Syntroids

Synthesizing a Game for FPGAs using Temporal Logic Specifications

Gideon Geier, Philippe Heim, Felix Klein and Bernd Finkbeiner
24th October 2019

Saarland University
What requirements should the system meet?

Push the button!

Get a correct system

Reactive Synthesis

temporal specification $\varphi$

realizable + Model $\mathcal{M}$, with $\mathcal{M} \models \varphi$

unrealizable
Reactive Synthesis

In theory

*Synthesized systems are “correct” by design and therefore nice.*

In practice

*Milestone: Synthesis of AMBA AHB*


But still not much used.

⇒ Synthesis is not suited for real world applications?
In our case study we present

**Syntroids**

a game for FPGAs synthesized with

**Temporal Stream Logic (TSL)**

as specification language
radar mode

cockpit mode

score mode
Architecture

FPGA

32×32 LED matrix

Sensor IO

Game logic

LED matrix Output

Accelerometer + Gyroscope

Syntroids, Philippe Heim, Gideon Geier
Specification language – LTL?

$LTL : \varphi := a \mid \varphi \land \varphi \mid \neg \varphi \mid O \varphi \mid \varphi U \varphi \mid \Diamond \varphi \mid \Box \varphi \mid \varphi R \varphi \mid \cdots$

where $a \in AP$

But we have to . . .

- manage 16-bit Sensor values using SPI
- integrate these to get absolute positions
- control a 1024-pixel LED-matrix
- work with Cartesian- and Polar-coordinates

$\implies$ LTL is not suited for the Syntroids game
“Everytime the sensor input is greater than 100, we want to increase our score by one.”

\[ ("100 < \) \text{sensor} \rightarrow [\text{score} \leftarrow "1 + " \text{score}] \]

- sensor, score are data streams
- "100 < " is a predicate literal
- "1 + " is a function literal
- Important: The synthesis tool does not know the implementation (which may be defined after the synthesis).
- \([\text{score} \leftarrow "1 + " \text{score}]\) is an update
"Everytime the sensor input is greater than 100, we want to increase our score by one."

\[
\square ("100 < \) \text{sensor} \rightarrow [\text{score} \leftarrow "1 + " \text{score}])
\]
Temporal Stream Logic (TSL)

\[ TSL \ni \varphi, \psi ::= \top | \varphi \land \psi | \neg \varphi | \Diamond \varphi | \varphi U \psi | [s_0 \leftarrow T_f] | T_p \]

- TSL works with arbitrary data streams
- TSL specifies the control structure
- Data manipulations and predicates are given manually, but implementation is not needed for the synthesis

\(\square([\text{out} \leftrightarrow \text{scoreboardpoint}] \lor [\text{out} \leftrightarrow \text{radarboardpoint}] \lor [\text{out} \leftrightarrow \text{cockpitboardpoint}])\)
\(\land \square(\text{gameover} \rightarrow [\text{out} \leftrightarrow \text{scoreboardpoint}])\)
\(\land \square(\neg \text{gameover} \rightarrow ([\text{isScoreMode gamemode} \rightarrow [\text{out} \leftrightarrow \text{scoreboardpoint}]) \land ([\text{isRadarMode gamemode} \rightarrow [\text{out} \leftrightarrow \text{radarboardpoint}]) \land ([\text{isCockpitMode gamemode} \rightarrow [\text{out} \leftrightarrow \text{cockpitboardpoint}]))\)

\[\text{GameModule}\]

