-axis, when axes are available.
3) *Thresholds* are constants associated to rising and falling edges in ramp and double signals. Typically, a threshold is defined as a percentage of the maximum expected signal value. For example, there are two thresholds defined for the signal V_{OUTA} in Fig. 4. They are annotated with "90%" and "10%" and are associated to the *riseRamp* and *fallRamp* edges, respectively.
4) *Events* are instantaneous occurrences of interest in signals. In digital signals, an event corresponds to either a *riseStep* or a *fallStep* edge (e.g., signal V_{INA} in Fig. 4). In ramp and double signal, an event corresponds to the time when a *riseRamp*, a *fallRamp* or a *double* edge crosses a threshold (e.g., the crossing of the 90% threshold by the first *riseRamp* in signal V_{OUTA} in Fig. 4). The events in all types of signals are often explicitly marked with a vertical line.
5) *Timing constraints* between TD events capture timing properties, typically marked by a double-headed arrow, between two vertical lines. The constraint is annotated with a timing parameter (e.g., the parameter $t_D(on)$ anotates the timing constraint between the rising edge of the signal V_{INA} and the rising ramp event of the signal V_{OUTA} in Fig. 4). The timing parameter values are often defined in the data sheets outside of the TD. This is for instance the case for $t_D(on)$, which is specified in the Table 10 [18].

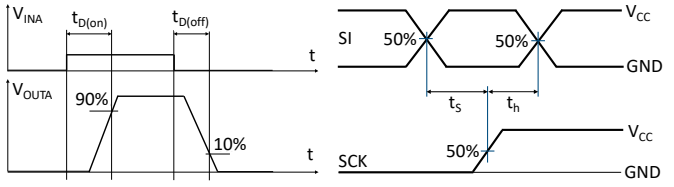

Fig. 4: (left) Example 1 (redrawn from Fig. 6 in [18]), (right) Example 2 (redrawn similar to Fig. 9 in [19])

TD pictures might also have optional (e.g., in Fig. 4 on the left are present, in Fig. 4 on the right are absent) components such as axes.

TD semantics. The semantics of a TD, that is its formal specification, is captured with a strict partial order.

Definition 1 (SPO): A tuple $P=(E,<)$ is an SPO (strict partial order), if $<$ is an irreflexive, asymmetric, and transitive relation over a set of events E. In other words, for all events $e,f,g \in E$: (1) $e \not< e$, (2) $e<f \Rightarrow f \not< e$, and (3) $e<f \wedge f<g \Rightarrow e<g$.

Below, we formally define the set of events E and the partial order relation $<$. We will annotate $<$ with an associated duration. The SPO can be alternatively viewed as a directed acyclic graph (DAG), with events as nodes and arrows as edges.

- *Nodes*. A node $n=(sn,ei,et,th)$ is a four-tuple, where sn is the name of the signal containing the event, ei is the index of the edge in the event, et is the the edge type, and th is the threshold associated to the event. All events are indexed by their global occurrence in the TD, sorted from left to right. If the event occurs on a *riseStep* or *fallStep* edge, then th has the value $None$.
- *Edges*. An edge $e=(sn,td,dn)$ is a timing constraint linking the source node sn with the destination node dn through the time delay td. The formal specification of a TD can be extracted through a depth-first search from its DAG.

Example 1: According to the TD semantics above, the formal specification of the TD in Fig. 4(left) can be given as follows:

$$n_1=(V_{INA},1,riseStep,None)$$
$$n_2=(V_{OUTA},1,riseRamp,90\%) \qquad e_1=(n_1,t_{D(on)},n_2)$$
$$n_3=(V_{INA},2,fallStep,None) \qquad e_2=(n_3,t_{D(off)},n_4)$$
$$n_4=(V_{OUTA},2,fallRamp,10\%)$$

Example 2: Fig. 4(right) has three events: (1) The first 50% crossing point of signal SI; (2) The time point when signal SCK increases

above the 50% threshold; (3) The second 50% crossing point of signal SI. Event 1 and Event 2 are associated with the timing constraint t_s, and Event 2 and Event 3 are associated with the timing constraint t_h. The ranges of t_s and t_h can be found in Table 7 of [19]. The formal specification of the TD results as follows:

$$n_1 = (SI, 1, double, 50\%) \qquad e_1 = (n_1, t_s, n_2)$$
$$n_2 = (SCK, 1, riseRamp, 50\%) \qquad e_2 = (n_2, t_h, n_3)$$
$$n_3 = (SI, 2, double, 50\%)$$

IV. Dataset Generation

Supervised DL requires labeled TDs, and manual labeling is tedious. It also requires expertise in both circuit design and FM. To solve this problem, we developed *L-TD-G*, an algorithm that automatically generates synthetic, labeled TDs, for training DL models.

