Jihočeská univerzita v Českých Budějovicích

Přírodovědecká Fakulta

OVĚŘENÍ MULTIPLATFORMNOSTI WINDOWS 10 NA POČÍTAČI RASPBERRY Pi 2

Bakalářská práce

Petrásek Jan

Školitel: Mgr. Jiří Pech, Ph.D.

České Budějovice 2016

Bibliografické údaje

Petrásek, J., 2016: Ověření multiplatformnosti Windows 10 na počítači Raspbery Pi2. [Verification of multiplatformity of Windows 10 on computer Raspberry Pi2. Bc. Thesis, in Czech.] – 70p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Abstrakt

Bakalářská práce se týká Windows 10 IoT Core a ověření jejich funkčnosti na platformě Raspberry Pi 2. Je ověřen postup instalace systému a jeho prvotní konfigurace. Je popsána problematika řízení systému pomocí Webserveru a PowerShellu 5. Pro ověření vlastností systému jsem využil testy práce s GPIO sběrnicí a testy simulují reálné IoT scénáře.

Abstract

This bachelor thesis concerns the Windows 10 IoT Core and verification of its functionality on the platform Rapsberry Pi 2. The process of installation and initial configuration is validated. The system management using PowerShell 5 or webserver is described. To verify the properties of operating system I tested the performance using GPIO bus and tests simulating realistic IoT scenarios.

Klíčová slova

Windows 10, IoT, Raspberry Pi 2, Visual Studio 2015, Windows Universal Apps.

Key words

Windows 10, IoT, the Raspberry Pi 2, Visual Studio 2015, Windows Universal Apps.

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne……

Podpis: ……

Petrásek Jan

Poděkování

Tímto bych rád poděkoval školiteli Mgr. Jiřímu Pechovi Ph.D. za odbornou pomoc při zpracování mé bakalářské práce.

Také chci poděkovat Janu Pospíšilovi (Technickému evangelistovi Microsoftu ČR) za odborné poradenství v oblasti operačního systému a psaní aplikací pro tento systém a Michaelu Grafnetterovi (nezávislému IT konzultantovi pracujícím pro Microsoft, Gopas a Univerzitu Karlovu) za odborné konzultace v oblasti Windows PowerShell.

V neposlední řadě chci také poděkovat mím blízkým, kteří se mnou po celou dobu studií měli trpělivost.

Obsah

1. ÚVOD

Téma své bakalářské práce jsem si zvolil na základě zkušeností s testováním Windows 10 pro domácí počítače a zajímalo mě, jestli se Microsoftu podaří s Windows proniknout i na zařízení typu Raspberry. Jednodeskové počítače jako je Raspberry Pi2 nebo MinnowBoard Max jsou v dnešní době velmi oblíbené pro své velké možnosti, což si uvědomují i v Microsoftu. Do příchodu Windows 10 IoT ovládaly tento sektor různé Linuxové distribuce. Na jednoúčelových zařízeních jako jsou bankomaty, či pokladní a informační systémy se snažil Microsoft držet krok pomocí systému řady embedded. Tato řada nikdy nebyla široce dostupná vývojářům a bylo nutné objednávat hodně licencí, které stály od 100 dolarů výše.

Rodina systémů Windows 10 představuje první opravdu multiplatformní operační systém z dílen Microsoftu, ale tomu tak? Když se podíváme na vývoj systémů Windows, tak jejich hlavní doménou jsou osobní počítače a servery. Na osobních počítačích je již k dispozici finální verze Windows 10, která je průběžně vylepšována a pro servery vyšla finální verze na konci září 2016. Mezi mobilními telefony se od verze Windows Phone 8.1 podařilo Microsoftu systém rozšířit i mimo modelovou řadu Lumia a vybrané modely HTC, a i zde je již k dispozici jak vývojová, tak finální verze systému Windows 10 Mobile, která se postupně připravuje pro jednotlivé podporované modely telefonů. Novinkou je však verze pro IoT, kde Microsoft sice není úplně poprvé, ale jeho Embedded verze Windows se příliš nerozšířily. Odpověď na otázku, zda se Microsoftu opravdu podařilo přijít s konkurence schopným IoT systémem a dosáhnout tak plné multiplatformnosti rodiny Windows 10 jsem se pokusil zodpovědět v této práci.

V první části práce je vytvořena podrobná instalační metodika systému. Jsou zde uspořádány informace z různých zdrojů Microsoftu doplněné o mé vlastní postřehy. Jsou zde uvedeny všechny SW a HW požadavky pro instalaci systému i pro jeho běh. Celá instalace a konfigurace systému je popsána podrobně krok za krokem, stejně jako využití nástrojů pro vzdálené ladění ve Visual studiu i instalace aplikací na tento systém. V tomto bloku také udám přesnou specifikaci systému a HW vybavení, na kterém probíhali testy systému. Je to z toho důvodu, že některé výsledky testů se mohou s využitím jiného obrazu systému výrazně lišit.

V druhé části jsou popsány vlastní testy provedené na systému a jejich vyhodnocení. Součástí testování je i nasazení systému v některých vzorových příkladech (tzv. Hands on

labs) a vyhodnocení provedení požadovaných úkonů. Protože jde o IoT systém, který komunikuje s externím HW, jako jsou snímače, převodníky apod. pomocí GPIO sběrnice a I^2C sběrnice, tak je do testování systému též zařazen i test, který zkoumá odezvu GPIO sběrnice z pohledu externího zařízení. Závěrečné vyhodnocení testů je inspirováno modelem kvality SW FURPS od společnosti HP, o kterém si myslím, že vystihuje všechno potřebné pro účely této práce.

2. CÍLE A METODIKA PRÁCE

2.1. CÍLE PRÁCE

Jedním z cílů mé bakalářské práce je na základě praktické zkušenosti s používáním systému a studia oficiálních zdrojů vytvořit ucelený zdroj informací, který poskytne uživateli vše potřebné, co potřebuje vědět pro nasazení systému Windows 10 IoT Core ve svém projektu. Jde o požadavky systému, jeho charakteristické rysy, ale především o postupy umožňující jeho reálné nasazení a následnou správu.

Druhým z cílů je na testech prokázat jak kvality systému, tak jeho nedostatky. Hlavním úkolem praktické části je názorně ukázat, kdy je Windows 10 IoT dobrou volbou pro nasazení na projektu a kdy by jeho volba mohla s sebou přinést jisté komplikace. Ze získaných poznatků a dat následně vytvořit souhrnné resumé o vlastnostech systému.

2.2. METODIKA PRÁCE

První část práce je vypracována na základě studia oficiálních informací na webu Microsoftu a studia záznamu z konference „Otevřené Widle“. Následně jsou jednotlivé kroky ověřeny a podrobně popsány. Pro větší názornost jsou součástí popisů screen’s jednotlivých kroků a vlastní postřehy. Po dokončení instalace a konfigurace na jednoduchém příkladu ukáži Windows Universal Apps, jak ji ladit a následně nainstalovat na Windows 10 IoT Core.

Druhá část je vedena na základě studia odborné literatury z oblasti testování SW, podkladů z certifikačního školení řízení jakosti SW a zkušeností s testováním několika generací systémů Windows pro stolní PC. Postupným provedením jednotlivých testů a jejich vyhodnocením je získána objektivní představa o vlastnostech systému.

3. Nasazení Windows 10 IoT Core na Raspberry Pi2

3.1. O systémech Windows 10 IoT

„Windows 10 IoT patří do rodiny Windows 10, která je zaměřená na široký okruh inteligentních přístrojů, od malých průmyslových bran k větším, komplexnějším přístrojům jako jsou místa prodejních terminálů a ATMs. Kombinace nejnovějšího vývoje nástrojů Microsoftu a partnerů služeb Azure IoT umožňuje shromažďovat, ukládat a zpracovávat data, vytvářet obchodní inteligenci a zefektivnit obchodní výnosy. Řešení založené na této spolupráci, jenž se opírá o Windows 10 IoT, zajistí rozšiřující se možnosti s využitím plné šíře technologií Microsoftu tak, aby byla nabídnuta řešení na principu „end-to-end“.

Windows 10 IoT Enterprise přináší schopnosti Windows 10 Enterprise do širokého spektra průmyslových přístrojů od maloobchodu, přes výrobou, oblast zdraví, financí až k dalším průmyslovým odvětvím. Přístroje Windows 10 IoT Enterprise řídí řadu obchodních aplikací a vykonávají specializované funkce bezpečným, spolehlivým a aerodynamickým způsobem tak, aby podpořili mise rozhodujících průmyslových přístrojů. Windows 10 IoT Enterprise podporuje jak aplikace Universal Windows, tak Classic Windows a mnoho dalších inovativních schopností: pokročilá ochrana proti moderním bezpečnostním hrozbám, plná flexibilita v rozmístění, updating a podpora možností, pokročilá infrastruktura, přístroje a rysy app managementu. Společnosti, které požadují ovladatelnost, jednotnost a předpověditelnost si vybírají Windows 10 IoT Enterprise a Windows 10 Enterprise.

Windows 10 IoT Mobile Enterprise přináší schopnosti Windows 10 Mobile Enterprise do tzv.“lone-of business“ mobilních přístrojů, které spojuje přirozenou uživatelskou zkušenost se zabezpečením podnikového stupně a dobrou ovladatelností. Okamžitý přístup k aplikaci, automatická podpora skenování čárového kódu a další periferní zařízení stejně jako pocit bezpečnosti přístroje zvyšují produktivitu pro mnoho mobilních scénářů. Mobilní edice Enterprise také nabízí další schopnosti jako např. profil „multiple user“ a pokročilé uzamčení, tak aby bylo možné užívat řadu různých mobilních scénářů od maloobchodu, přes zdravotní péči, výrobu až k dalším vertikálním odvětvím.

Windows 10 loT Core je verze Windows 10, která je optimalizována pro menší a nízkonákladová průmyslová zařízení. Jelikož jsou vytvořena tak, aby řídila přístroje jako IoT brány nebo informační stánky, je určena k řízení jedné obchodní aplikace. Windows 10 loT

Core řídí aplikaci Universal Windows a využívá stejný vývoj, konfiguraci a stejné řídící nástroje jako ostatní edice Windows 10 a tím usnadňuje integraci do IoT scénářů a maximální využití stávajících zdrojů.“ *Windows 10 for the Internet of Your Things* [online]. [cit. 2015-11-13]. Dostupné z: https://www.microsoft.com/en-us/WindowsForBusiness/windows-iot (přeloženo)

3.2. Požadavky Windows 10 IoT Core

Systém je instalován obdobně jako většina jeho Linuxových konkurentů ze stolního počítače, což sebou přináší rozdílné požadavky pro běh a pro instalaci systému. Pro běh systému ve verzi RTM je požadován jednodeskový počítač Raspberry Pi 2 a Pi 3, nebo MinnowBoard MAX, DragonBoard 410c ve verzi Insider Preview je krom již zmiňovaných počítačů podporované i Intel Joule (stav k 28.10.2016). Jako systémový disk v tomto případě slouží microSD karta, která musí být minimálně class 10 (či U1) s minimální kapacitou 8 GB, na mnoho aplikací je však doporučena karta o velikosti 16 GB. Jednodeskový počítač je potřeba připojit k domácí síti (nemusí mít přístup k internetu) pomocí Ethernetu, nebo pomocí Wi-Fi. Pokud se rozhodneme využít Wi-Fi, tak na Raspberry Pi je potřeba oficiální USB modul (platí pro obrazy starší, než 10.0.10558, tento a novější už mají širší podporu). Na rozdíl od Wi-Fi modulů je podpora Bluetooth modulů mnohem lepší, zde neplatí žádná omezení. Osobně doporučuji alespoň pro prvotní konfiguraci připojit i klávesnici a myš. Pokud jde o zobrazovací jednotky, tak i v případě Raspberry doporučuji užít jednotku připojenou pomocí HDMI portu (od verze 10.0.10558 je již možné použít originální display Raspberry). Z vlastní zkušenosti mohu říct, že Display port na staršíchu buildech funguje jen se svou vlastní ovládací aplikací, která pak umožňuje zprovoznit i dotykové ovládání, ale za cenu toho, že pokud chci spustit nějakou uživatelskou aplikaci, tak je nutné ji integrovat do aplikace pro ovládání displeje, to samé platí i pro zobrazovací jednotky připojené pomocí GPIO sběrnice.

Pokud jde o počítač, který bude provádět instalaci systému, jsou na něj kladeny jiné nároky, nežli jsou potřebné pro běh systému. Instalující počítač musí mít nainstalované Windows 10 (od verze Windows 10 IoT 10.0.10556 je možné instalaci provést ze Systému Windows 8.1 a Windows server 2012 R2) v jakékoli edici a je potřeba čtečka paměťových karet pro formát MicroSD, tedy buď USB čtečka, nebo integrovaná čtečka SD a adaptér. Pokud jde o samotný Windows 10 instalovaný na počítači, tak nezáleží na tom, zda bude ve verzi x86 nebo x64, protože instalační nástroje jsou 32 bitové. Bohužel není možné užít starší verzi systému, ale pokud má někdo testovací verzi systému Windows Server 2016 TP3

a novější na svém počítači, tak i z tohoto systému je možné instalaci provést. Pro ty, kteří potřebují provést upgrade ze starší verze systému, připomínám v následující tabulce požadavky systému Windows 10. Co není nikde uvedeno, ale osobně mohu doporučit, je mít na počítači instalovaný 7 zip, Winrar, či jiný nástroj pro práci s archivy. Tento nástroj se bude hodit především pro přechod na nový obraz systému Windows 10 IoT Core.

Položka	64 bitová edice		32 bitová edice	
	Minimální	Doporučené	Minimální	Doporučené
Volné místo na HDD	20 GB	30 GB	16 GB	20 GB
Velikost RAM	2 GB	4 GB	1 GB	2 GB
Procesor	1 GHz kompatibilní s Windows 8			
Síťová karta	Wi-Fi karta, nebo standartní 100 Mb, či rychlejší Ethernet			
Grafická karta	Podpora DirectX 9			

Tabulka 1: HW požadavky Windows 10 pro stolní počítač

Požadavky na PC, které bude vytvářet a překládat aplikace určené pro Windows 10 IoT Core jsou bohužel vyšší nežli minimální požadavky pro instalaci. Toto navýšení vyplývá z toho, že aplikace pro Windows 10 IoT Core jsou shodné jako pro všechny ostatní systémy rodiny Windows 10, a je potřeba použít Visual Studio 2015. Pro slabší zařízení je alternativou Visual Studio Code, které je dostupné i pro Linux a MacOS, ale nemám s tím zkušenosti a oficiální informace o této verzi Visual studia nemluví v souvislosti s Windows 10 IoT Core. Doporučuji využít pro programování aplikací Visual studio Prosfessional, ač je placené[1]. Srovnání jednotlivých edic Visual Studia 2015 je dostupné na oficiálním webu aplikace[2]. Do většiny edic Visual Studia jde nainstalovat český jazykový balíček. Pro lepší přehlednost uvádím požadavky na systém a hardware v následující tabulce.

Položka	Minimální požadavek
Volné místo na HDD	10 GB
RAM	1 GB
Procesor	1,6 GHz
Grafická karta	DirectX 9 s rozlišením 1024 × 768
Operační systém	Windows 8.1, Windows 10, Windows Server 2012R2
Pro emulátory	64 bitový systém s Hyper-V

Tabulka 2: Požadavky Visual Studia 2015 Professional

[1] Cena na českém trhu se pohybovala v říjnu 2016 kolem 17 tisíc korun.

[2] Srovnání edic je dostupné z: https://www.visualstudio.com/products/compare-visual-studio-2015-products-vs

3.3. Získání systému

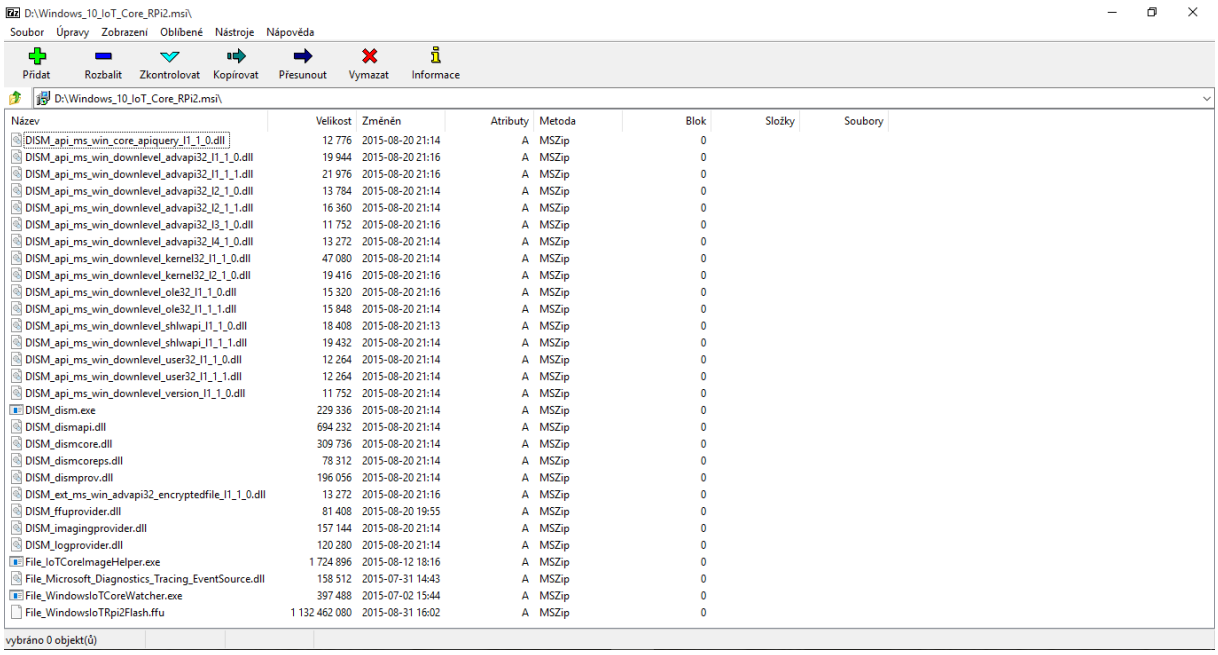
Na rozdíl od většiny systémů rodiny Windows je tento systém stahnutelný zcela zdarma a není pro jeho stažení potřeba ani registrace, nebo účet Microsoft. Stáhnout lze RTM i Insider Preview z adresy: https://ms-iot.github.io/content/en-US/Downloads.htm.