- cockpitboardpoint
- radarboardpoint
- outpoint

Syntroids, Philippe Heim, Gideon Geier
\( \square((\bigcirc[^{\color{red} \text{color}_1 \leftarrow \text{ramout}]}) \to[^{\color{red} \text{rampos} \leftarrow \text{"rampos}_1" \ \text{coord}_x ("1 + " \ \text{coord}_y)]}) \wedge \square([^\text{extclock} \leftarrow \text{low}()] \to ([^\color{red} \text{color}_1 \leftarrow \text{ramout}] \ \mathcal{R} \neg[^\text{extclock} \leftarrow \text{high}()])) \)
\[
\begin{align*}
& [\text{extclock} \leftrightarrow \text{low()}] \\
& \land (\Diamond \lozenge [\text{extclock} \leftrightarrow \text{high()}]) \\
& \land (\Diamond \lozenge [\text{extclock} \leftrightarrow \text{low()}])
\end{align*}
\]
And now both together

\[
\begin{align*}
&(□◇ [\text{extclock} \leftarrow \text{high()}]) \\
\land (□◇ [\text{extclock} \leftarrow \text{low()}]) \\
\land (□◇ [\text{coord}_x \leftarrow "1 + " \text{coord}_x]) \\
\land (□◇ [\text{color}_R \leftarrow \text{ramout}]) \\
\land (□((□ [\text{color}_1 \leftarrow \text{ramout}]) \\
\quad \land \quad [\text{rampos} \leftarrow "\text{rampos}_1" \text{coord}_x ("1 + " \text{coord}_y)]) \\
\land (□([\text{extclock} \leftarrow \text{low()}]) \\
\land (□([\text{extclock} \leftarrow \text{high()}])) \\
\land (□([\text{coord}_x \leftarrow "1 + " \text{coord}_x]) \\
\quad \land \quad \emptyset ([\text{extclock} \leftarrow \text{high()}]) \Rightarrow [\text{coord}_x \leftarrow "1 + " \text{coord}_x]) \\
\end{align*}
\]

\(\Rightarrow\) Each property is simple but the resolve complicated
Same specification with different function implementations
Assumptions

GameModule

\[ \text{Gamemode} := \{ \text{ScoreMode, RadarMode, CockpitMode} \} \]

\[ \square \neg (\text{isScoreMode gamemode} \land \text{isRadarMode gamemode}) \]

SPIReadManage

\[ \square \neg ("0 = " \text{counter} \land "18 < " \text{counter}) \]

LedMatrix

\[ \text{waitcounter} \in \mathbb{B}^7, \mathbb{N}_{<128} \]

\[ (\square \Diamond [\text{waitcounter} \leftrightarrow "1 + " \text{waitcounter}]) \]

\[ \rightarrow \square \Diamond ("0 \neq " \text{waitcounter}) \]
Synthesis Toolchain

TSL

CFM

Project Context

Compiler

Verilog

LTL

Controller

FRP (CλaSH)

Design Pattern

FRP Library (CλaSH)

Functions and Predicates

maybe unrealizable

LTL Synthesis Tool

TSL Tools

Synthesis Toolchain

Syntroids, Philippe Heim, Gideon Geier
LTL Approximation

- TSL synthesis undecidable in general
- Use LTL under-approximation
- LTL approximation is sound, i.e. every time we get a synthesized LTL approximation result, this is a valid TSL synthesis result
LTL Approximation

\[ \square ("100 < " \text{sensor} \rightarrow [\text{score} \leftrightarrow "1 + " \text{score}]) \]

Atomic propositions encoding

- \( p_1 : "100 < " \text{sensor} \)
- \( u_1 : [\text{score} \leftrightarrow "1 + " \text{score}] \)
- \( u_2 : [\text{score} \leftrightarrow \text{score}] \)

\[ (\square(p_1 \rightarrow u_1)) \land (\square(u_1 \lor u_2)) \land (\square \neg (u_1 \land u_2)) \]

\[ \implies \text{Encode updates and predicate evaluations in atomic propositions not the data} \]
Lines of Code

Handmade Game

- About 1000 lines of code

Synthesized Game

- 194 guarantees and 40 assumptions
- About 200 lines of code functions, predicates and datatypes
- About 300 lines of code for the composition (tool support for generation missing)
- About 7400 lines of generated code
Synthesis Toolchain

TSL

CFM

TSL Tools

Project Context

Compiler

Verilog

maybe unrealizable

LTL Synthesis Tool

FRP (CλaSH) Design Pattern

FRP Library (CλaSH)