Since we use DL to recognize different edges, we will maximise the diversity of their shapes (e.g., height, slope), and consider their interaction with timing constraints accordingly. The overall generation procedure is based on TDs that have been collected first.

Signal/Edge Selection. L-TD-G produces TDs with only two stacked signals (as in most industrial examples, and enough for learning). It first randomly selects two signal types, each with a rising and a falling edge. They also determine the type of the rise and fall edges, associated to a bounding box. L-TD-G will then create and randomly place two edge boxes inside the reference coordinate system of the signals. Starting from a sequence of horizontally aligned boxes, we can reconstruct the waveform of the entire signals.

Inter/Intra-Signal-Constraints Selection. To illustrate the possible inter-signal-constraints, we denote with b_{11} and b_{12} the edge boxes generated for *Signal*$_1$, and with b_{21} and b_{22} for *Signal*$_2$. Basically, b_{i1} is always on the left of b_{i2}, but despite this, there are still many combinations for the box order across signals, and some of them rarely occur in real TDs. To simplify SPO generation, we propose the following rules: (1) *Signal*$_2$ has to lag behind *Signal*$_1$; (2) There is no crossing of arrows when linking boxes between *Signal*$_1$ and *Signal*$_2$; (3) Only one arrow is allowed to point to one box.

These rules thus lead to only five cases of inter-relation orders supported by L-TG-G: (1) $b_{11} < b_{21}$, (2) $b_{12} < b_{21}$, (3) $b_{11} < b_{21} \wedge b_{12} < b_{22}$, (4) $b_{11} < b_{22}$, and (5) $b_{12} < b_{22}$. L-TD-G randomly selects one case for generating one labeled TD. Next, the annotation of the intra-relations $b_{i1} < b_{i2}$ in *Signal*$_i$, will be randomly selected.

Once boxes and their order are selected, L-TD-G proceeds concertizing the SPO-guided labeled TD. This implies defining the concrete sizes of the boxes in the signals waveform, and placing arrows for annotating inter/intra-relations of signals. As there is a lot of freedom in these choices, L-TD-G captures part the above in form of a set of SMT (satisfiability modulo-theories) constraints, such that each solution defines one possible realization of the SPO and its associated labeled TD. We discuss in detail these steps below on hand of Fig. 5.

SMT Inequality Constraints. In Fig. 5, *Signal*$_1$ and *Signal*$_2$ are shown in dark-blue and dark-red colors, respectively. Accordingly, the bounding boxes of their rising/falling edges are shown in light-blue and light-red colors, respectively. There are six parameters defining one bounding box. We enumerate them below, using box b_{11} as an example: (1) The leftmost x coordinate, e.g., $x_{1,1l}$, (2) The rightmost x coordinate, e.g., $x_{1,1r}$, (3) The downmost y coordinate, e.g., $y_{1,1d}$, (4) The topmost y coordinate, e.g., $y_{1,1u}$, (5) The width of the box, e.g., $w_{1,1}$, and (6) The height of the box, e.g., $h_{1,1}$. The naming rules for the parameters of the other three bounding boxes are similar.

The shape and placement of boxes determine the function values of the piecewise-continuous waveforms. We take *Signal*$_1$ as an example,