RTM verze systému je stabilnější verze systému, ale obsahuje i chyby, které jsou v Insider Preview již vyřešeny. Verze Insider Preview je vývojovou verzí, takže obrazy jsou poměrně často zaměňovány a obsahují řadu chyb. Pro počítače Raspberry Pi 2 a MinnowBoard MAX si můžeme vybrat dle požadované aplikace systému, kterou verzi si stáhneme, pro počítač Intel Joule máme k dispozici na konci října 2016 pouze verzi Insider Preview.

Stahování ISO souboru s potřebnými instalačními soubory nastane plně automaticky po kliknutí na požadovanou verzi systému. Pro vlastní stažení souborů je jedno, na jaké verzi operačního systému je to provedeno a pomocí jakého prohlížeče, ale z vlastní zkušenosti doporučuji použít Internet Explorer, nebo Microsoft Edge. Bez ohledu na zvolenou verzi systému se nám vždy stáhne ISO soubor, který má na první pohled stejný obsah.

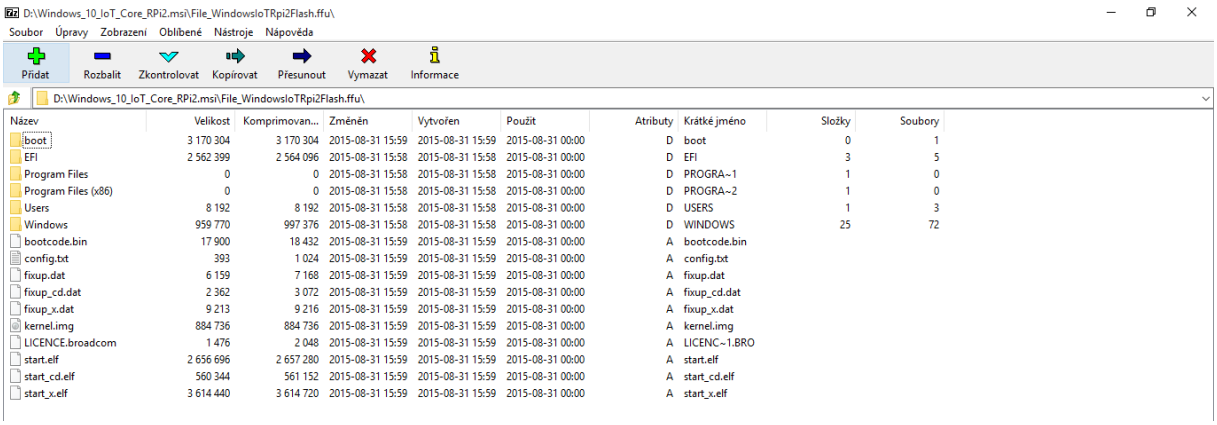
Na stránce pro stažení systému nalezneme i technické popisy, odkaz na vzorové projekty a také návody, jak vyvíjet aplikace ve starší verzi Visual Studia, tedy ve verzi 2013 pod systémy Windows 8 a novější. Pro Visual Studio 2013 je potřeba stáhnout doplňkové knihovny a nástroje, takže doporučuji využívat raději verzi 2015, která tento vývoj umožňuje přímo a je uváděna jako oficiální požadavek pro vývoj aplikací na Windows 10 IoT Core. Web pro stažení systému je kvalitním výchozím rozcestníkem k získání dalších informací potřebných pro práci se systémem a psaní aplikací, i pro ostatní systémy Windows 10.

Po stažení příslušné verze ISO souboru jej dvojklikem otevřeme v průzkumníku souborů (program Explorer.exe). Uvnitř ISO souboru najdeme instalační soubor např.: Windows_IoT_Core_Rpi2.msi. Pokud si instalační soubor rozbalíme v archivačním programu, dostaneme jeho obsah, který je následující:



Obr.1: Obsah instalátoru Windows 10 IoT Core

Z obsahu tohoto souboru nás zajímá nejvíce soubor File_WindowsIoTRpi2.ffu, který budeme extrahovat při aktualizaci na nový obraz systému a k jehož popisu se ještě vrátím. Dále jsou pro nás důležité dva spustitelné soubory, se kterými budeme pracovat po instalaci tohoto balíčku, jsou jimi File_WindowsIoTCoreWatcher.exe, který se po instalaci bude jmenovat WindowsIoTCoreWatcher.exe, a File_IoTCoreImageHelpr.exe, který se po instalaci bude jmenovat IoTCoreImageHelpr.exe. Ale zpět k souboru File_WindowsIoTRpi2.ffu, který po instalaci nalezneme jako WindowsIoTRpi2.ffu v adresáři programu. Tento soubor je obdobou souboru Install.wim, který se nachází na instalačním médiu Windows ve složce Sources. Formát ffu značí v podání Microsoftu, že se jedná o instalační soubor určený pro ARM platformu. Když si pomocí archivačního programu rozbalíme soubor File_WindowsIoTRpi2.ffu, získáme stejnou strukturu, jako u souboru Install.wim a jako se nám vytvoří na paměťové kartě při instalaci systému. Jak je možné si všimout, složky Program Fles(x86) a Program Files jsou oddělené, i když systém pracuje na architektuře ARM, protože existují i verze pro x86 a x64.



Název	Velikost	Komprimovan...	Změněn	Vytvořen	Použit	Atributy	Krátké jméno	Složky	Soubory
boot	3 170 304	3 170 304	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	D	boot	0	1
EFI	2 562 399	2 564 096	2015-08-31 15:58	2015-08-31 15:58	2015-08-31 00:00	D	EFI	3	5
Program Files	0	0	2015-08-31 15:58	2015-08-31 15:58	2015-08-31 00:00	D	PROGRA~1	1	0
Program Files (x86)	0	0	2015-08-31 15:58	2015-08-31 15:58	2015-08-31 00:00	D	PROGRA~2	1	0
Users	8 192	8 192	2015-08-31 15:58	2015-08-31 15:58	2015-08-31 00:00	D	USERS	1	3
Windows	959 770	997 376	2015-08-31 15:58	2015-08-31 15:58	2015-08-31 00:00	D	WINDOWS	25	72
bootcode.bin	17 900	18 432	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	bootcode.bin		
config.txt	393	1 024	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	config.txt		
fixup.dat	6 159	7 168	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	fixup.dat		
fixup_cd.dat	2 362	3 072	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	fixup_cd.dat		
fixup_x.dat	9 213	9 216	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	fixup_x.dat		
kernel.img	884 736	884 736	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	kernel.img		
LICENCE.broadcom	1 476	2 048	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	LICENC~1.BRO		
start.elf	2 656 696	2 657 280	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	start.elf		
start_cd.elf	560 344	561 152	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	start_cd.elf		
start_x.elf	3 614 440	3 614 720	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	start_x.elf		

Obr. 2: Obsah souboru File_WindowsIoTRpi2.ffu

Je nutné podotknout, že celý instalátor má velmi dobrou kompresi vnitřních souborů, jen soubor File_WindowsIoTRpi2.ffu má vysoký kompresní poměr a jeho plná velikost je 1132,5 MB, přitom velikost celého instalátoru, tedy i se všemi programy, které se instalují do PC, a podpůrnými knihovnami, má jen 511,43 MB. Při bližším zkoumání nalezneme i další analogie s plnou verzí systému Windows, třeba aplikace DISM.exe je obdobou F1_imagex, či F3_imagex, která má jako grafickou nadstavbu WindowsIoTImageHelper.exe. Oba totiž umožňují rozbalení zdrojového souboru instalace (v případě IoT WindowsIoTRpi2.ffu a v případě počítačové verze Install.wim) na cílové médium, které může být připojeno pomocí USB, tedy nemusí se jednat, a v případě Windows IoT se ani nejedná, o systémový disk připojený přes IDE, SATA či SCII rozhraní. K aplikaci WindowsIoTCoreWatcher.exe se vrátíme později, až s ní budeme pracovat.

K plnohodnotné práci se systémem již potřebujeme jen Visual Studio, které si můžeme zakoupit ve verzi Professional nebo Enterprise (pouze s MSDN), či si stáhnout bezplatnou verzi. V případě bezplatné verze máme na výběr ze dvou vydání, buď si stáhneme Visual Studio Community 2015, které je možné doplnit i o český jazykový balíček, nebo si stáhneme Visual Studio Express 2015 for Windows 10. Obě verze jsou bezplatné a pro aktivaci vyžadují e-mailový účet společnosti Microsoft, tedy domény windowslive.com, outlook.com, outlook.cz, hotmail.com, či hotmail.cz. Bez přihlášení se do programu pomocí účtu Microsoft je aplikace funkční po dobu 30 dní. Pokud se rozhodneme pro verzi Visual Studio Express 2015 for Windows 10, instalace probíhá pomocí online instalátoru, a můžeme vytvářet pouze aplikace Windows Universal v jazycích HTML5/Javascript, C++, C# a Visual Basic. Tyto aplikace následně, pokud jsme majiteli ověřovacího kódu, můžeme umístit do MS Store, nebo je nahrát na naše Windows 10 IoT Core. Tato verze programu má omezené možnosti ladění kódu. Její výhodou je, že nevyžaduje tolik místa na pevném disku a umožňuje stažení emulátorů pro psaní aplikací

určených na telefony. Pokud se rozhodneme pro tuto variantu, můžeme si program stáhnout z: https://www.visualstudio.com/cs-cz/downloads/download-visual-studio-vs.aspx.

Naproti tomu Visual Studio Community 2015 umožňuje vývoj mobilních řešení napříč platformami pro Windows, iOS a Android a obsahuje nástroje pro webový a cloudový vývoj. Je v ní možné programovat i běžné desktopové aplikace pro Windows a je dostupná zdarma pro samostatné vývojáře, vývoj Open Source, akademický výzkum, vzdělávací instituce a malé profesionální týmy (do 5 programátorů včetně). Tuto verzi Visual Studia je možné již doplnit o český jazykový balíček a stáhnout ji buď jako ISO otisk instalačního disku, nebo jako aplikaci online instalátoru. Tato edice již podporuje vývoj ve všech jazycích, obdobně, jako je tomu u placené verze Professional, a je možné ji doplnit o emulátory mobilních zařízení všech platforem. Tuto verzi bych doporučil všem, kteří by uvažovali o Visual Studiu jako o svém hlavním IDE. Pokud se pro tuto edici rozhodneme, je možné si ji stáhnout z: https://www.visualstudio.com/cs-cz/downloads/download-visual-studio-vs.aspx#.

Pro studenty existuje ještě jedna varianta, a to ta, že jejich škola disponuje Dreamsparkem, kde si mohou stáhnout edici Professional zcela bezplatně ve studentské licenci, která nikterak neomezuje funkce programu, ale právní cestou omezuje výdělečnou činnost.[3]

3.3.1. Užitý systém pro práci

Pro vypracování teoretické části bakalářské práce jsem si zvolil Insider Preview ze dne 14.10.2015, tedy ISO soubor jménem 10531.0.150820-1710.TH2_RELEASE_IOTCoreRPi_armFRE. Tento soubor je součástí přiloženého DVD a na přiloženém DVD je i soubor formátu ffu pro ty, kteří již instalovali jinou verzi systému a chtějí zkusit tuto konkrétní. Jedná se o verzi systému 10.0.10531. Pro vypracování praktické části a srovnání rychlosti vývoje jsem použil verzi systému 10.0.14342.1000 ze dne 8.5.2016.

Jako HW základní mám k dispozici microSDHC kartu rychlosti U1 a kapacitě 16 GB od Verbatimu, jednodeskovým počítačem je Raspberry Pi2, Wi-Fi komunikaci obstarává originální Raspberry USB Wi-Fi Doungle a zobrazovací jednotkou je 7“ displej připojený pomocí HDMI. Aplikace jsou vyvíjeny ve Visual Studiu 2015 Enterprise běžícím na Windows 10 Enterprise x64, či Windows Server 2016 Datacenter TP5, s aktivním Hyper-V.

[3] MSDNAA (Dreamspark) JCU disponuje Visual Studiem 2013 Pro a 2015 Enterprise k 28.10.2016

3.4. Specifika Windows 10 IoT Core

Proti svým Linuxovým konkurentům má tento systém dvě velká specifika, kterých si všimne každý uživatel. Prvním specifikem je, že systém umožňuje mít spuštěnou jen jednu aplikaci typu Windows Universal (desktopové aplikace nejsou podporovány). Tato vlastnost má v první řadě zajistit stálý výkon HW pro řízení výstupních jednotek. Další na první pohled viditelným rozdílem je to, že vlastní ovládání systému neprobíhá grafickým rozhraním, ale vzdáleně z jiného počítače pomocí webového serveru umístěného v systému. Z výchozí aplikace systému (jak u RTM, tak u Insider Preview) je potřeba nastavit jazyk systému a je možné konfigurovat připojení k Wi-Fi, dále lze otevírat manuály a zobrazit si adresy nápovědy. Výchozí aplikace především vypisuje důležité informace o systému, jako je jméno počítače, IP adresa verze 4 a 6, MAC adresa a verze systému.

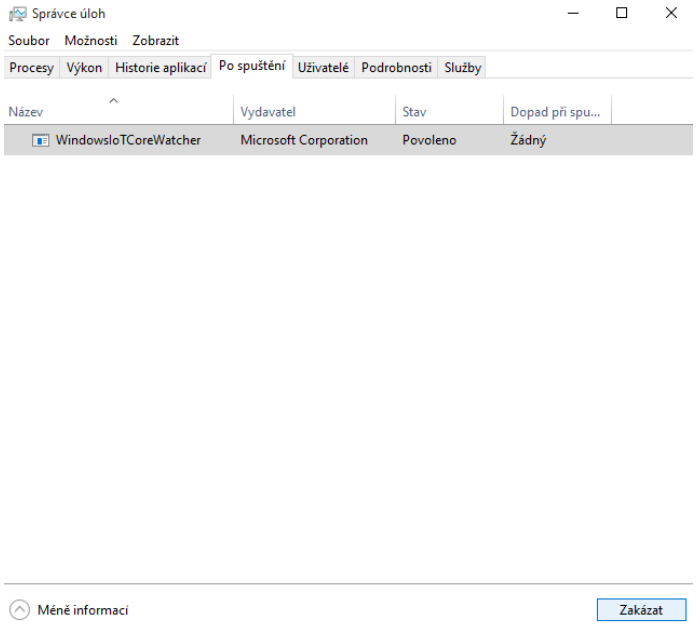
Co je u Windows 10 IoT Core pravidlem a u Linuxové konkurence možností, je bootování do konkrétní aplikace. V případě Windows lze aplikaci přepnout pomocí webserveru nebo zapnutím ladícího programu Visual Studia. V případě přerušení aplikace ladícím programem Visual Studia se po jeho ukončení systém vrací systém k původní aplikaci. V případě kolapsu systémového procesu nebo chybě aplikace (nemusí to být nutně úplný pád, stačí stav, který je na desktopových aplikacích označován jako „neodpovídá“) dojde k automatickému restartování systému (není pravidlem pro kolaps aplikace nasazené ladícím programem Visual Studia), zpravidla po 50 vteřinách (pokud nenastavíme jinak a systém problém nevyřeší) a k opětovnému spuštění výchozí aplikace, i v případě, že zkolabovala aplikace ručně spuštěná pomocí webového rozhraní. Pokud jde o konfiguraci pomocí SSH u Linuxových konkurentů, má Windows obdobu v konfiguraci pomocí PowerShellu 5, jde to i pomocí starší verze, ale tato verze umožňuje tvorbu a sdílení modulů pro práci se systémem. Pokud chci, aby systém prováděl výhradně operace zadávané pomocí PowerShellu, existuje mezi vzorovými projekty od Microsoftu též projekt umožňující toto nastavení systému, ale pro systémovou konfiguraci, jak si ukážeme dále, není potřeba nic víc, nežli znát IP adresu nebo jméno počítače, a můžeme psát příkazy v PowerShellu. K systému je možné se připojit i pomocí SSH, ale je nutné si uvědomit, že se připojujeme k PowerShellu, nikoliv k Bashi.

Tím nejvýraznějším rozdílem proti Linuxové konkurenci je chráněné jádro systému založené na technologii NT, které je pro tento systém obdobné jako v případě mobilní verze. Systém jako takový podporuje Plug and Play pro většinu externího hardwaru připojeného pomocí USB. Rozšiřující knihovny umožňují práci s GPIO sběrnicí.

Na rozdíl od zbytku rodiny systémů Windows 10 není tento systém vybaven službou Windows update[4] (vyjímaje Windows 10 IoT Enterprise a Windows 10 IoT Mobile Enterprise) a přístupem do Microsoft Store. Z tohoto faktu vyplývá, že veškeré aktualizace na nový obraz systému, který bude opravovat chyby stávajícího obrazu, je nutné provést formou reinstalace systému, jak je popsáno níže.

3.5. Instalace systému Windows 10 IoT Core

Instalace systému na paměťovou kartu je začátkem práce se systémem. Pokud nemáme nainstalovaný IoTCoreImageHelpr.exe, dvojklikem si otevřeme stažený ISO soubor a spustíme instalační balíček, který je v ISO souboru umístěn. Instalaci balíčku dokončíme dle pokynů průvodce na obrazovce a všechny hodnoty necháváme výchozí. Po dokončení instalace přejdeme do správce úloh (ctl+shift+esc, nebo ctrl+alt+dele a správce úloh), kde zvolíme záložku „Po spuštění“ a nalezneme tam WindowsIoTCoreWatcher od vydavatele Microsoft, klineme na něj (jednou levým tlačítkem) a následně dole na „zakázat“, pro trošku zběhlejší je možné kliknout pravým tlačítkem a zvolit zakázat.