Functions and Predicates

Syntroids, Philippe Heim, Gideon Geier
Hardware Setup

- icoBoard
- iCE40 FPGA
  - 7680 logic cells
- PmodNAV sensor
- accelerometer & gyroscope
- Custom board to connect FPGA and LED matrix
Circuit size

LCs limit

handmade
completely synthesized
Sensor IO
Game logic
LED matrix Output

Syntroids, Philippe Heim, Gideon Geier
## Synthesis results

<table>
<thead>
<tr>
<th>Module</th>
<th>G</th>
<th>A</th>
<th>Bosy Time</th>
<th>Lat</th>
<th>Gat</th>
<th>Bowser Time</th>
<th>Lat</th>
<th>Gat</th>
<th>Strix Time</th>
<th>Lat</th>
<th>Gat</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionConverter (AC)</td>
<td>4</td>
<td>0</td>
<td>0.316</td>
<td>1</td>
<td>8</td>
<td>4.384</td>
<td>0</td>
<td>4</td>
<td>1.192</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Cockpitboard (CB)</td>
<td>12</td>
<td>0</td>
<td>1548.48</td>
<td>1</td>
<td>11</td>
<td>227.136</td>
<td>0</td>
<td>7</td>
<td>6.512</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>EnemyModule (EM)</td>
<td>4</td>
<td>0</td>
<td>0.272</td>
<td>1</td>
<td>6</td>
<td>0.904</td>
<td>0</td>
<td>2</td>
<td>1.196</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>GameLogic (GL)</td>
<td>21</td>
<td>0</td>
<td>&gt; 99999</td>
<td>-</td>
<td>-</td>
<td>13230.5</td>
<td>2</td>
<td>226</td>
<td>696.288</td>
<td>2</td>
<td>29</td>
</tr>
<tr>
<td>GameModeChooser (GC)</td>
<td>7</td>
<td>4</td>
<td>111.072</td>
<td>1</td>
<td>100</td>
<td>13247.0</td>
<td>0</td>
<td>2377</td>
<td>2.164</td>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>GameModule (GM)</td>
<td>3</td>
<td>3</td>
<td>0.328</td>
<td>1</td>
<td>10</td>
<td>1.056</td>
<td>0</td>
<td>3</td>
<td>1.288</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>LedMatrix (LM)</td>
<td>27</td>
<td>1</td>
<td>&gt; 99999</td>
<td>-</td>
<td>-</td>
<td>&gt; 99999</td>
<td>-</td>
<td>-</td>
<td>53732.0</td>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>Radarboard (RB)</td>
<td>13</td>
<td>0</td>
<td>41319.2</td>
<td>1</td>
<td>10</td>
<td>26.448</td>
<td>0</td>
<td>6</td>
<td>79.376</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>RegisterManager (RM)</td>
<td>5</td>
<td>0</td>
<td>0.292</td>
<td>1</td>
<td>4</td>
<td>0.164</td>
<td>0</td>
<td>0</td>
<td>1.084</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RotationCalculator (RC)</td>
<td>5</td>
<td>3</td>
<td>1.324</td>
<td>1</td>
<td>18</td>
<td>8045.37</td>
<td>0</td>
<td>9</td>
<td>1.66</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>SPI (SPI)</td>
<td>15</td>
<td>2</td>
<td>&gt; 99999</td>
<td>-</td>
<td>-</td>
<td>13214.7</td>
<td>3</td>
<td>413</td>
<td>3.608</td>
<td>3</td>
<td>72</td>
</tr>
<tr>
<td>SPIReadClk (SPI_R)</td>
<td>2</td>
<td>0</td>
<td>0.272</td>
<td>1</td>
<td>4</td>
<td>0.348</td>
<td>0</td>
<td>2</td>
<td>1.168</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>SPIReadManag (SPI_R)</td>
<td>11</td>
<td>2</td>
<td>3497.26</td>
<td>1</td>
<td>31</td>
<td>11821.0</td>
<td>1</td>
<td>10</td>
<td>14.684</td>
<td>2</td>
<td>27</td>
</tr>
<tr>