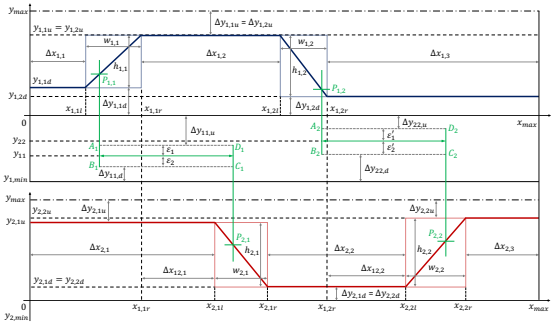

Fig. 5: Demonstration of Core Algorithm

again. Horizontally, we plan to reserve enough space to plot the starting/ending plateaus of the signal curve. Therefore we introduce two auxiliary variables $\Delta x_{1,1} = x_{1,1l} - 0$ and $\Delta x_{1,3} = x_{max} - x_{1,2r}$. Similarly, the two adjacent bounding boxes of *Signal*$_1$ should keep enough horizontal distance, so we introduce $\Delta x_{1,2} = x_{1,2l} - x_{1,1r}$. Vertically, we also plan to reserve enough bottom/top margins. At the bottom of the bounding boxes, we introduce two auxiliary variables: $\Delta y_{1,1d} = y_{1,1d} - 0$ and $\Delta y_{1,2d} = y_{1,2d} - 0$. At the top of the bounding boxes, since *Signal*$_1$ follows a rise-to-fall style, $y_{1,1u} = y_{1,2u}$, which leads to the same upper margin of the two bounding boxes: $\Delta y_{1,1u} = \Delta y_{1,2u} = y_{max} - y_{1,1u}$.

1) Signal-waveforms constraints: We now discuss the first two groups of constraints to be solved for plotting signal waveforms.

- *Group 1: The y-constraints of the waveform*. These constraints, with i representing *Signal*$_i$, are partitioned in the following subgroups:

  1.1. Eight unsolved y-parameters and their bounds $(i = 1, 2)$: (1) $0 \le y_{i,1d} < y_{i,1u} \le y_{max}$; (2) $0 \le y_{i,2d} < y_{i,2u} \le y_{max}$.

  1.2. Height of bounding boxes $(i = 1, 2)$: (1) $h_{i,1} = y_{i,1u} - y_{i,1d} \ge r_{yh} \cdot y_{max}$; (2) $h_{i,2} = y_{i,2u} - y_{i,2d} \ge r_{yh} \cdot y_{max}$.

  1.3. Vertical margins for *Signal*$_1$: (1) $\Delta y_{1,1d} = y_{1,1d} - 0 \ge r_{yd} \cdot y_{max}$; (2) $\Delta y_{1,2d} = y_{1,2d} - 0 \ge r_{yd} \cdot y_{max}$; (3) $\Delta y_{1,1u} = \Delta y_{1,2u} = y_{max} - y_{1,1u} \ge r_{yu} \cdot y_{max}$.

  1.4. Vertical margins for *Signal*$_2$: (1) $\Delta y_{2,1u} = y_{max} - y_{2,1u} \ge r_{yu} \cdot y_{max}$; (2) $\Delta y_{2,2u} = y_{max} - y_{2,2u} \ge r_{yu} \cdot y_{max}$; (3) $\Delta y_{2,1d} = \Delta y_{2,2d} \ge r_{yd} \cdot y_{max}$.

  In Group 1, the parameters r_{yh}, r_{yd}, r_{yu} are ratios within $[0, 1]$ for the height, bottom and upper margin of the bounding boxes.

- *Group 2: The x-constraints of the waveforms*. These constraints are partitioned in the following subgroups:

  2.1. Eight unsolved x-parameters and their bounds $(i = 1, 2)$: $0 \le x_{i,1l} < x_{i,1r} < x_{i,2l} < x_{i,2r} \le x_{max}$.

  2.2. Width of bounding boxes $(i = 1, 2)$: (1) $w_{i,1} = x_{i,1r} - x_{i,1l} \ge r_{xw} \cdot y_{max}$; (2) $w_{i,2} = x_{i,2r} - x_{i,2l} \ge r_{xw} \cdot y_{max}$.

  2.3. Horizontal margins $(i = 1, 2)$: (1) $\Delta x_{i,1} = x_{i,1l} - 0 \ge r_{xl} \cdot x_{max}$; (2) $\Delta x_{i,2} = x_{i,2l} - x_{i,1r} \ge r_{xm} \cdot x_{max}$; (3) $\Delta x_{i,3} = x_{max} - x_{i,2r} \ge r_{xr} \cdot x_{max}$.

  2.4. Inter-relations (Case 3): (1) $\Delta x_{12,1} = x_{2,1l} - x_{1,1r} \ge r_{xi} \cdot x_{max}$; (2) $\Delta x_{12,2} = x_{2,2l} - x_{1,2r} \ge r_{xi} \cdot x_{max}$.

  Similarly to Group 1, $r_{xw}, r_{xl}/r_{xm}/r_{xr}$ and r_{xi} are the ratios within $[0, 1]$ for the width of bounding boxes, left/intra-signal box distance/right margin, and inter-signal box distance margin.

2) Intra/inter-relations constraints: For arrows denoting intra- and inter-relations timing constraints, our strategy is to reserve a specialized area with height in $[y_{1,min}, 0]$ to plot them, as shown by the green lines and green arrows in Figure 5.