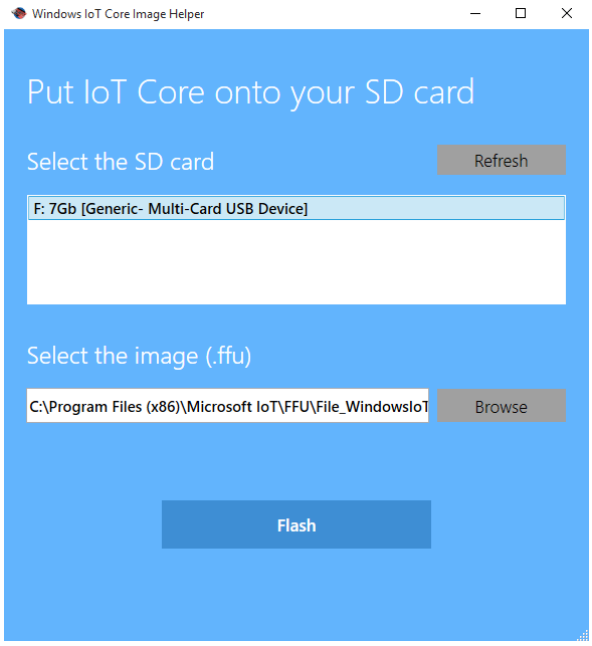
Obr. 3: Zakázání automatického spuštění WindowsIoTCoreWatcher se startem počítače

Doporučuji si připnout WindowsIoTCoreWatcher ke startu nebo na lištu, či si vytvořit zástupce na ploše, protože jde o utilitu, která je hodně používaná. WindowsIoTCoreWatcher a IoTCoreImageHelpr nalezneme ve startu ve všech programech ve složce Microsoft IoT, nebo si jej můžeme vyhledat pomocí vyhledávání.

[4] Platí po verze systému starší, nežli 10.0.14342.1000

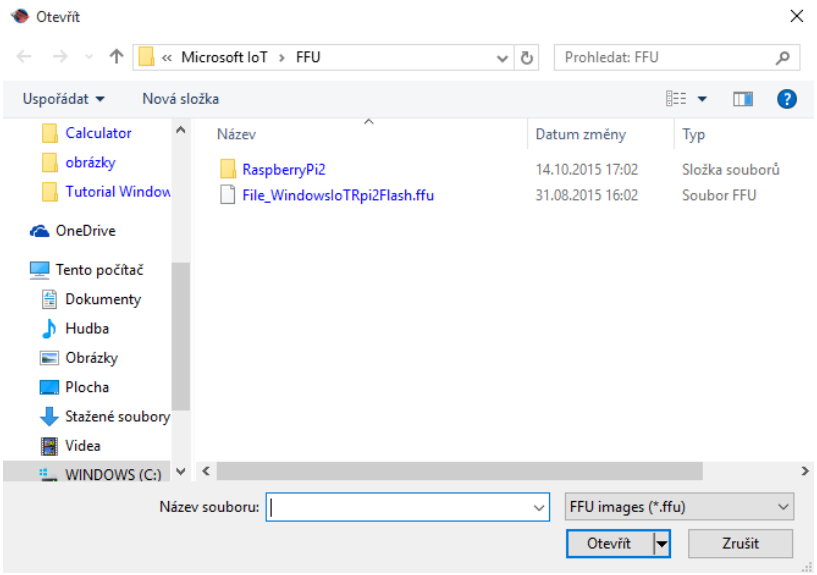
Nyní stačí jen připojit připravenou microSD kartu do čtečky karet v počítači, nebo pomocí USB portu. Pokud máme na kartě, kterou chceme použít, nějaké soubory, překopírujeme si je do počítače, abychom o ně nepřišli. Kartu doporučuji naformátovat na formát NTFS, ale není to nutné. V tuto chvíli máme již všechno připravené k tomu, abychom mohli provést vlastní instalaci.

Vlastní instalaci započneme spuštěním programu IoTCoreImageHelpr, který nám z paměťové karty udělá spustitelné médium s nainstalovaným operačním systémem. Program si sám nalezne paměťovou kartu, která je připojena k počítači, ale pozor, pokud jsou připojeny dvě karty, nebo USB flash disk, musíme správný disk vybrat z nabídky. Pomocí průzkumníka souborů, nebo Total Comanderu, zjistíme písmeno, které systém přiřadil dané kartě, abychom se nedopustili chyby. Toto písmeno lze zjistit i ze správce disků, který otevřeme kliknutím pravým tlačítkem na ikonu Start a zvolením položky Správa disků. Ještě je dobré dát si pozor, aby daný disk nebyl otevřený v nějakém jiném programu (včetně průzkumníku). Na následujícím obrázku je zobrazeno nakonfigurované okno programu IoTCoreImageHelper připravené k provedení vlastní instalace.



Obr. 4: Nastavený IoTCoreImageHelper pro instalaci systému na vlastní disk

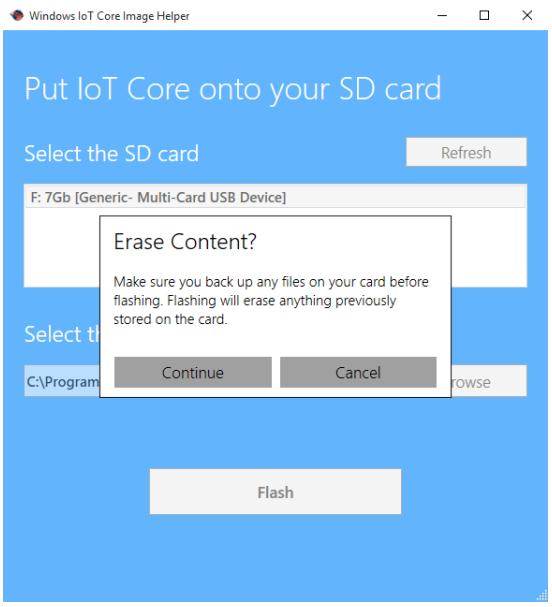
Ze zobrazeného seznamu paměťových zařízení jedním kliknutím vybereme požadovaný disk. Kliknutím na tlačítko Browse se dostaneme do průzkumníku, kde jsme ve složce programu, a zvolíme si požadovaný soubor formátu ffu.



Obr. 5: Vybrání souboru obrazu systému, který bude instalován

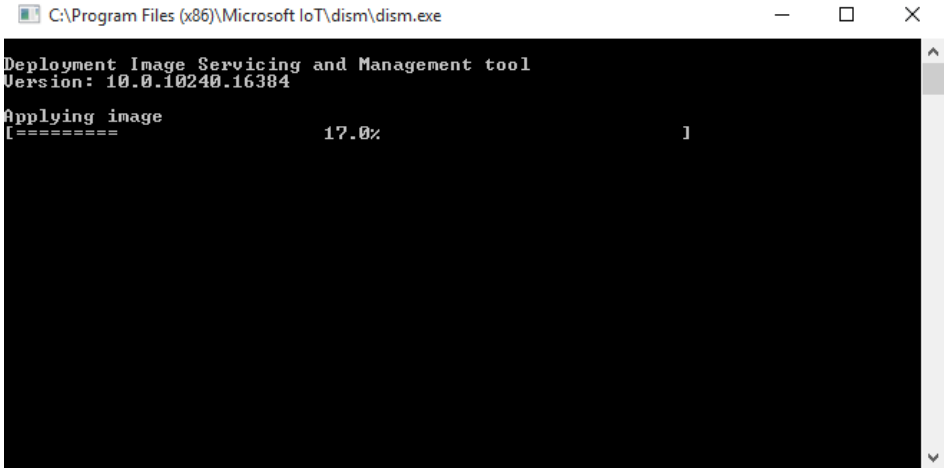
Běžně po instalaci nalezneme obraz v zobrazené složce, pokud provádíme upgrade a extrahovali jsme si soubor formátu ffu, zadáme cestu k tomuto souboru. Průzkumník se chová obdobně jako v případě Wordu, či jiného programu. Po vybrání souboru zvolíme položku otevřít a dostaneme se do stavu, který je na obrázku 4.

Ještě jednou zkontrolujeme, zda máme všechno správně nastaveno, tedy zda máme vybraný správný disk pro instalaci a požadovaný obraz systému. Pokud je všechno správně, tak již stačí jen stisknout tlačítko Flash a automatika již provede zbytek práce za nás. Jen nesmíme zapomenout potvrdit pomocí Continue, že opravdu chceme kartu smazat a nainstalovat na ni operační systém.



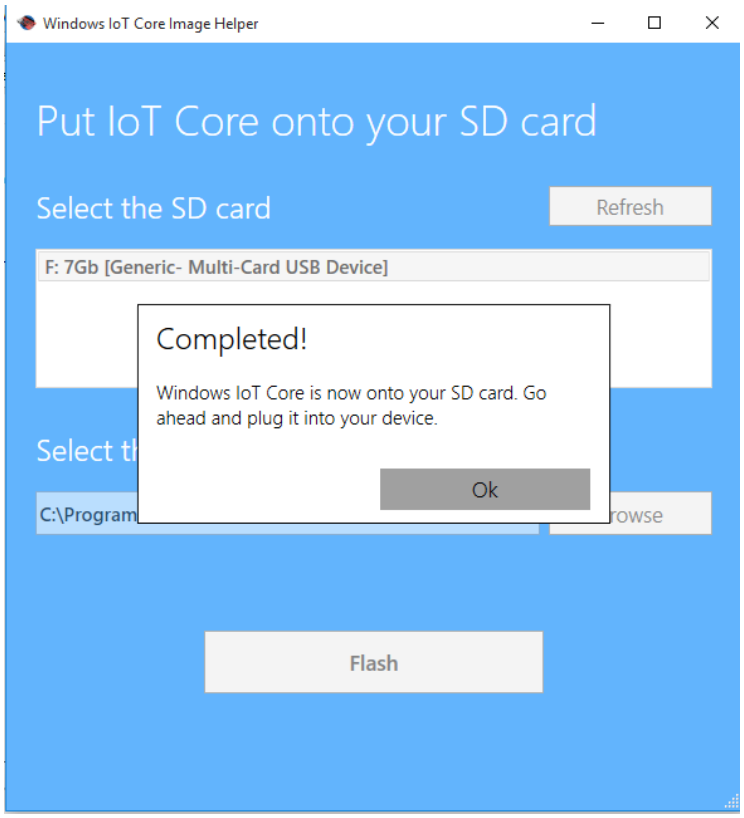
Obr. 6: Spuštění instalace systému

Pokud spustíme vlastní instalaci, dojde k zavolání programu dism.exe, který se nám nainstaloval do počítače při aplikaci instalačního balíčku z ISO souboru. Tento program běží v příkazové řádce a provede vlastní vytvoření systémové jednotky ze zvolené karty.



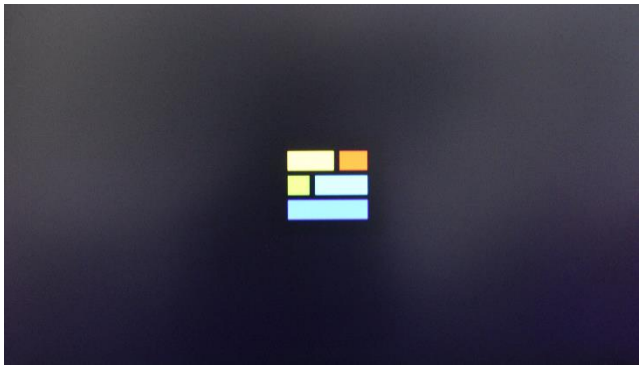
Obr. 7: Program dism

Vlastní rozbalení systému na kartu trvá cca 2 až 3 minuty a po nás jako uživatelích není nic vyžadováno. Pokud jsme všechno provedli správně, zobrazí se nám obrázek 8 a můžeme program IoTCoreImageHelpr uzavřít. Nyní jen nesmíme zapomenout kartu bezpečně odebrat, tedy vysunout, a následně ji můžeme vyndat ze čtečky a vložit do slotu na vlastním jednodeskovém počítači.



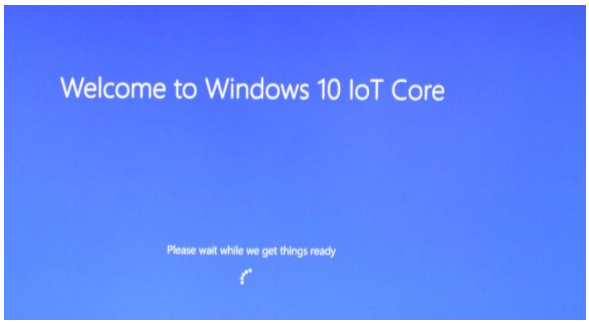
Obr. 8: Informace o dokončení instalace systému Windows 10 IoT Core na microSD kartu

Po vložení paměťové karty připojíme k jednodeskovému počítači všechny potřebné periferie, i když pro začátek doporučuji omezit se na Ethernet či Wi-Fi, myš, klávesnici a eventuálně Bluetooth, když bychom si chtěli spárovat periferie. Co budeme potřebovat zcela jistě, je připojit zobrazovací jednotku, monitor, televizi, nebo displej pomocí HDMI. Pokud máme všechno připraveno, můžeme připojit napájení a spustit počítač. Při spouštění uvidíme na obrazovce logo Windows stejně jako v případě, když spouštíme Windows 8 a novější verze na klasickém PC. Následuje načítání výchozí aplikace (viz obrázek 9). Když je na obrazovce toto logo, je systém již načtený, ale spouští zvolenou výchozí aplikaci.



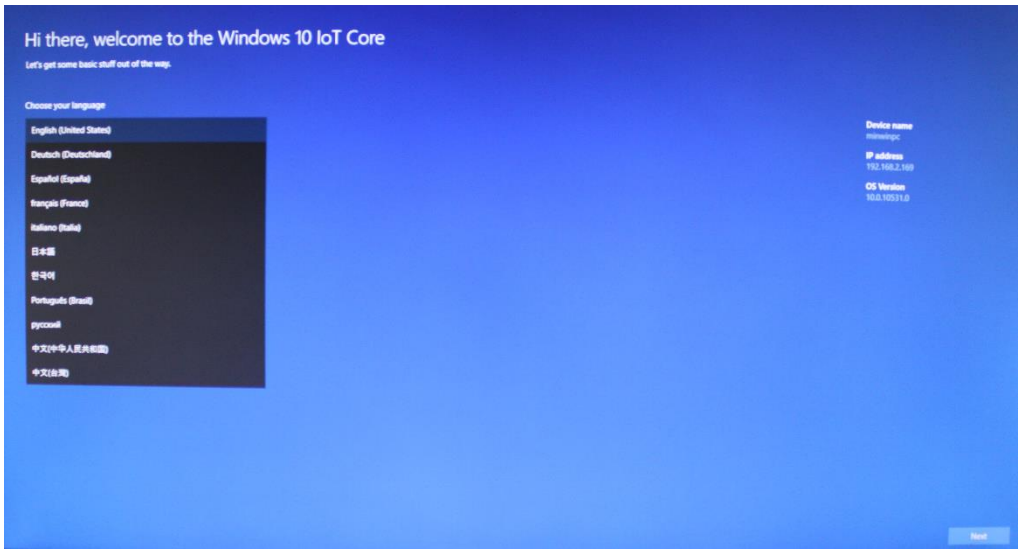
Obr. 9: Načítání aplikace na Windows 10 IoT Core

Jako první uvidíme pomocnou konfigurační aplikaci, obdobu zdlouhavého dokončování instalace Windows do klasického počítače, ze kterého se systém dostane na plochu. Tato pomocná aplikace je ukázána na obrázku 10.

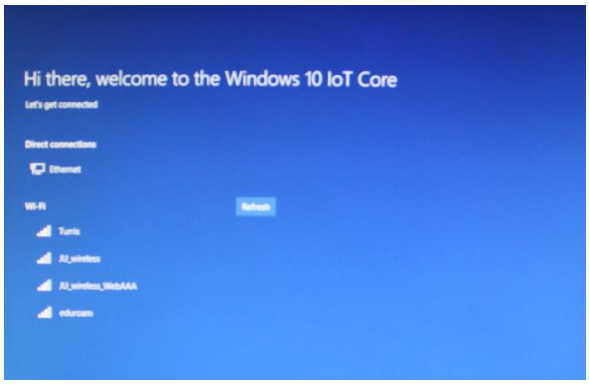


Obr. 10: Pomocná aplikace konfigurace OS

Aplikace nám poslouží pro nastavení jazyka zobrazení systému (obr. 11) a nastavení Wi-Fi připojení (obr. 12). Po dokončení této konfigurace systém přejde do standartní výchozí aplikace, kterou uvidíme již při každém dalším spuštění, pokud nenastavíme jinak. Tím máme první spuštění za sebou.