<td>SPIReadSdi (SPI_R)</td>
<td>2</td>
<td>2</td>
<td>0.304</td>
<td>1</td>
<td>5</td>
<td>0.844</td>
<td>0</td>
<td>1</td>
<td>1.36</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>SPIWriteClk (SPI_W)</td>
<td>2</td>
<td>0</td>
<td>0.276</td>
<td>1</td>
<td>6</td>
<td>3.628</td>
<td>0</td>
<td>4</td>
<td>1.196</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>SPIWriteManag (SPI_W)</td>
<td>9</td>
<td>4</td>
<td>196.808</td>
<td>1</td>
<td>6</td>
<td>61.704</td>
<td>1</td>
<td>6</td>
<td>2.22</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>SPIWriteSdi (SPI_W)</td>
<td>3</td>
<td>5</td>
<td>0.396</td>
<td>1</td>
<td>15</td>
<td>12.536</td>
<td>0</td>
<td>4</td>
<td>1.3</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>Scoreboard (SB)</td>
<td>7</td>
<td>0</td>
<td>1.26</td>
<td>1</td>
<td>8</td>
<td>15.576</td>
<td>0</td>
<td>4</td>
<td>1.516</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Sensor (Sen)</td>
<td>6</td>
<td>4</td>
<td>7429.74</td>
<td>2</td>
<td>29</td>
<td>&gt; 99999</td>
<td>-</td>
<td>-</td>
<td>1.912</td>
<td>4</td>
<td>70</td>
</tr>
<tr>
<td>SensorInit (Sen)</td>
<td>14</td>
<td>0</td>
<td>159.076</td>
<td>4</td>
<td>95</td>
<td>6613.8</td>
<td>4</td>
<td>84</td>
<td>3.676</td>
<td>4</td>
<td>46</td>
</tr>
<tr>
<td>SensorPart (Sen)</td>
<td>18</td>
<td>0</td>
<td>1985.21</td>
<td>3</td>
<td>34</td>
<td>12224.9</td>
<td>3</td>
<td>64</td>
<td>13.864</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>SensorRegister (RM)</td>
<td>1</td>
<td>0</td>
<td>0.292</td>
<td>1</td>
<td>2</td>
<td>0.048</td>
<td>0</td>
<td>0</td>
<td>1.188</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SensorSelector (SS)</td>
<td>5</td>
<td>4</td>
<td>&gt; 99999</td>
<td>-</td>
<td>-</td>
<td>37.884</td>
<td>0</td>
<td>0</td>
<td>277.288</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>SensorSubmodulChooser (Sen)</td>
<td>5</td>
<td>6</td>
<td>766.084</td>
<td>2</td>
<td>44</td>
<td>13007.8</td>
<td>2</td>
<td>369</td>
<td>3.176</td>
<td>3</td>
<td>39</td>
</tr>
</tbody>
</table>
Experience

- Synthesis yields usable results
- TSL allows . . .
  - to describe behavior effectively
  - to work on arbitrary data
  - to build parametric systems
  - to extend modules easily
- Better tool support still needed
  - Automatically compose and instantiate modules
  - Analyze or prove global properties, as manual composition might lead to mistakes
  - Debugging still needs some work
Unrealizable Specifications

- Helpful as we know that there is a mistake
- Long synthesis times make finding conflicts difficult
- Not all tools yield counterstrategy
- Lack of tools to analyze counterstrategies

“Faulty” (realizable) Specifications

- Assumption violations difficult to find
- In this case unit testing is necessary
Conclusion

- Development using Reactive Synthesis is possible
- Reactive Synthesis is a promising technology
- More real world applications & experience needed

All code available at:

www.react.uni-saarland.de/casestudies/syntroids/

Play Syntroids at the poster session