We first generate a set of constraints to compute the vertical coordinates of arrows denoting inter-relations, as shown by y_{11} (linking b_{11} and b_{21}) and y_{22} (linking b_{12} and b_{22}) in Figure 5. Afterwards, surrounding these two arrows, we draw two pseudo-rectangles to represent the endpoints of vertical lines ($A_1B_1C_1D_1$ and $A_2B_2C_2D_2$). The height of these two rectangles are controlled by ε_1, ε_2, ε_1' and ε_2' which are randomly sampled. Afterwards, for the arrows of intra-relations, for simplicity, we only place them, in a randomized manner, above or below these two pseudo-rectangles. The above gives rise to the following group of constraints.

- *Group 3: The y-constraints of inter-relation arrows*. These constraints are partitioned in the following subgroups:

  3.1. Two unsolved y-parameters and their bounds: (1) $y_{1,min} \leq y_{11} \leq 0$; (2) $y_{1,min} \leq y_{22} \leq 0$.

  3.2. Vertical margins for inter-relation arrow at y_{11}: (1) $\Delta y_{11,u} = 0 - (y_{11} + \varepsilon_1) \geq l_1$; (2) $\Delta y_{11,d} = (y_{11} - \varepsilon_2) - y_{1,min} \geq l_2$.

  3.3. Vertical margins for inter-relation arrow at y_{22}: (1) $\Delta y_{22,u} = 0 - (y_{22} + \varepsilon_1') \geq l_1$; (2) $\Delta y_{22,d} = (y_{22} - \varepsilon_2') - y_{1,min} \geq l_2$.

  3.4. Overlap avoidance: $y_{11} + \varepsilon_1 < y_{22} - \varepsilon_2'$.

In Constraints 3.2 and 3.3, l_1 and l_2 are linear functions of the maximum height of the text. These two constraints plus 3.4 are designed to guarantee that, when random sampling is conducted, no overlap will happen, and the layout will be flexible enough.

3) Solving inequality constraints: Groups 1-3 introduce constraints for 18 variables. We use a hit-and-run Markov Chain Monte Carlo library [20] to uniformly sample the solution.

Other Features. Given selected signals, edges, and SPOs with computed key coordinates from the constraints, L-TD-G will randomly plot graphic features (including axes, lines and arrows), and textual annotations (like signal names, upper/lower boundaries, thresholds and timing parameters), according to our empirical analysis of TDs.

V. TD-MAGIC

We now discuss how TD-Magic translates TDs into SPOs by combining object-detection, image-processing, and symbolic-reasoning techniques, through its SED, OCR, LAD, and SEI modules.

SED and OCR Modules. SED and OCR, taking TD picture as input, are DL-based modules trained from synthetic data. Although L-TD-G creates labelled TDs of a simplified form, in Section VI we will show TD-Magic is able to extrapolate and interpret more complex TDs. SED first creates a list $L_B = \{b_1, b_2, ..., b_n\}$, of all edge-boxes it detected. Each box $b \in L_B$ contains the coordinates of the box and the edge type. Next, it sorts L_B according to the box coordinates (top to bottom first, then left to right). Finally, it partitions L_B on a per signal basis. Similarly, OCR creates a list $L_T = \{t_1, t_2, ..., t_n\}$ of all detected text-boxes. A box $t \in L_T$ consists of the text box coordinates, and the recognized string. Next, it sorts L_T the same way as L_B.

LAD Module. LAD uses IP morphological operations, for accurately detecting annotating lines and arrows. LAD first transforms the input TD image into an inverse binary image named *imgBW*. It then applies vertical contour detection on *imgBW*. This operation: (1) Strengthens vertical structures in the image (e.g., turning dashed vertical into solid lines), (2) Filters out all non-vertical elements from the image, and (3) Collects all detected vertical contours with their coordinates into a list L_V. A similar method is applied to extract the list L_H of horizontal contours (either lines or arrows), and their coordinates, too.

SEI Module. SEI provides a semantic interpretation to a TD image. It takes as input the lists L_B and L_T of edge and text boxes, and the lists L_H and L_V of horizontal and vertical lines, respectively. It then finds the appropriate associations and relations between the features in the TD, organizing them into an SPO, returned as the output.