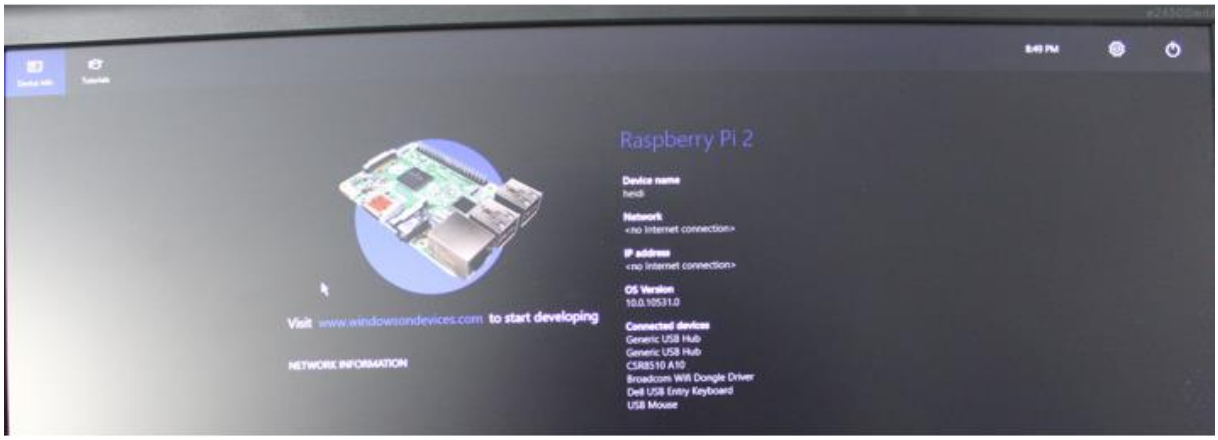


Obr. 11: Nastavení jazyka zobrazení v systému Windows IoT

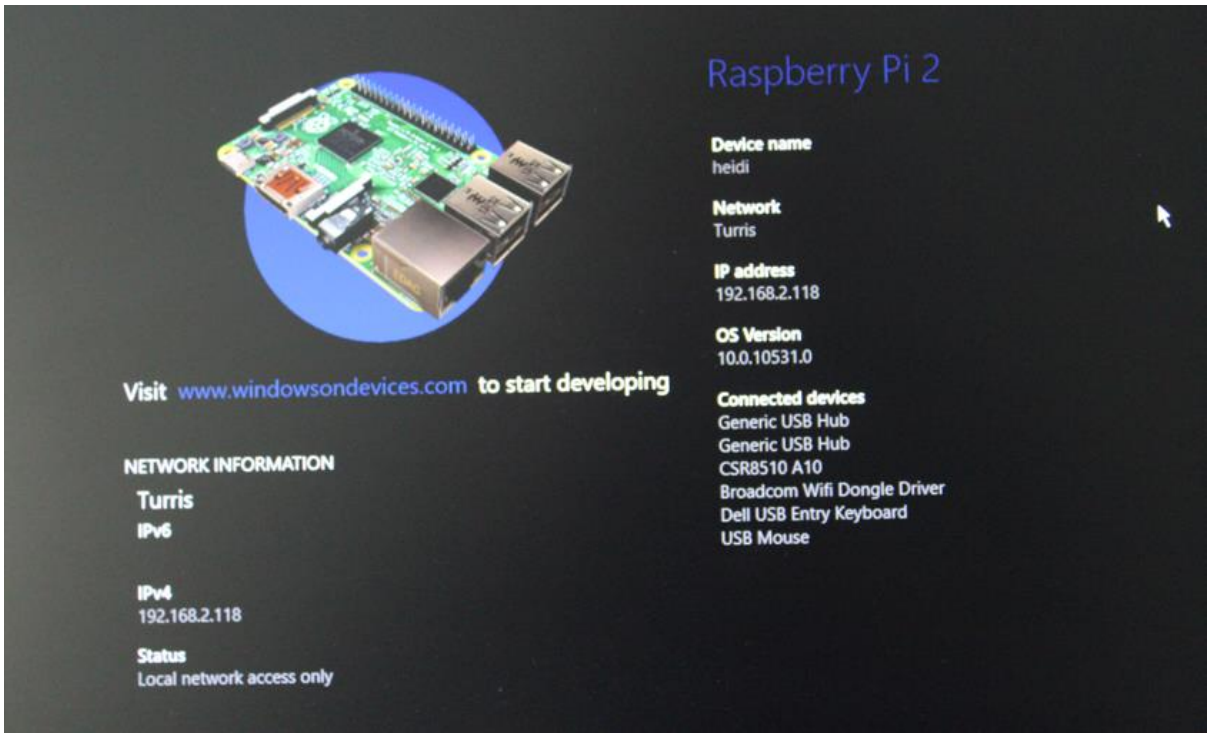


Obr. 12: Nastavení Wi-Fi připojení

Upgrade na nový obraz systému se provede velmi jednoduše. Po stažení nového ISO souboru si pomocí archivačního programu rozbalíme z instalačního balíčku soubor formátu ffu a umístíme jej někam na disk, doporučuji do adresáře „C:\Program Files (x86)\Microsoft IoT\FFU“, kde je v podsložce náš původní instalační obraz. Jen podotýkám, že pro zápis do tohoto adresáře mohou být požadována práva administrátora. Následně si zálohujeme soubory z paměťové karty, spustíme si IoTCoreImageHelpr, a provedeme všechny kroky jako v případě instalace. Následně dokončíme konfiguraci systému a nahrajeme aplikace. Po dokončení instalace se nám zobrazí výchozí aplikace systému (viz obrázek 13 a 14).



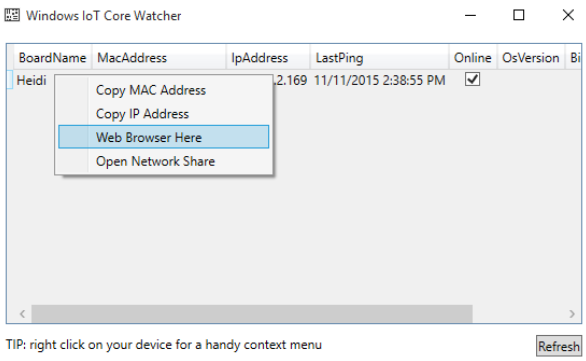
Obr. 13: Výchozí aplikace systému



Obr. 14: Výchozí aplikace, pokud je systém připojen k Wi-Fi i Ethernetu

3.5.1. Základní konfigurace Windows 10 IoT Core

Pro vyhledání zařízení v síti máme dvě možnosti, buď opsat ze zobrazovače zobrazenou IP adresu, nebo si spustit IoTCoreWatcher. Po nalezení zařízení pomocí IoTCoreWatcheru stačí kliknout pravým tlačítkem na název počítače a zvolit kopírovat IP adresu (viz obrázek 15).

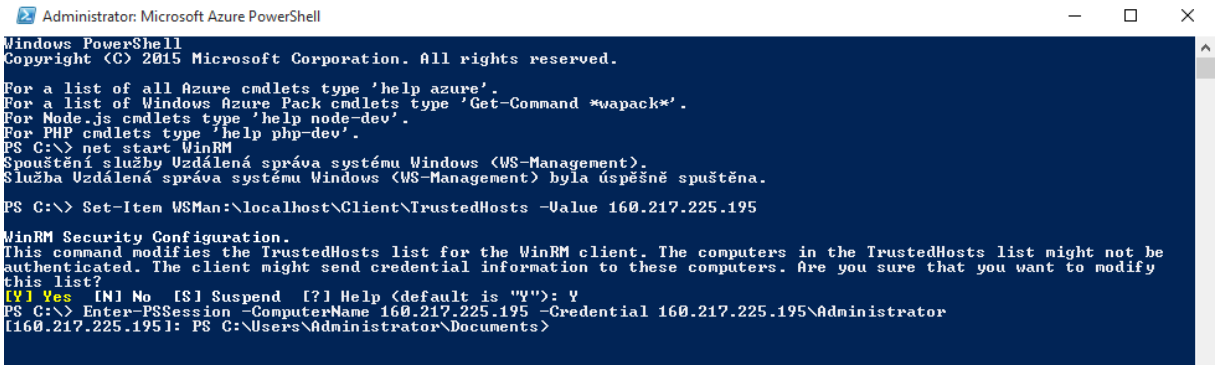


Obr. 15: WindowsIoTCoreWatcher

Jen pozor na častý problém, vždy je dobré podívat se, jestli bliká síťová karta, nebo se přesvědčit, zda jsme opravdu připojeni k Wi-Fi. To jsou nejčastější důvody toho, proč nelze počítač najít v síti a připojit se k němu. Windows IoTCoreWatcher nám zobrazí všechna zařízení s Windows IoT, která se nachází v síti.

Po instalaci má systém nastavené výchozí hodnoty pro přihlášení. Uživatelský účet je *Administrator* a heslo k němu je: p@ssw0rd

Pro změnu výchozího hesla je možné použít PowerShell, který je potřeba mít spuštěný jako správce, pomocí vzdáleného připojení k Windows 10 IoT Core. Na následujícím obrázku je vidět postup pro připojení se k Windows 10 IoT Core pomocí PowerShellu. Doporučuji užít základní systémový PowerShell, kdo užije (jak je na obrázku ukázáno) Azure PowerShell, nic se neděje, jen nemá tak dobře barevně vyznačenou syntaxi.

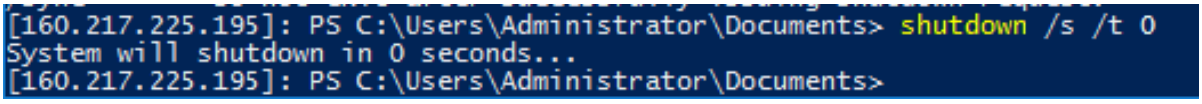


Obr. 16: Přihlášení pomocí PowerShellu k Windows 10 IoT Core

Pokud jsme přihlášeni, užijeme příkaz: „net user Administrator [new password]“ pro provedení změny hesla účtu administrátora. Na obrázku 19 nalezneme příklad použití tohoto příkazu, jak je aplikován při vypracování mojí bakalářské práce.

Obr. 17: Změna hesla administrátora z p@ssw0rd na LM27C256

Pro změnu názvu zařízení z výchozího minwinpc na náš libovolný (v mém případě na Heidi) použijeme příkaz: „setcomputername heidi“. Po provedení tohoto příkazu je nutné zařízení restartovat či vypnout. Vypnutí zařízení je ukázáno na obrázku 18, pokud budeme chtít počítač restartovat, příkaz budeme modifikovat tak, že místo „/s“ napíšeme „/r“. Parametr „/t 0“ značí, že restart proběhne hned, pokud si přejeme jiný čas, tak místo 0 dosadíme jiný čas v sekundách, kdy největší možné oddálení je 10 let, tedy 315360000 sekund. Nápovědu k příkazu vyvoláme pomocí „shutdown /?“. Velkým problémem je, že se špatně poznává okamžik, kdy je systém skutečně vypnutý, protože kontrolky na zařízení svítí hodně dlouho a PowerShell neumožní spustit po tomto příkazu v rámci relace nic dalšího. Z vlastní zkušenosti mohu říct, je že, když systém přejde do stavu off-line dle IoTCoreWatcheru, je prakticky vypnutý. V další podkapitole si řekneme, jak toto poznat při práci ve webovém rozhraní.

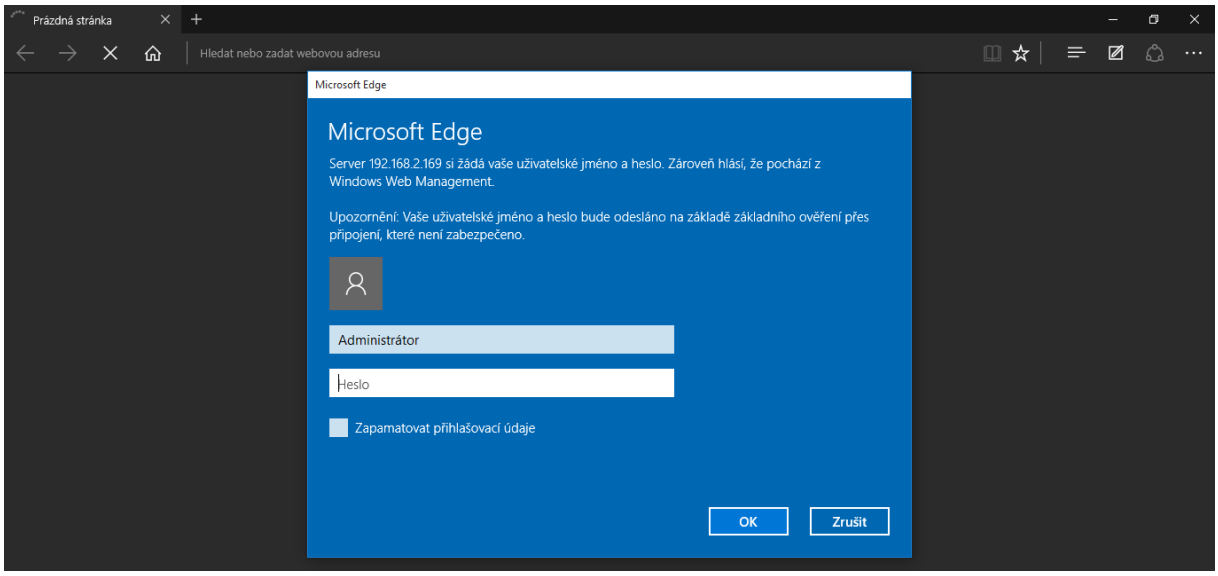


Obr. 18: Vypnutí počítače Raspberry pomocí PowerShellu

V PowerShellu opustíme relaci pomocí příkazu „exit“ a pro připojení se k zařízení po restartu musíme začít od příkazu „Set-Item WSMan:\localhost\Client\TrustedHosts -Value Heidi“, (uvedený tvar je konkrétní příklad) pokud jsme místo IP adresy použili název počítače, pokud jsme použili IP adresu, je možné rovnou vyvolat vzdálené připojení. V případě změny IP adresy platí nutnost užití výše zmiňovaného příkaz s novou IP adresou.

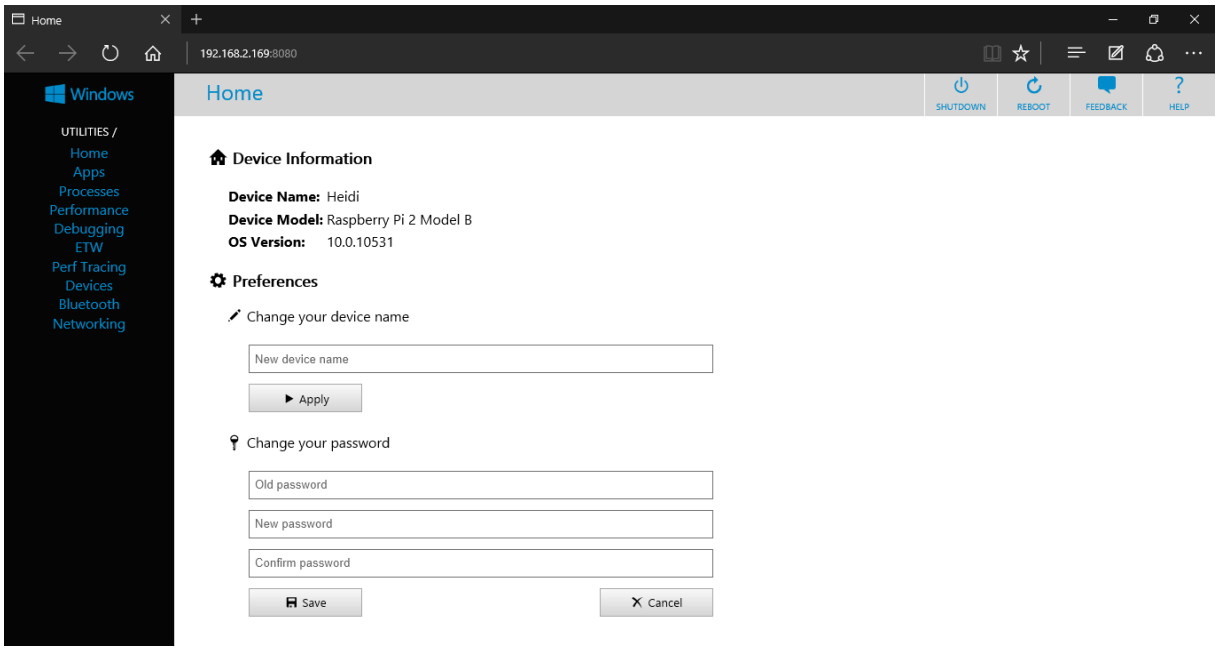
3.5.2. Webové rozhraní pro řízení systému

Mnohem pohodlnější možností, jak pracovat s Windows 10 IoT Core je použít integrovaného webového rozhraní pro ovládání systému. Jsou teoreticky dvě možnosti, tou první je opsání IP adresy (myslím verzi 4) do adresního řádku webového prohlížeče, druhou je užít oblíbený IoTCoreWatcher a kliknout na jméno počítače pravým tlačítkem a zvolit „Web Browse Here“ jak je zvýrazněno na obrázku 15. Mně se prakticky osvědčila jen druhá zmiňovaná cesta, u té první se mi často stává, že skončím hlášením „Stránku nelze zobrazit“. Nicméně obě cesty vedou na přihlášení se k systému pomocí webserveru, jak je vidět na obrázku 19. Do formuláře zadáváme jak uživatelské jméno, tak heslo, protože si můžeme vytvořit i další uživatele.