Algorithm 1 Edge-Box-Event Association

```
1: function EdgeBoxEvent(L_B, L_V)
2:     L_{B,E} = {}
3:     for all b ∈ L_B do
4:         L_{b,E} = {}
5:         for all v ∈ L_V do
6:             if v ∩ b ≠ ∅ then
7:                 x = center x-coordinate of v
8:                 y, h = FindHLine(x, b)
9:                 L_{b,E} = L_{b,E} ∪ {((x, y), v)}
10:        if L_{b,E} ≠ ∅ then
11:            L_{B,E} = L_{B,E} ∪ {(b, L_{b,E})}
12:    return L_{B,E}
```

Algorithm 2 Arrow Association

```
1: function ArrowAssociate(L_B, L_V, L_H)
2:     L̂_B = Expand(L_B)
3:     L_{H,V} = {}
4:     for h ∈ L_H do
5:         if not FullSpan(h) and h ∩ b = ∅ for all b ∈ L̂_B then
6:             L_{h,V} = {}
7:             for v ∈ L_V do
8:                 if h ∩ v ≠ ∅ then
9:                     y = center y-coordinate of h ∩ v
10:                    L_{h,V} = L_{h,V} ∪ {(h, v, y)}
11:            L_{H,V} = L_{H,V} ∪ L_{h,V}
12:    L_TC = {}
13:    for L_{h,V} ∈ L_{H,V} do
14:        for (h_1, v_1, y_1) ∈ L_{h,V} do
15:            for (h_2, v_2, y_2) ∈ L_{h,V} do
16:                if h_1 = h_2 and y_1 = y_2 and v_1 ≠ v_2 then
17:                    L_TC = L_TC ∪ {(v_1, v_2, y_1, t)}
18:    return L_TC
```

1) Events association: Every detected edge box can have associated several events. Accordingly, Algorithm 1, Lines 3-11, computes for each box b, a possibly empty set $L_{b,E}$ of events associated to b. An event is represented by a threshold point of an edge box, where a vertical annotation-line crosses a horizontal annotation-line. Hence, for every vertical line that intersects the box edge, Lines 5-9, finds a horizontal line that intersects it. The intersection point is the event (Procedure FindHLine, Line 8) associated to the edge box (Line 9).

2) Arrows Association: Next, SEI identifies horizontal arrows, and infers their timing constraints while building the SPO, as described in Algorithm 2. First (Line 5), it filters out horizontal lines that are not arrows. These lines are either (1) Spanning over the entire signal, or (2) Intersecting or touching an (expanded) edge box. The edge box expansion (see Line 2) is done to facilitate identifying a line that just touches an edge box, a frequent case in horizontal lines corresponding to constant segments of a piecewise-linear signal. Every remaining horizontal line is an arrow candidate. Then, it associates to every candidate, a set of vertical lines that intersect it, together with the y-coordinate of the intersection. This is stored in a set $L_{H,V}$ (see Lines $7-11$). Finally, it searches for horizontal lines that have associated two vertical lines with the same y-coordinate. Each is an arrow, i.e. a timing constraint t that we add to the structure L_{TC}. The algorithm then returns the set L_{TC} of all inferred timing constraints.

3) SPO generation: SEI generates the SPO from L_{TC} and $L_{B,\mathcal{E}}$. They contain all the necessary information about the timing constraints in the TD. SEI constructs a DAG G representing the SPO by:

(1) Associating a node in G to every unique vertical line in L_{TC}, and checking this line's relation with the edge boxes in $L_{B,\mathcal{E}}$, (2) Adding a directed edge from a node v_1 to v_2, if there exists a tuple (v_1, v_2, y_1, t) in L_{TC}. The edge is labeled by the timing constraint t.

VI. Experimental Evaluation

Evaluation on Synthetic Data. Here we evaluate how effective is TD-Magic in learning from synthetic TDs.

1) Data Preparation: (a) For edge detection, we generated 15,000 labelled pictures (size: 3600×2160 for *G1* and *G2*, 2880×1728 for *G3*; resolution: 144 pixels/inch), divided into three groups: *(G1)* Created in default mode, with two signals and two edges; *(G2)* Including only one big signal per picture; *(G3)* With simplified constraints and a special focus on ramp signals. The number of samples in *G1*, *G2* and *G3* is 8000, 4000 and 3000 respectively, with 100 for validation in each. (b) For text detection, the training data consists of 4,000 labeled pictures, and the validation data of 100 labeled pictures: both from *G1*. (c) For text recognition, the training data, of 10,000 cropped pictures, and validation data, of 1,022 cropped pictures, are also from *G1*.

2) Results: We used YOLO5 [21] for edge detection. We conducted three independent experiments, each with 30 epochs and a batch size of 128. Table I shows the overall validation results (mean values over three experiments). We omitted the variance as it is too small (10^{-6}). In Table I, *P* (Precision) represents how well YOLO makes a correct object classification, and *R* (Recall) indicates how many objects are not detected. Both *P* and *R* reach the almost optimal value of 1, meaning that nearly all objects in the validation set are correctly detected, and no objects are missing in detection. The metrics in the last two columns are the mean average precision under different confidence and IoU (Intersection over Union between detected box and labelled box) values. They both approach the best value of 1.

TABLE I: Validation Accuracy of Edge Detection.

Class	Labels	P	R	mAP@.5	mAP@.5:.95
all	1000	0.9990	1	0.995	0.995
riseRamp	388	0.9987	1	0.995	0.995
fallRamp	388	0.9997	1	0.995	0.995
riseStep	79	0.9990	1	0.995	0.995
fallStep	79	0.9963	1	0.995	0.995
double	66	0.9990	1	0.995	0.995

For the two OCR tasks, we conducted three independent experiments using `PaddleOCR` [22], an open source library for common OCR tasks. We trained 30 epochs (with one experiment stopping earlier at the 22nd epoch) for the text detection task, and trained 120 epochs for text recognition. For both OCR tasks, the validation results reach the best accuracy of 1, with negligibly small derivation.

The deep neural network actually overfits on the synthetic examples. This means TD-Magic is very effective in learning from synthetic TDs. Traditionally, over-fitting should be avoided, but if the features are stable enough in both synthetic TDs and extrapolation examples, this phenomenon is beneficial, which will be shown subsequently.

Extrapolation Evaluation. We now evaluate how well TD-Magic extrapolates to industrial TDs. To this end, we evaluated its performance on 30 TDs (29 from industry, and one inspired by industry and manually drawn in Figure 1). All results shown in this subsection are based on the best combination of the trained neural networks.

1) Basic Statistics: For the 30 TDs (size: $3119 \pm 606 \times 1709 \pm 232$; resolution: 144 pixels/inch), 6 (20%), 19 (63.3%), 5 (16.7%) were with 1, 2, and 3 signals respectively. Hence, there are 59 signals, of which, 14 (23.7%), 38 (64.4%), 4 (6.8%), 3 (5.1%) signals have 1, 2, 3 and 4 edges respectively. The last one thus also evaluates TD-Magic's analysis ability on TDs out of synthetic distribution.

2) Object- and OCR-detection evaluation: We first evaluate the performance of object detection. Table II shows the results for all graphical classes. The first five are edges detected by YOLO, and the last three are detected as described in Section V (*V-line* is vertical annotation-line, *H-line* is horizontal annotation-line).

The precision for detecting edges is 1 for all types. On the other hand, as reflected in the minor drop of recall, a small number of edges are not detected, in particular the vertical edges. This might be a consequence of a not large-enough number of vertical edges in the training set. Despite the decrease of recall values, this reduction is in a reasonable range, which still shows the efficiency of TD-Magic in detecting edges of real signals. For *V-line* and *H-line*, as their own detection requires the edge-detection results from YOLO, undetected edges by YOLO will influence their recognition, too. This explains why the recall values for these two objects are not 1. For arrows, both their *P* and *R* value are not 1. This is on the one hand, due to the undetected edges. On the other hand, it is because arrows are the most flexibly annotation in terms of shape and position. The approach proposed in Section V is therefore unable to cover all situations. Despite the imperfect results for lines and arrows, their metrics all exceed 0.9, which proves the efficiency of our approach.

TABLE II: Object Detection Accuracy in Extrapolation.

Metrics	riseRamp	fallRamp	riseStep	fallStep
number	44	42	9	10
P	1	1	1	1
R	0.977	0.976	0.889	0.900
Metrics	**double**	**V-line**	**H-line**	**arrow**
number	8	128	106	84
P	1	1	1	0.951
R	1	0.969	0.972	0.929

Table III illustrates the recognition accuracy for key texts. We only counted totally correct texts. The results show that both the signal names and the signal values enjoy a high accuracy. This shows that our prepared database for common signal names takes effect, and our empirical study on the style of annotating signal values is helpful. For timing constraints, since they are the most flexibly written texts (with subscripts and special symbols) in TDs, their accuracy does not reach 0.9. We claim that a more diverse text set for training purpose will increase the accuracy of OCR.

TABLE III: OCR Accuracy in Extrapolation.