Obr. 19: Přihlášení k Windows 10 IoT Core pomocí Microsoft Edge

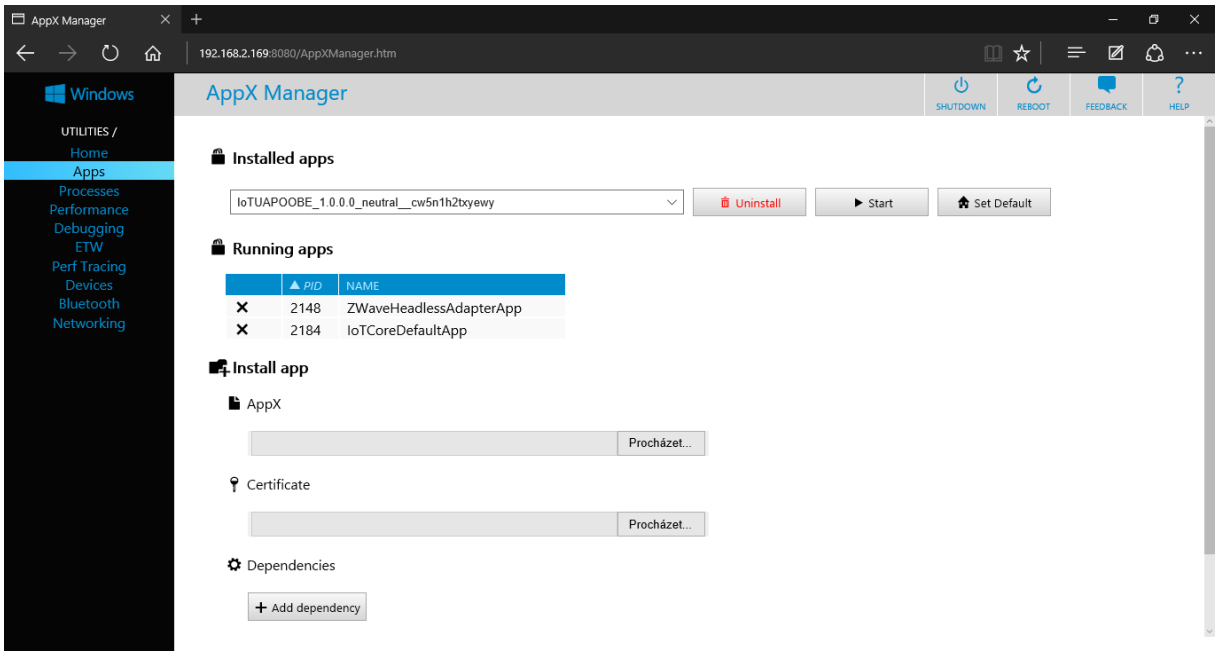
Po přihlášení se nám zobrazí stránka Utilities, která nám umožňuje provést všechna nastavení, která jsme dělali v minulé podkapitole pomocí PowerShellu, tedy umožňuje změnit jméno zařízení a změnit heslo přihlášeného uživatele. Ve skutečnosti však nejde o stránku Utilities, ale o stránku Home, protože Utilities není stránkou, ale adresářem. V horním neměnném pásu je možnost rozbalit si nápovědu, poslat zpětnou vazbu do Microsoftu a restartovat, či vypnout zařízení. Když jsem se opět dostal k problematice vypínání, je dobré vypínat, když jsme připojeni pomocí webového rozhraní na stránce Performance, protože se zmizením křivek z grafů můžeme zařízení považovat za vypnuté.



Obr. 20: Úvodní stránka webového rozhraní

3.5.2.1. Aplikace

Stránka Apps nám umožní práci s univerzálními aplikacemi pro Windows. Na stránce je možné nahrát balíček aplikace, který si vytvoříme pomocí Visual Studia, zvolit výchozí aplikaci, která se spustí se startem systému a také zvolit aplikaci, která poběží v daný okamžik, tedy spustit konkrétní aplikaci ze seznamu nainstalovaných aplikací. Na této stránce je možné aplikace i odinstalovat či zastavit.

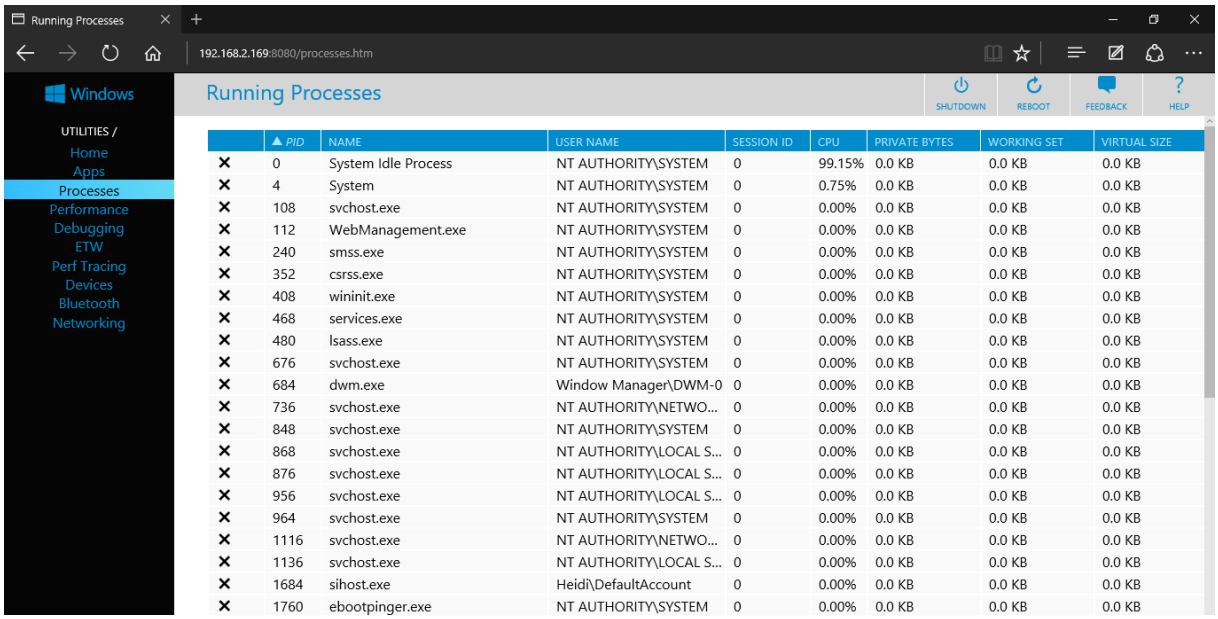


Obr. 21: Stránka Apps

Pokud máme vytvořený balíček aplikace a certifikát k aplikaci, přihlásíme se do webového rozhraní systému a přejdeme na záložku „Apps“ a ve spodních 2/3 stránky je sekce, která slouží k instalaci nových aplikací do systému. Do první kolonky vložíme cestu k balíčku s příponou appx stejným způsobem, jako když bychom chtěli nahrát soubor na kteroukoli jinou webovou stránku. Do druhého slotu vložíme obdobně cestu k certifikátu a všechno stvrdíme úplně dole tlačítkem „Go“. Nyní stačí počkat, až se balíčky přenesou do zařízení, což trvá o něco déle nežli tomu při vzdáleném ladění pomocí Visual Studia. Po dokončení nahrávání (obnoví se stránka) si aplikaci vyhledáme v horní části stránky, konkrétně v sektoru Installed Apps, v seznamu, který si zobrazíme rozkliknutím. Následně doporučuji nejdříve systém přepnout do naší aplikace pomocí tlačítka „start“ a po ověření, že aplikace pracuje jak má, ji teprve nastavit jako výchozí tlačítkem „Set Default“.

3.5.2.2. Procesy

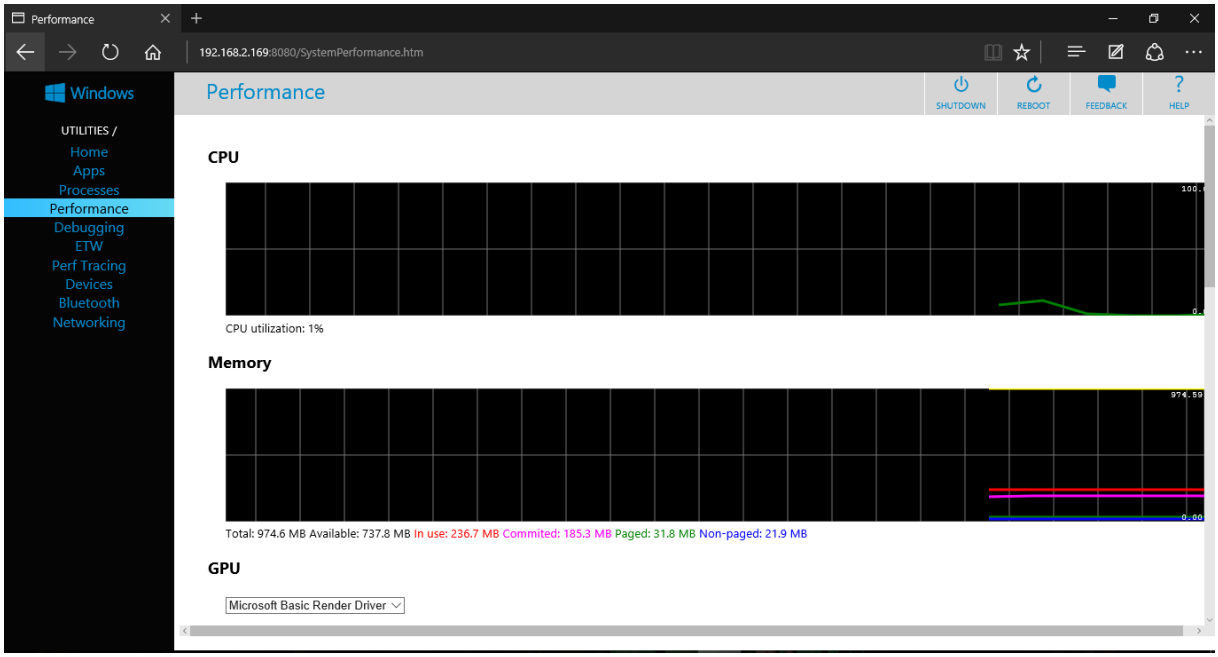
Velmi důležitou je stránka Processes, která zobrazuje podrobný výpis běžících procesů, umožňuje jejich ukončování a ve spodní části má i okno umožňující spouštění procesů či příkazů. Proces či příkaz se spouští s právy přihlášeného uživatele.



Obr. 22: Stránka procesů

3.5.2.3. Sledování výkonu

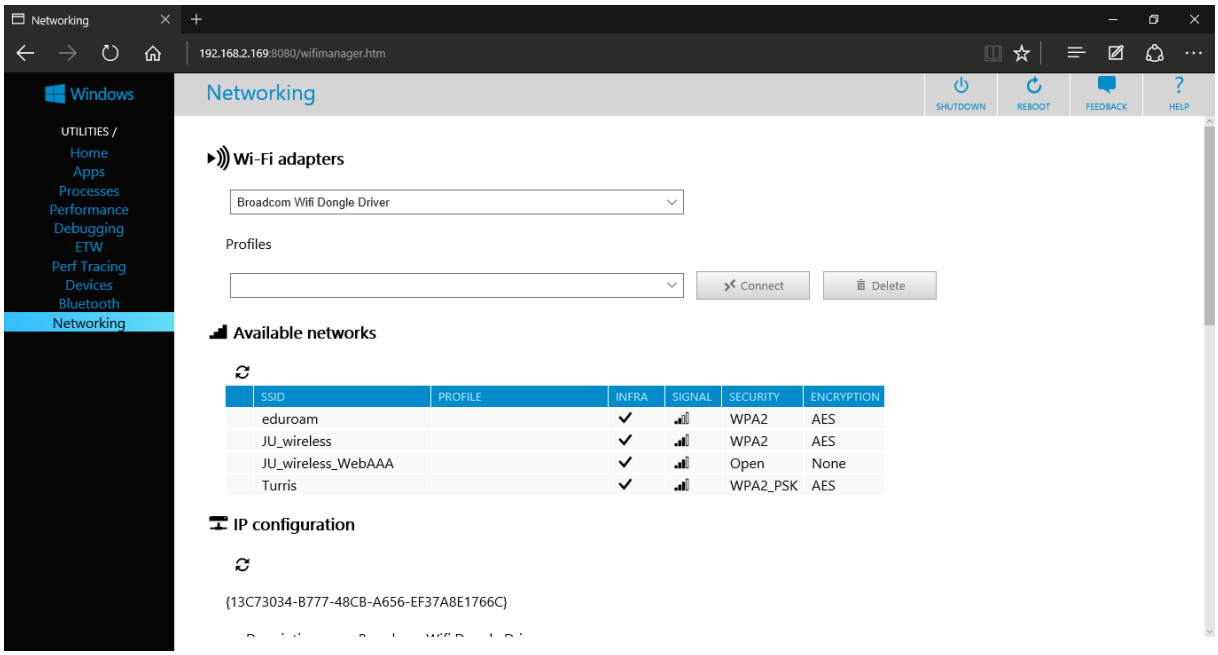
Ke sledování výkonu použijeme stránku Performance, kde je možné graficky i číselně zjistit aktuální využití systémových prostředků, externích síťových a grafických karet.



Obr. 23: Stránka Performance

3.5.2.4. Správa sítě

Poslední stránkou, která je z hlediska běžného užívání systému velmi důležitou je stránka Networking, kde je možné nastavit Wi-Fi připojení, zobrazit aktuální konfiguraci síťového připojení a v případě více zařízení pro síťový přístup mezi těmito zařízeními přepínat. Dále je možné na této stránce spravovat profily sítí. Pro informace o ovladačích a o tom, kde je, co připojeno, použijeme stránku Devices.



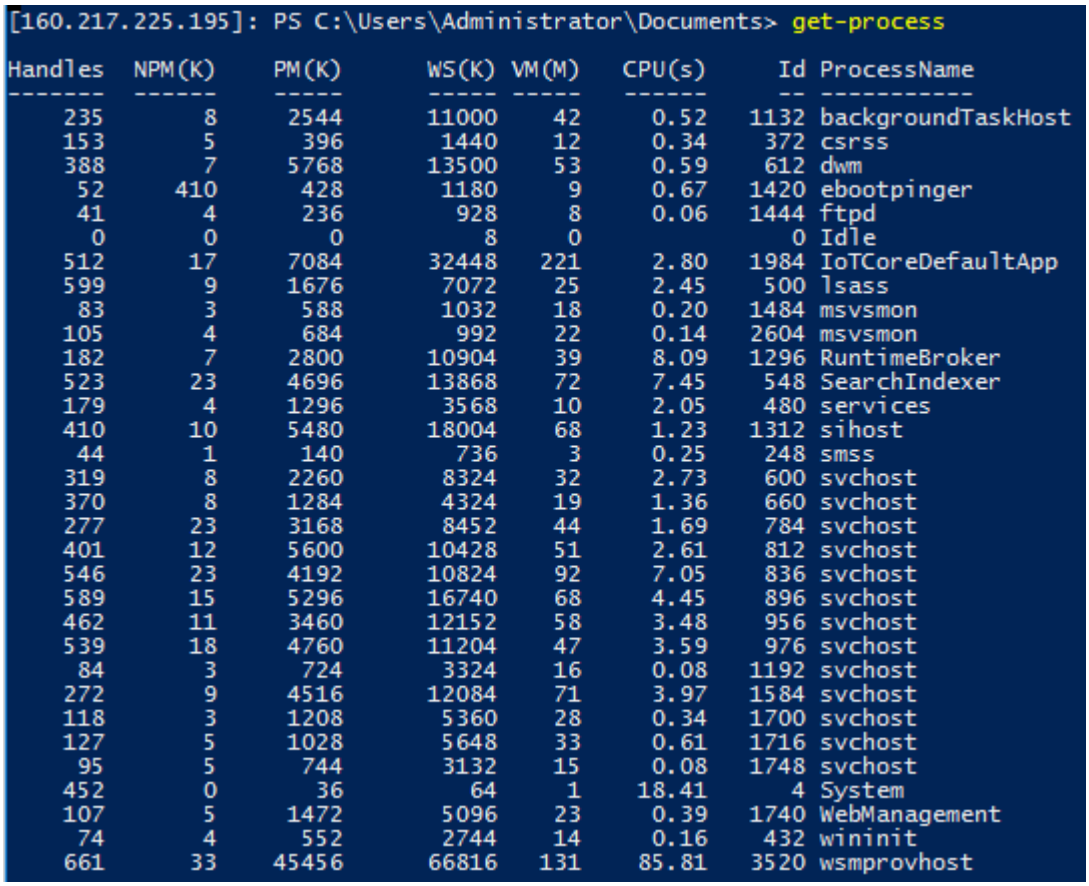
Obr. 24: Stránka pro konfiguraci síťového připojení

3.6. Práce se systémem pomocí PowerShellu

PowerShell je velmi mocný nástroj pro práci se systémem Windows a v případě Windows IoT to platí obzvláště. Pokud chceme v tomto systému třeba vytvořit nového uživatele, nemáme v podstatě jinou možnost nežli použít PowerShell. I při zpracování této práce jsem PowerShell hodně používal, takže nyní chci uvést pár užitečných příkazů, které se mohou hodit při práci se systémem. Všechny příkazy, které tu uvádím, předpokládají, že je uživatel již připojený pomocí vzdálené relace k systému.

3.6.1. Procesy

Prvním užitečným příkazem je „tlist“, který vypíše seznam běžících procesů a jejich ID. Jeho lepší dvojče je příkaz „get-process“, který vypíše podrobné informace o běžících procesech, obdobně jako je tomu u webového rozhraní. Jak vypadá výstup z tohoto příkazu, je vidět na obrázku 25. Pro ukončení procesu užijeme příkaz „kill.exe [jméno či ID procesu]“.



Obr. 25: Výstup příkazu Get-Process

Následující tabulka vysvětlí význam zkratek, které se objevují ve výpisu.

Název zdroje	Zkratka	Popis
Handles	žádný	Popisovač umístění v paměti. Při zavření popisovače je paměť uvolněna.
Nestránkovaný fond (v kilobajtech)	NPM(k)	Nestránkovaný fond je ukládání paměti, která není nikdy stránkována na pevný disk, takže je to rychleji dostupné.
Stránkovaný fond (v kilobajtech)	PM(k)	Stránkovaný fond může zaslat disk v případě, že místo je vyžadováno. Díky stránkovanému fondu je místo účinně větší než u nestránkovaného fondu. Některé čtení paměti může trvat déle, protože požadovaná data jsou nyní uložena na disku.
Pracovní sada (v kilobajtech)	WS(k)	Pracovní sada je sada stránek fyzické paměti, kterou proces používá. Pouze data uložená ve fyzické paměti (momentálně stránkování na disk) je v pracovní sadě.
Virtuální paměť (v megabajtech)	VM(M)	Velikost virtuální paměti přidělené pro výhradní použití v procesu.
Čas procesoru (v sekundách)	CPU (s)	Množství času procesoru, které proces používá (na všech dostupných procesorech).
ID procesu	žádný	Jedinečný identifikátor pro daný proces. I na sdílený počítač bude mít pouze jeden proces daným identifikátorem.
Název procesu	žádný	Popisný identifikátor procesu, ale na rozdíl od ID procesu nikoli nutně jedinečný.