Metrics	Signal Name	Signal Value	Time Constraint
Accuracy	0.915	0.925	0.845

3) Overall-performance evaluation: From the 30 TDs used for extrapolation, the SPOs in 23 (76.7%) pictures can be successfully extracted at a template level. Some minor mistakes in recognition of texts may occur in these structurally correct cases, but no timing constraints are missing or mistakenly extracted. Within these 23 cases, 15 (50.0%) cases are totally correct, at both structural and textual levels. For most of the other 7 (23.3%) TDs, TD-Magic can still partially extract their SPOs. Some common sources of errors are from: (1) Undetected edge boxes. (2) Corner cases. For example: the y-axis is used to represent a partial order; the thresholds are on the boundary of edges; the special position and shape of arrows. Finally, we provide below three examples, illustrating the output of TD-Magic for each. Here rR, fR and fS is used to express riseRamp, fallRamp and fallStep. The detected edge boxes, V-lines, H-lines, arrows are in grey, blue (also for wrong texts), red and green.

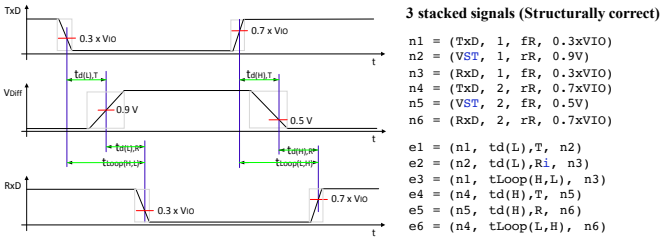

Fig. 6: Example 1 (input: original; plot: redrawn) [23]

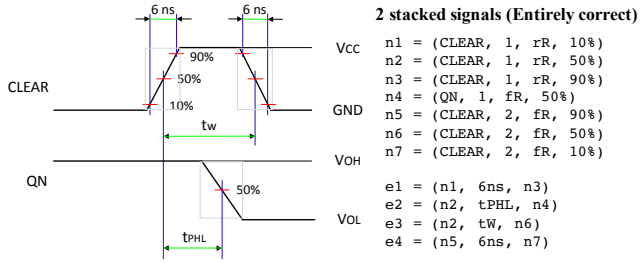

Fig. 7: Example 2 (input: original; plot: redrawn) [24]

Example 3: Input is the original picture of Figure 4 (left). Since the vertical lines are nearly as thick as the step edges, it may confuse TD-Magic, so the tool returns the structurally incorrect SPO where `n2`, `n3` should be `n3`, `n4`. OCR also returns `IO0%` instead of `10%`.

```
n2 = (VINA, 1, fS, None)
n3 = (VOUTA, 2, fR, IO0%)    e1 = (n2, tD(Off), n3)
```

We also tested original versions of Figures 1, 4 (right). TD-Magic made a minor mistake in the former, while the results are all correct in the latter. Example 1 demonstrates that TD-Magic can extrapolate to three or more signals, although the synthetic data-set provided at most two signals in one picture, only. This is because the YOLO-network focuses on the shape of edges, and not on the number of signals. The successful analysis of Example 2 shows that TD-Magic is able to detect edges with dense thresholds (with more than two key values). This is because the design of L-TD-G pays special attention to this kind of annotation. One notable observation from Example 2 is that although the shape of the left and right arrows delimiting annotation "6ns" is not supported by our generator, TD-Magic is able to extrapolate, and properly detect them. This demonstrates the efficiency of the SEI module.

The failure of Example 3 to extract the entire PSO of the TD picture in Figure 4 (left), is because YOLO does not successfully detect the first vertical edge in Signal V_{INA} as shown in Figure 4. To solve this problem, more vertical edges should be provided to the training set. Providing more training data is also an effective method to increase the accuracy of OCR. We leave this to future work.

VII. Conclusion

We considered the open problem of translating bitmap TD-pictures into formal specifications. We tamed the lack of publicly available labelled TDs, necessary for training object detection techniques, by developing L-TD-G, a synthetic data generator producing pictures of realistic industrial timing diagrams, annotated with their corresponding formal specification. We then designed TD-Magic, an automatic approach combining object-detection techniques, OCR, and image-processing algorithms, to extract the relevant features from TD-pictures, and construct their semantic interpretation as a formal specification. Our experimental results on industrial TDs, found in publicly available products' data sheets, are very encouraging, and demonstrate a high level of accuracy. As future work, we plan to extract formal specifications from more complex timing diagrams with mixed analog-digital signals.

Acknowledgment

This project has received funding from the Austrian FFG ICT of the Future program under grant agreement No 880811, and funding from the European Key Digital Technologies Joint Undertaking (KDT JU) under grant agreement No 101097300.

References

[1] P. Rony, "Interfacing fundamentals: Timing diagram conventions," *Computer Design*, vol. 19, no. 1, pp. 152–153, 1980.
[2] O. M. G. (OMG), "Unified modeling language: Superstructure, v2.0," https://www.omg.org/cgi-bin/doc?formal/05-07-04, 2004, online; accessed 05 August 2022.
[3] F. Kathi, "Timing diagrams: Formalization and algorithmic verification," *J. Log. Lang. Inf.*, vol. 8, no. 3, pp. 323–361, 1999.
[4] R. Schlor, "A prover for vhdl-based hardware design," in *Proc. of ASP-DAC'95/CHDL'95/VLSI'95 with EDA Technofair*, 1995, pp. 643–650.
[5] N. Amla, E. A. Emerson, and K. S. Namjoshi, "Efficient decompositional model checking for regular timing diagrams," in *Proc. of CHARME*, ser. LNCS, vol. 1703. Springer, 1999, pp. 67–81.
[6] N. Ortigosa and V. M. Giménez, "Raw data extraction from electrocardiograms with portable document format," *Comput Methods Programs Biomed*, vol. 113, no. 1, pp. 284–289, 2014.
[7] D. Chung, J. Choi, J.-H. Jang, T. Y. Kim, J. Byun, H. Park, H.-S. Lim, R. W. Park, and D. Yoon, "Construction of an electrocardiogram database including 12 lead waveforms," *JHIR*, vol. 24, no. 3, pp. 242–246, 2018.
[8] G. S. Waits and E. Z. Soliman, "Digitizing paper electrocardiograms: Status and challenges," *J. Electrocardiol.*, vol. 50, no. 1, pp. 123–130, 2017.
[9] L. Ravichandran, C. Harless, A. J. Shah, C. A. Wick, J. H. Mcclellan, and S. Tridandapani, "Novel tool for complete digitization of paper electrocardiography data," *IEEE J. Transl. Eng. Health Med.*, vol. 1, p. 1800107, 2013.
[10] M. Baydoun, L. Safatly, O. K. Abou Hassan, H. Ghaziri, A. El Hajj, and H. Isma'eel, "High precision digitization of paper-based ECG records: a step toward machine learning," *IEEE J. Transl. Eng. Health Med.*, vol. 7, pp. 1–8, 2019.
[11] X. Sun, Q. Li, K. Wang, R. He, and H. Zhang, "A novel method for ECG paper records digitization," in *Proc. of CinC*. IEEE, 2019, pp. 1–4.
[12] S. Ganesh, "Novel method of digitization of electrocardiogram signals," Ph.D. dissertation, Georgia Institute of Technology, 2020.
[13] S. Mishra, G. Khatwani, R. Patil, D. Sapariya, V. Shah, D. Parmar, S. Dinesh, P. Daphal, and N. Mehendale, "ECG paper record digitization and diagnosis using deep learning," *JMBE*, vol. 41, no. 4, pp. 422–432, 2021.
[14] Y. Li, Q. Qu, M. Wang, L. Yu, J. Wang, L. Shen, and K. He, "Deep learning for digitizing highly noisy paper-based ECG records," *Computers in biology and medicine*, vol. 127, p. 104077, 2020.
[15] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. of IEEE ICCV*. IEEE, 2017, pp. 2961–2969.
[16] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Proc. of ECCV 2020*, 2020, pp. 213–229.
[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. of IEEE CVPR*. IEEE, 2016, pp. 779–788.
[18] "Example of timing diagram similar to Figure 6 (on page 15)," https://www.st.com/resource/en/datasheet/vnh5050a-e.pdf.
[19] "Example of timing diagram similar to Figure 9 (on page 13)," https://www.st.com/resource/en/datasheet/m74hc595.pdf.
[20] "anyhr," https://github.com/figlerg/anyHR.
[21] "Yolo5," https://github.com/ultralytics/yolov5.
[22] "Paddleocr," https://github.com/PaddlePaddle/PaddleOCR.
[23] "Example of timing diagram similar to Figure 31 (on page 43)," https://www.infineon.com/dgdl/Infineon-TLE9252V-DataSheet-v01_11-EN.pdf?fileId=5546d462602a9dc80160e0c3c92f5346.
[24] "Example of timing diagram similar to Figure 6 (on page 9)," https://www.st.com/resource/en/datasheet/m74hc4060.pdf.