Tab. 3: Vysvětlení zkratek ve výpisu příkazu Get-Process

3.6.2. Připojení k síti

Pokud chceme vidět, jak vypadá aktuální stav připojení systému k síti, zda je systém připojený pomocí Wi-Fi, nebo Ethernetu, či jaký rychlostní standard podporuje síťová karta pro Ethernet, je na místě použít příkaz „Get-NetIPConfiguration“, jak je vidět na obrázku.

```
[160.217.225.195]: PS C:\Users\Administrator\Documents> get-NetIPConfiguration


InterfaceAlias       : Ethernet
InterfaceIndex       : 5
InterfaceDescription : LAN9512/LAN9514 USB 2.0 to Ethernet 10/100 Adapter
IPv4Address          : 160.217.225.195
IPv6DefaultGateway   :
IPv4DefaultGateway   : 160.217.225.254

InterfaceAlias       : Wi-Fi
InterfaceIndex       : 4
InterfaceDescription : Broadcom Wifi Dongle Driver
NetAdapter.Status    : Disconnected
```

Obr. 26: Výpis IP konfigurace systému

3.6.3. Výpis informací o procesoru

Pokud chceme spustit vzdálené ladění z Visual Studia pro nějakou aplikaci a nevíme, zda systém běží na ARM, nebo na x86 či x64, není lepší možnost, nežli se připojit pomocí PowerShellu ke vzdálenému počítači a dát příkaz „get-ciminstance win32_processor“, který nám vrátí architekturu procesoru, jak je vidět na obrázku 27. Tento příkaz funguje v jakékoliv verzi systému Windows, včetně mobilních systémů, Windows CE a Windows Embedded. Jen pozor, vypisuje opravdu architekturu procesoru, nikoliv OS.

```
[160.217.225.195]: PS C:\Users\Administrator\Documents> get-ciminstance win32_processor
DeviceID Name                            Caption                         MaxClockSpeed SocketDesignation Manufa
                                                                                                        cturer
-------- ----                            -------                         ------------- ----------------- ------
CPU0     ARM Family 7 Model C07 Revision  5 ARM Family 7 Model C07 Revision  5 600          Socket            A
```

Obr. 27: Zobrazení architektury procesoru pomocí PowerShell

3.6.4. Správa uživatelských účtů

Kromě výchozího účtu administrátora si můžeme vytvořit další uživatelský účet, a to jako na běžném počítači s právy administrátora, nebo s právy běžného uživatele. Na rozdíl od plných Windows 10 jsou všechny uživatelské účty jen lokální. Pro vytvoření účtu běžného uživatele použijeme příkaz „net user [uživatelské jméno] [heslo] /add“ a účet administrátora vytvoříme tak, že vytvoříme běžného uživatele a následně aplikujeme příkaz „net localgroup Administrators [uživatelské jméno] /add“, kterým přidáme uživatele do skupiny administrátorů.

3.6.5. Další praktické příkazy

Pokud se připojujeme z počítače, kde není IoTCoreWatcher, a potřebujeme zjistit jméno počítače za běhu aplikace, tedy v době, kdy není vypsáno na zobrazovací jednotce, můžeme si pomoci příkazem „hostname“, který nám vrátí jméno počítače, ke kterému jsme připojeni.

Mnoho ze základních síťových utilit je dostupných i ve Windows 10 IoT Core, jako např. asping.exe, netstat.exe, netsh.exe, ipconfig.exe, nslookup.exe, tracert.exe, a arp.exe. Tyto nástroje se vyvolají napsáním jména a následně se s nimi pracuje obdobně jako v příkazové řádce či PowerShellu na stolních Windows.

Pro práci se soubory lze využít nástroje xcopy.exe, jehož možnosti vyvoláme příkazem „xcopy.exe /?“. Příkaz se opět chová totožně jako na stolním počítači.

Pro zobrazení seznamu aktuálních startovních aplikací použijeme „startup.exe /d“. Pro přidání aplikace do automatického spuštění užijeme příkaz „start.exe / [Jméno] [Příkaz]“. Nápovědu vyvoláme pomocí „startup.exe /?“.

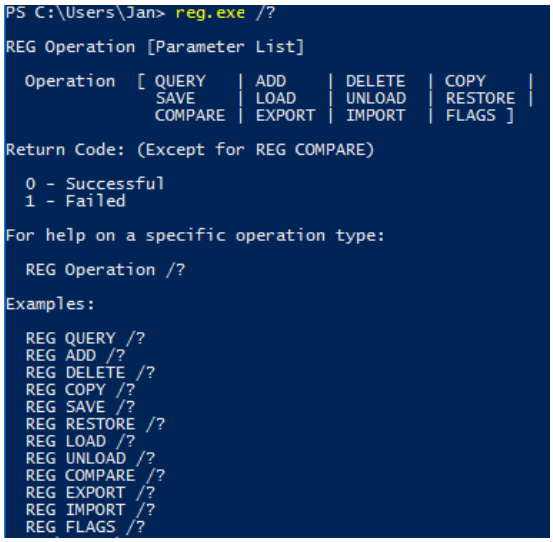
Obdobně jako u Linuxu je možné i Windows 10 IoT Core provozovat v grafickém režimu, nebo v režimu bez grafického výstupu. Tyto režimy se dají přepínat pomocí příkazu „setbootoption.exe [headed | headless]“, kde headed znamená grafický režim a headless znamená režim bez grafického výstupu. Tento příkaz není možné provozovat na žádném jiném systému z rodiny Windows 10.

K zobrazení aktuálních plánovaných úloh slouží příkaz „schtasks.exe“, který má oprávnění spustit jen administrátor. Pokud chceme do plánovaných úloh něco přidat, použijeme příkaz „schtasks.exe /Create“, pro okamžité spuštění určité úlohy „schtasks.exe /Run“ apod. Celý přehled možností získáme pomocí příkazu „schtasks.exe /?“.

Přístrojová utilita konzoly je užitečná v rozpoznávání a ovládání nainstalovaných zařízení a řidičů, a vyvolá se příkazem „devcon.exe“ doplněným o parametry, jejichž plný seznam zobrazíme příkazem „devcon.exe /?“. Příkaz nefunguje na ostatních systémech rodiny Windows 10.

3.6.6. Práce s registry OS

Pokud potřebujeme zpřístupnit registry, aby se zobrazily, nebo modifikovaly, je možné užít příkaz „reg.exe“ doplněný o parametry dle obrázku 28. Tento příkaz pracuje na všech zařízeních s Windows.



Obr. 28: Přehled parametrů příkazu „reg.exe“

3.6.7. Konfigurace služeb

Pro práci se službami systému máme několik možností. Úplně nejzákladnější výpis systémových služeb získáme pomocí příkazu „get-service“. Pro výpis aktuálně spuštěných systémových služeb můžeme užít příkaz „net start“. Pro spuštění služby můžeme užít například „net start jmeno_sluzby“ a pro zastavení služby „net stop jmeno_sluzby“. Další alternativou je užití nástroje sc.exe, jehož možnosti si vypíšeme příkazem „sc.exe /?“.

3.6.8. Úprava spouštění OS

Pokud bychom z nějakého důvodu potřebovali upravit spouštění systému, poslouží nám nástroj bcdedit.exe, jehož možnosti vyvoláme pomocí příkazu „bcdedit.exe /?“. Konfigurační možnosti jsou obdobné jako u starších systémů Windows v souboru boot.ini a jsou ještě o něco rozšířené. Toto je již opravdu třešnička na dortu, protože jde o něco, co v případě Windows 10 IoT Core není příliš potřeba.

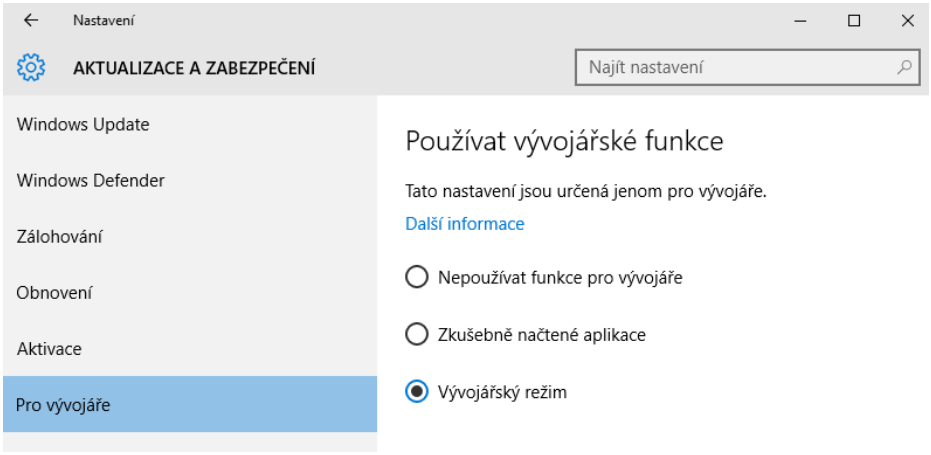
Protože je PowerShell velmi mocný nástroj pro práci s Windows 10 IoT Core, rozhodl jsem se v příloze A uvést přehled příkazů pro získání informací ze systému, tedy příkazů začínajících klíčovým slovem „get“.

3.7. Aplikace pro Windows 10 IoT Core

Jak jsem již zmínil, ve Windows 10 IoT Core se používají Windows Universal Apps. Psaní těchto aplikací je velmi blízké tzv. „Metro aplikacím“ pro Windows 8 a 8.1. Společnost Microsoft se rozhodla, že psaní těchto aplikací přiblíží programátorům formou velkého množství příkladů na GitHubu, kde je možné nalézt aplikace od úplně jednoduchých, jako je „Hallo world“, až po náročné projekty, jako je třeba domácí automatizace. Tyto příklady jsou doplněny o spoustu konferencí, jejichž záznamy lze nalézt i na serveru youtube.com. Chtěl bych upozornit, že tyto aplikace, ač mají hodně podobností s předešlou generací, jsou již opravdu univerzální a jsou odlišné od předešlé generace, i ve Visual Studiu se volí jiný typ projektu. Takovým na první pohled viditelným rozdílem je, že na rozdíl od předchozí generace nejsou tyto aplikace závislé na šablonách, jejich vzhled je již plně volitelný, ale kdo chce, může šablon využít.

Nadále platí, že pokud chceme svoji aplikaci umístit do Store, musíme vlastnit ověřovací kód. Ověřovací kód mohou studenti získat na Dreamsparku[5], kde je možné si stáhnout další pomoc v podobě GitHub Student Developer Pack a pro Visual Studio 2015 i Xamarin for Students. Ověřovací kód bohužel potřebujeme v případě, že aplikaci chceme užít ve Windows 10 IoT Core, ač tento systém nemá přístup do Store. Ověřovací kód nepotřebujeme v případě, kdy aplikace nasazujeme na počítače, které pracují v režimu „Zkušebně načtené aplikace“, nebo „Vývojářský režim“. Na počítači, kde jsou vyvíjeny aplikace, bych doporučoval mít zapnutý jeden z těchto režimů, ideálně „Vývojářský režim“, kde se dá pracovat i s operačním systémem jako takovým. Tyto režimy je možné zvolit v nastavení, tedy cestou start => nastavení => aktualizace a zabezpečení => Pro vývojáře (viz obrázek 29).

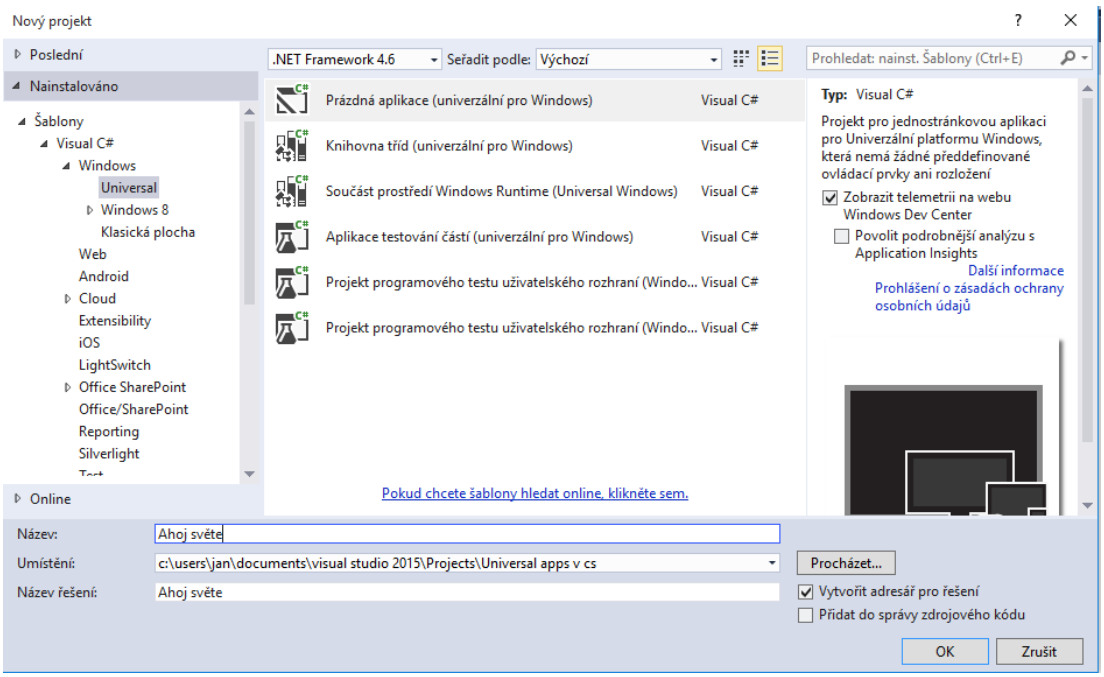
[5] Na MSDN AA, který vlastní škola registrovaná do programu (střední či vysoká), nebo na adrese: https://www.dreamspark.com/Student/Software-Catalog.aspx, kde je uveden jako Windows Store Developer Account.



Obr. 29: Aktivace vývojářského režimu v systému Windows 10

Nyní si ukážeme, jak správně založit projekt Windows Universal Apps a vytvořit aplikaci pracující na Windows 10 IoT Core. Na projektu aplikace „Ahoj lidi“ si ukážeme vše podstatné kolem ovládání Visual Studia 2015, které budeme potřebovat i pro složitější aplikace běžící na Windows 10 IoT Core.

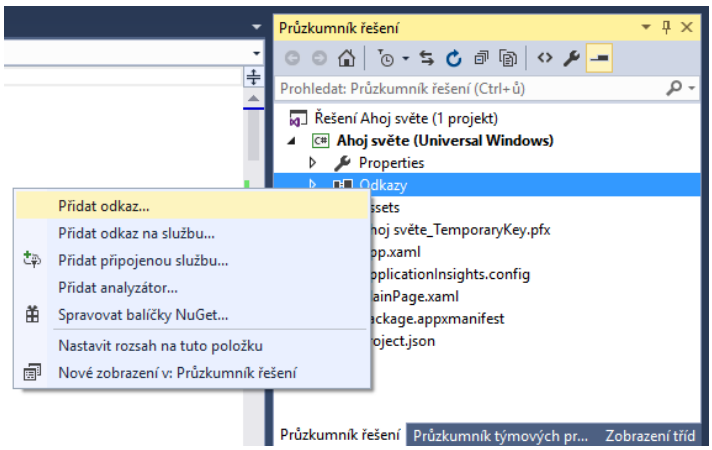
Při vytvoření nového projektu Visual Studia si zvolíme jazyk (C++, nebo C#), ten rozklikneme, vybereme Windows => Universal a položku prázdná aplikace (viz obr. 30). Následně si pojmenujeme aplikaci a zvolíme si adresář, do kterého se projekt uloží. Net Framework můžeme nechat výchozí, nebo využít verzi 4.6.



Obr. 30: Založení projektu Windows Universal Apps ve Visual Studiu

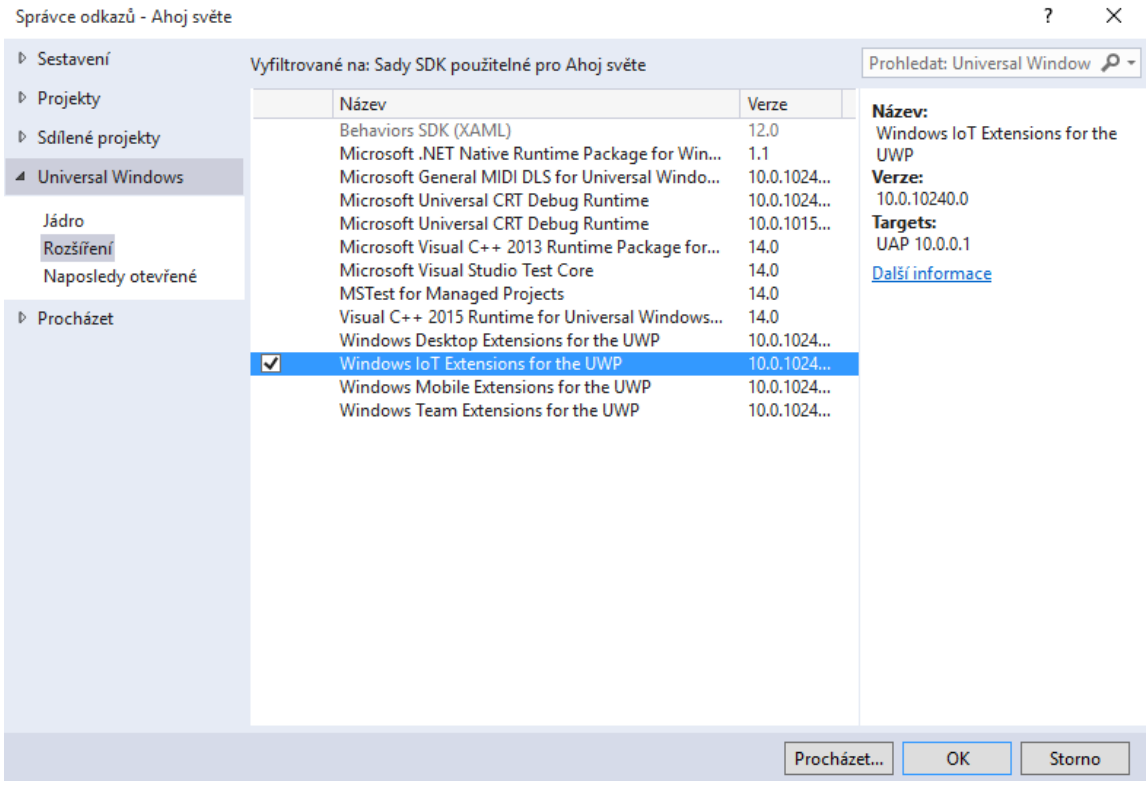
Pokud máme nastaveno, vše potvrdíme tlačítkem OK a počkáme, až Visual Studio vytvoří požadovaný projekt. Nežli se pustíme do vlastní práce na projektu, je potřeba ještě připojit reference, pokud se bude jednat o aplikaci pracující s GPIO sběrnicí, nebo aplikaci,

u které víme, že bude využívat senzory v telefonech či tabletech. Přidání těchto referencí provedeme kliknutím pravým tlačítkem na položku Odkazy v sekci Průzkumník řešení a zvolíme „Přidat odkaz“.



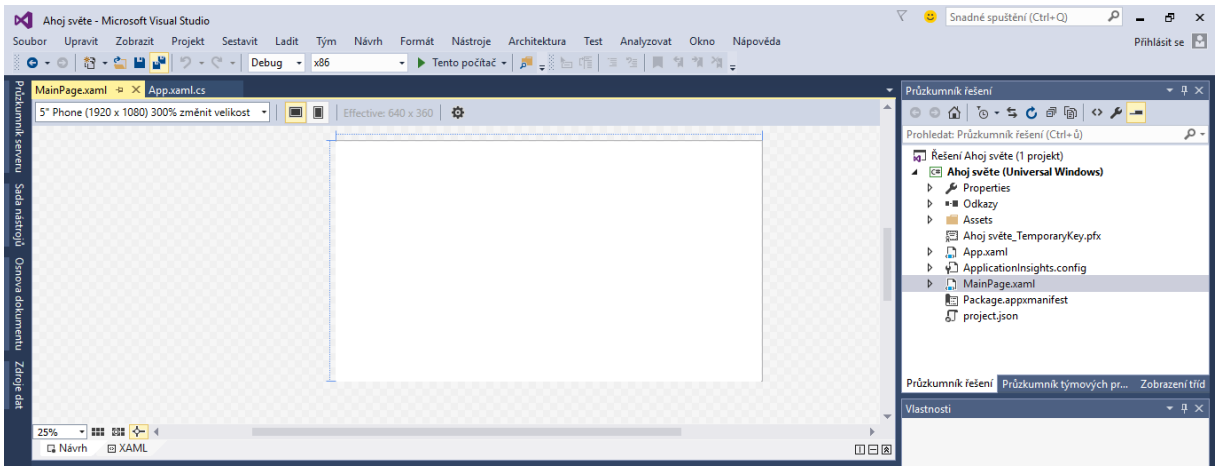
Obr. 31: Postup přidání referencí do projektu Visual Studia

V otevřeném dialogovém okně se přepneme v levém sloupci z položky Jádro na položku Rozšíření a zaškrtneme Windows IoT Extensions for the UWP. Na psaní pro Raspberry žádnou další rozšiřující knihovnu nepotřebujeme, pokud to nevyžaduje náš algoritmus, pak již musíme vycházet ze znalostních databází na webu. Všechno potvrdíme tlačítkem OK.



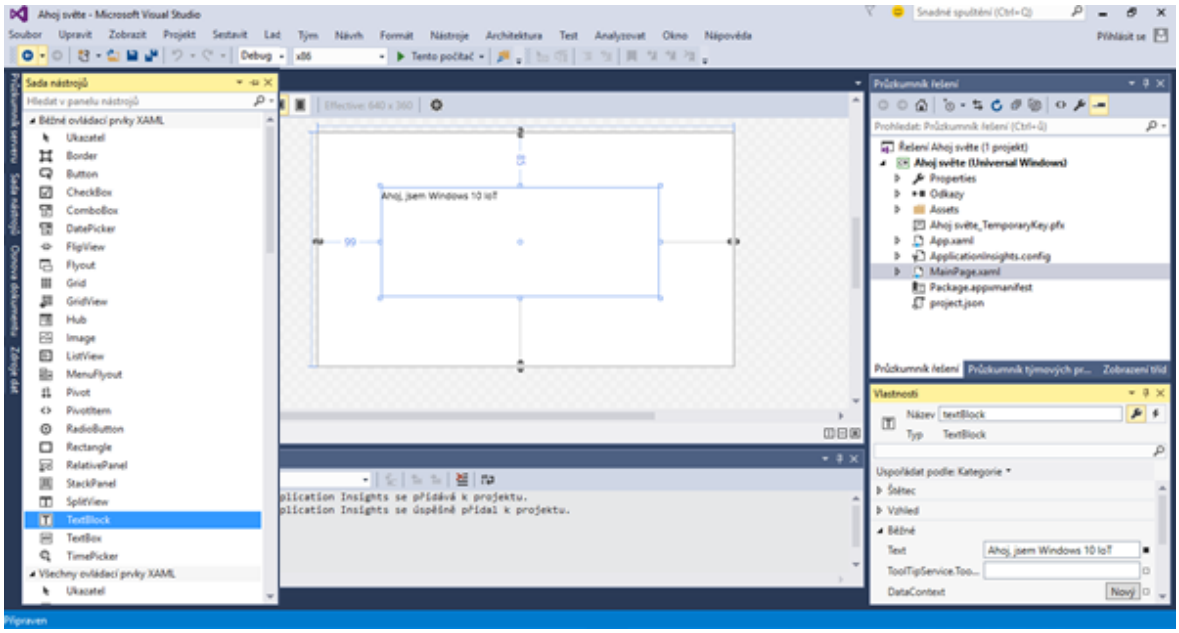
Obr. 32: Přidání referencí pro Windows IoT

Nyní máme již všechno připraveno k tomu, abychom mohli začít programovat. Jak jsem slíbil, ukážeme si jednoduchou aplikaci „Ahoj světe“, která vypíše text na monitor. Grafickou podobu aplikace vytvoříme buď v podobě příkazů v jazyce Xaml, nebo grafickým návrhem (nelze pod Win 8.1). Ukážeme si druhou cestu, takže si dvakrát klikneme v Průzkumníku řešení na MainPage.xaml a počkáme na načtení editoru.



Obr. 33: Grafický editor souboru MainPage.xaml

Zde si otevřeme sadu nástrojů (po levé straně) a zvolíme si TextBlock a jako v malování si vybereme oblast na „plátně“, kam chceme text umístit.

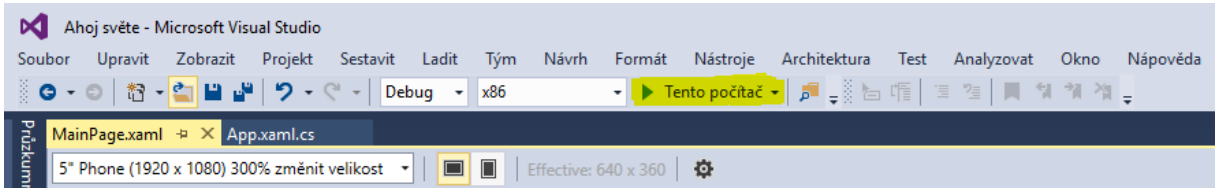


Obr. 34: Vložení textového bloku v návrhu MainPage

Vpravo dole (pod Průzkumníkem řešení) nalezneme okno Vlastností, kde vyplníme v záložce Běžné políčko text textem, který chceme zobrazit. Dále položku Běžné sbalíme kliknutím na šipku a otevřeme si záložku Text. Zde si zvolíme velikost, řez písma, zarovnání

apod. Nezapomeneme celý projekt uložit kliknutím na dvojici disket (nepoužijeme tentokrát jednu disketu, nebo ctrl+s, protože by se nám uložily změny jen v daném souboru).

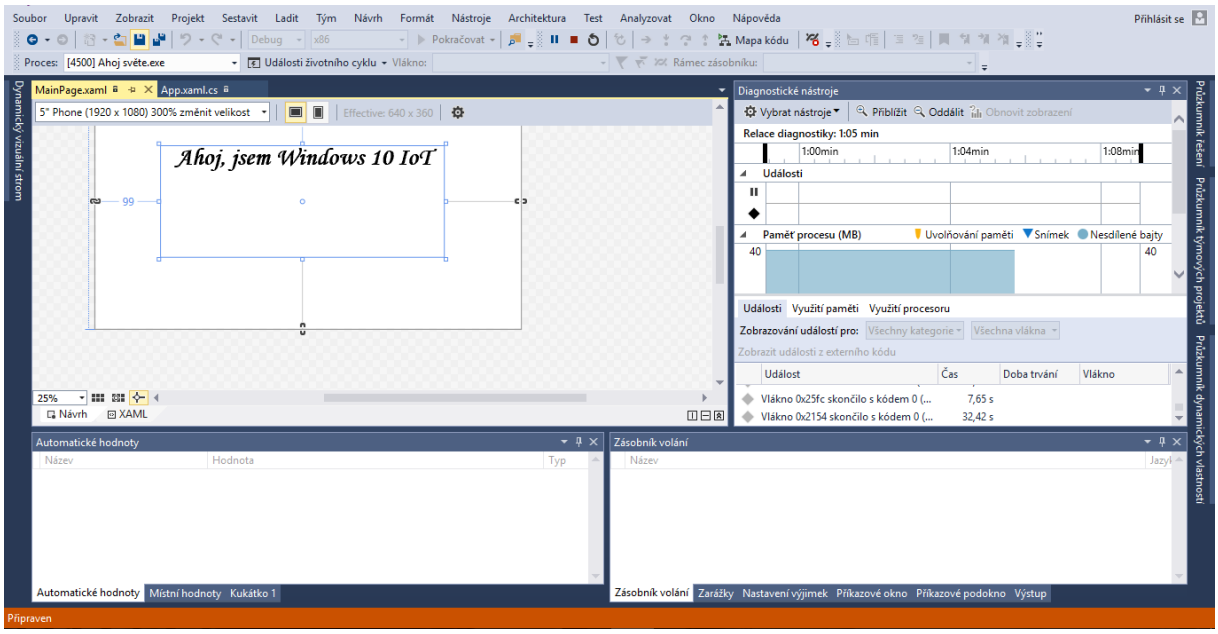
Nyní si můžeme naši aplikaci vyzkoušet na lokálním počítači, pokud jsme nikde neudělali chybu. Aplikaci spustíme pomocí zelené šipky nad editorem (viz obrázek).



Obr. 35: Spuštění ladění ve Visual Studiu

Pro otestování aplikace na lokálním počítači necháváme výchozí nastavení ladění (viz obrázek), nebo jej můžeme přepnout rozkliknutím do režimu x64. Úspěšným laděním zároveň dojde k instalaci aplikace na náš lokální počítač a nalezneme ji ve startu mezi všemi aplikacemi, pokud máme systém nastavený v jednom ze dvou výše zmiňovaných režimů. Podotýkám, že u první kompilace na danou platformu (x64, nebo x86) trvá několik minut, nežli dojde k sestavení aplikace, protože Visual Studio si ještě instaluje podpůrné komponenty. Pokud vše provedeme správně, měly bychom získat aplikaci a rozložení Visual Studia dle následujících obrázků.

Obr. 36: Aplikace Ahoj světe při ladění

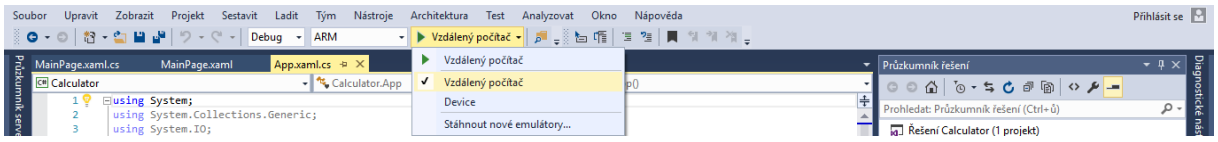


Obr. 37: Visual Studio při ladění aplikace Ahoj světe

3.7.1. Vzdálené ladění kódu

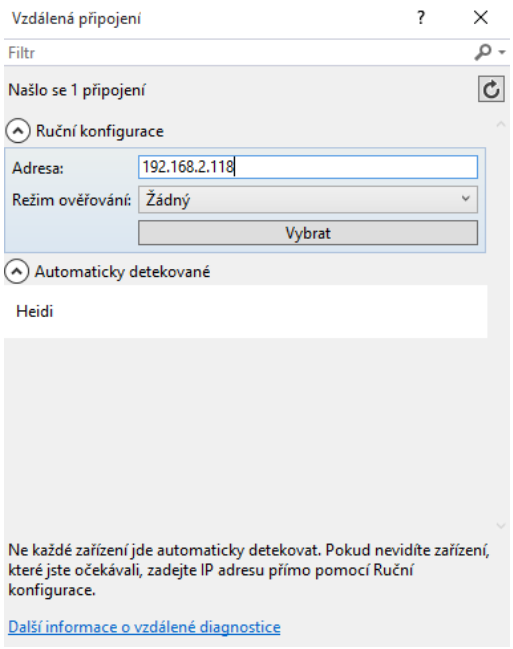
Před spuštěním vzdáleného ladění kódu je dobré si ověřit, že v systému běží „aplikace“ ZwaveHeadlessAdapterApp, což je aplikace zajišťující vzdálené ladění kódu z Visual Studia. Tato aplikace se automaticky spouští se startem systému a obyčejně není ukončována, jediná možnost jak ji ukončit, je chyba uživatele, který ji vypne pomocí webového prostředí, nebo pomocí PowerShellu. Pokud tato aplikace neběží, Visual Studio odpoví chybou nedostupného vzdáleného zařízení.

Prvním krokem je přepnutí Visual Studia do ARM (v případě MinnowBoard Max x86) a následně si rozklikneme nabídku u šipky, kterou spouštíme ladění, a zvolíme si vzdálený počítač (viz. Obr. 38).

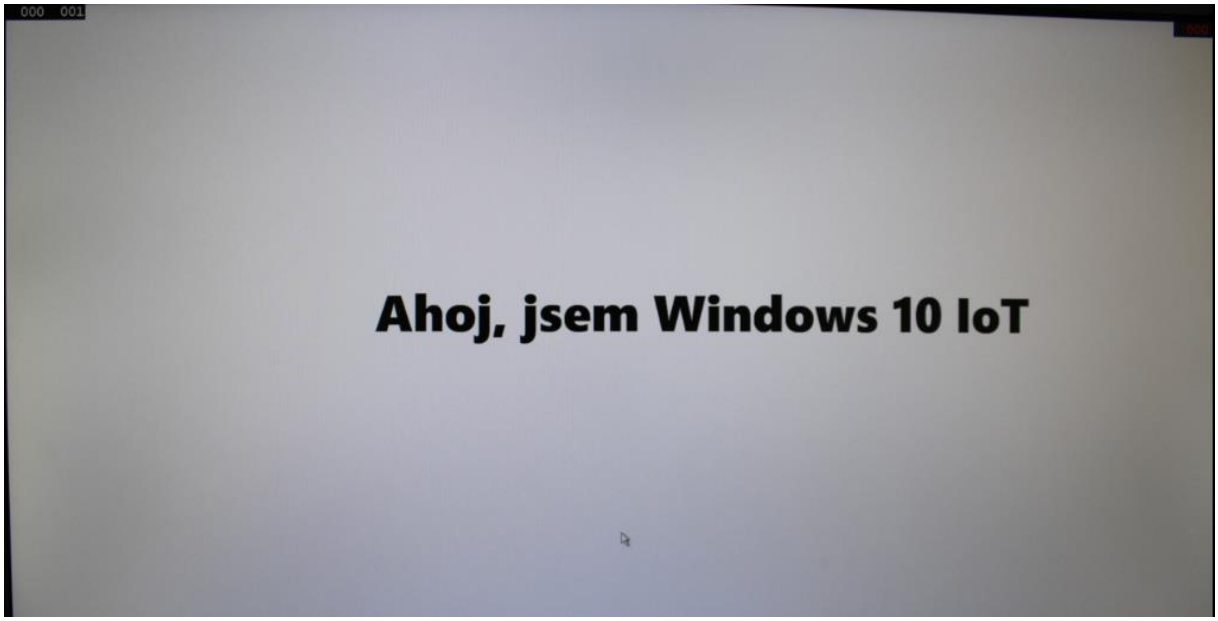


Obr. 38: Nastavení režimu vzdáleného ladění

Následně si nastavíme vzdálené zařízení, na kterém bude probíhat ladění. Vyplníme IP adresu (verze 4) a přepneme režim z „Windows“ do „Žádné“, jak je vidět na obrázku 39, a všechno potvrdíme tlačítkem „Vybrat“. Nyní je již všechno připraveno a můžeme spustit ladění stejně, jako na lokálním počítači.



Obr. 39: Nastavení vzdáleného ladění

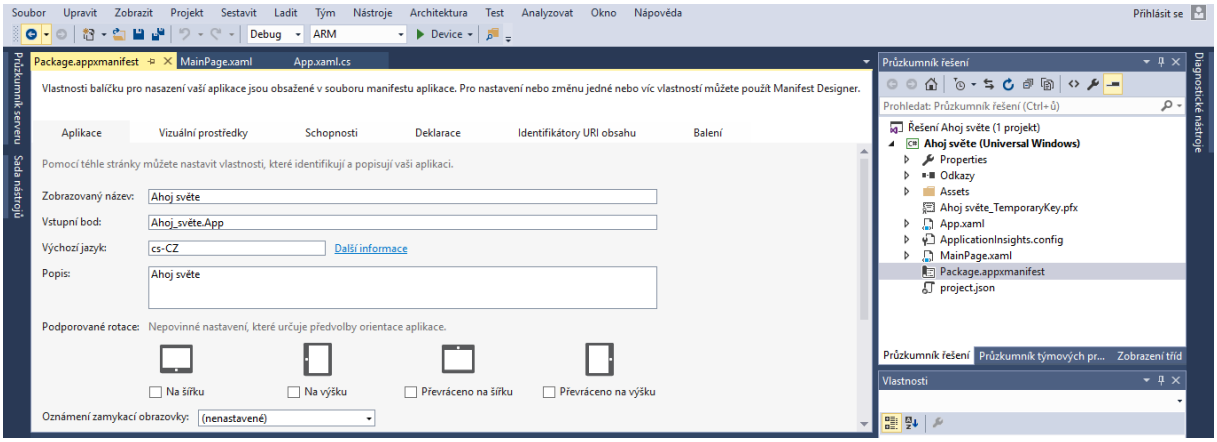


Obr. 40: Vzdálené ladění na zobrazovači Raspberry Pi

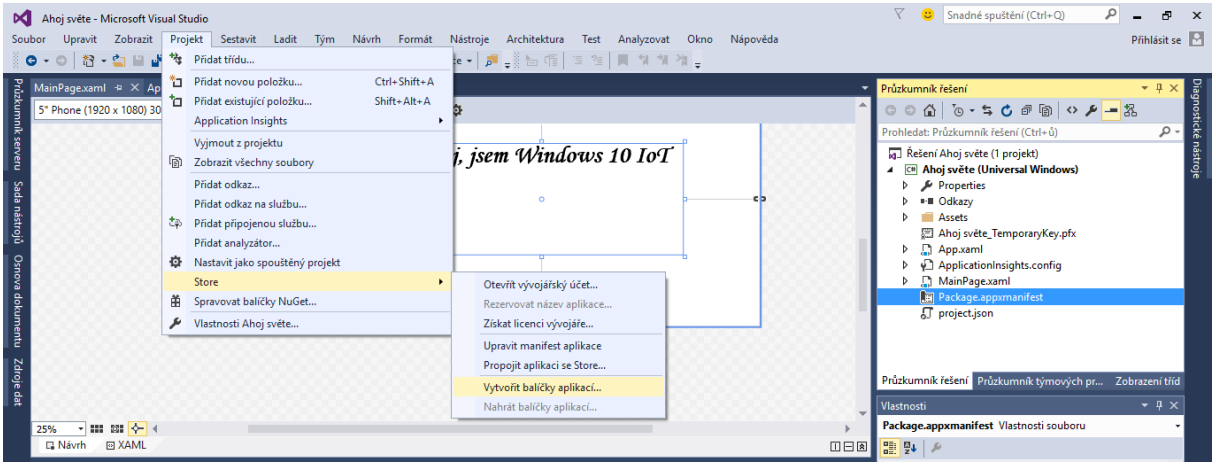
3.7.2. Vytvoření balíčku aplikace

Aplikaci máme napsanou a odladěnou, přišel čas vytvořit instalační balíček, který nainstalujeme do našeho Windows 10 IoT Core. Pro nahrání programu potřebujeme mít aplikaci ve formátu appx. Pokud chceme aplikaci blíže nastavit pro umístění do Store, provedeme nastavení pomocí dvojkliku na Package.appxmanifest (viz obrázek)

a jednoduchou konfigurací. Balíček aplikace získáme z Visual Studia velmi jednoduše (pokud máme vývojářský účet), stačí jen kliknout na nabídku Projekt, zvolit Store a vytvořit balíčky aplikací.

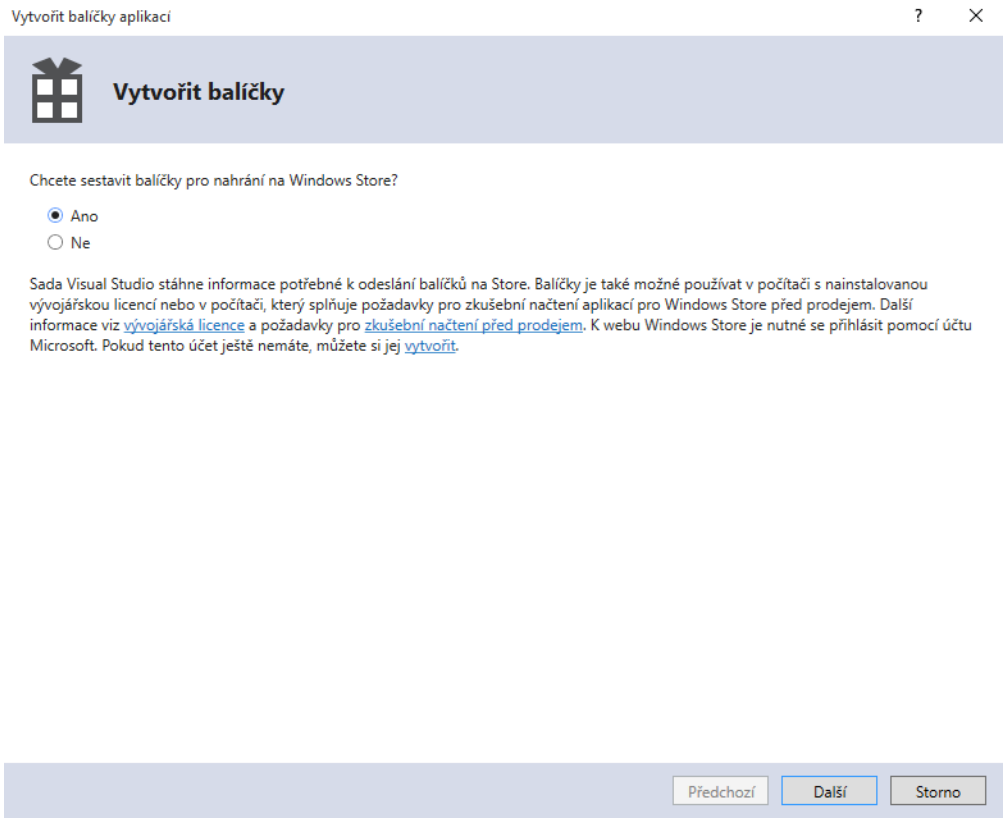


Obr. 41: Konfigurace aplikace pro umístění do Store

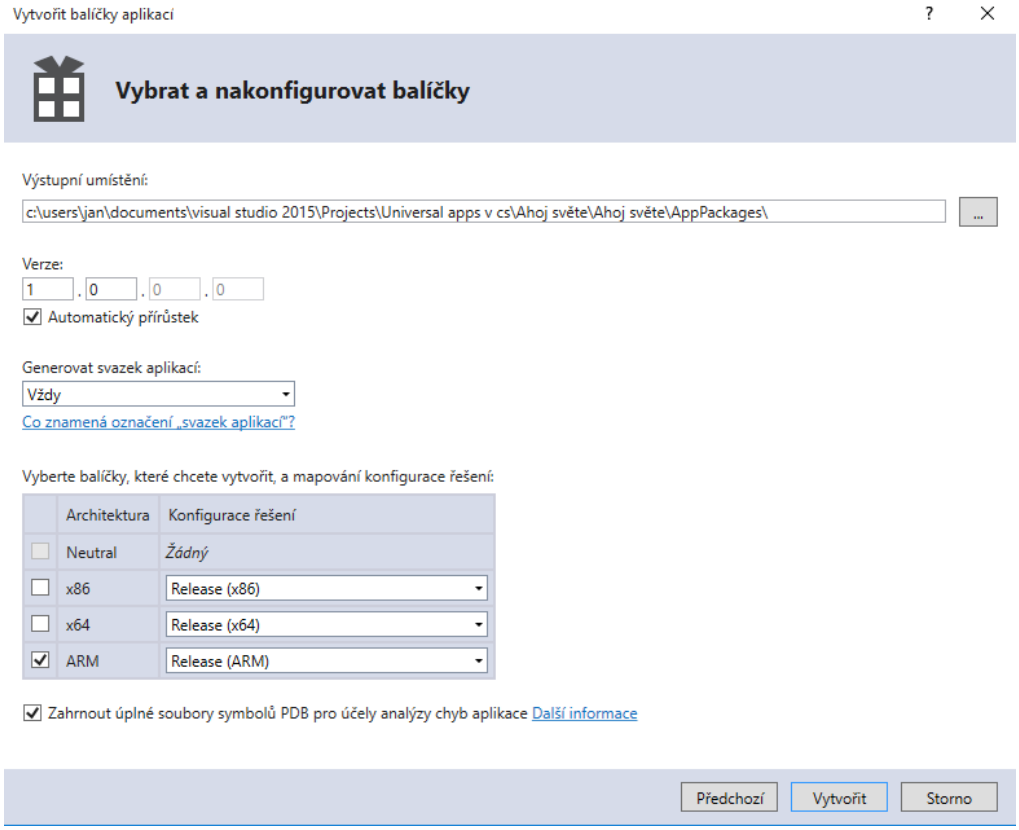


Obr. 42: Otevření průvodce pro tvorbu balíčku Windows universal Apps

Pokud opravdu chceme aplikaci použít jen pro Windows 10 IoT Core, průvodce nakonfigurujeme dle obrázků.

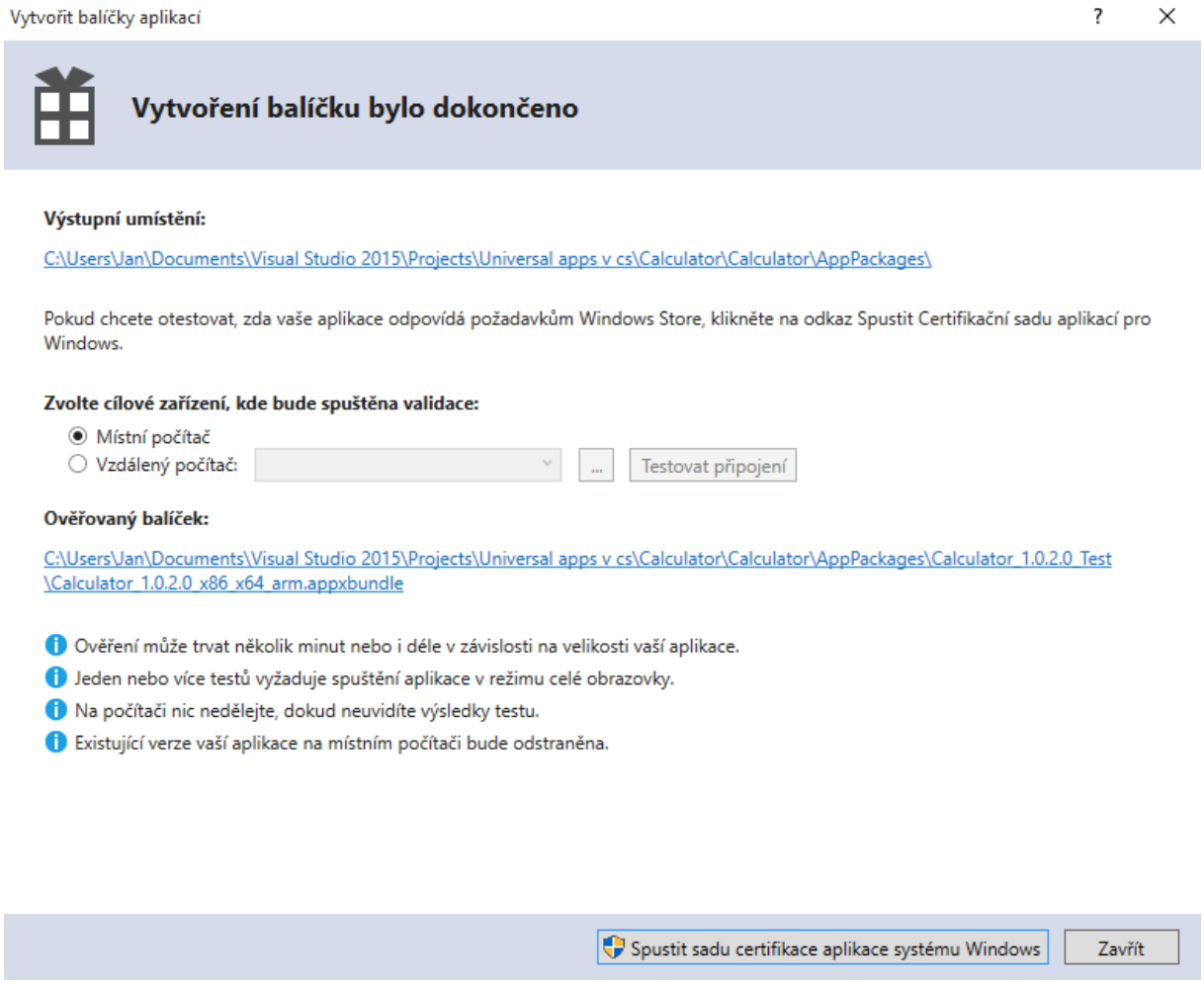


Obr. 43: Vytvoření balíčku aplikace krok 1



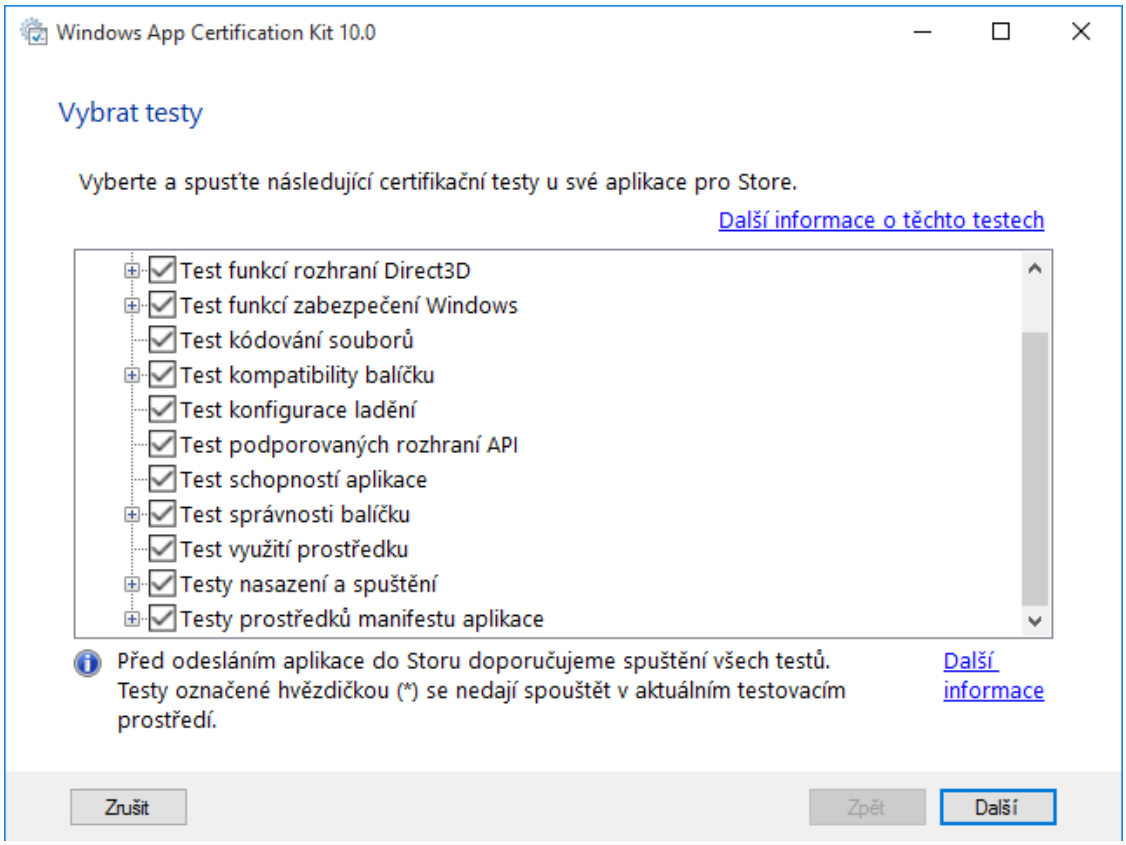
Obr. 44: Vytvoření balíčku aplikace krok 2

V kroku dva si zvolíme architekturu (můžeme klidně ponechat všechny), vyplníme verzi a nastavíme typ konfigurace řešení. Nic nám nebrání ponechat původní nastavení na Debug, ale platí tady, že konfigurace řešení musí být jednotná pro všechny vytvářené architektury. Všechno završíme kliknutím na tlačítko Vytvořit. Po dokončení procesu se nám zobrazí tabulka pro vytvoření certifikátu k aplikaci (pokud se jedná o konfiguraci řešení Release).



Obr. 45: Vytvoření certifikátu aplikace

Při vytváření certifikátu dojde k otevření sady automatizovaných testů, které prověří aplikaci, zda splňuje podmínky Store. Doporučuji provést všechny testy, pokud aplikace zůstává pouze na našem Windows 10 IoT Core, můžeme si dovolit zrušit některé testy, o kterých víme, že je nebudeme potřebovat.



Obr. 46: Nastavení automatizovaných testů aplikace při generování certifikátu

Po dokončení testů budeme mít k dispozici kompletní výsledky a v případě hodnocení „Neúspěšné“ budeme mít popis chyby a nástin možného řešení chyby. K tomuto html souboru je možné se kdykoliv vrátit, protože je uložen v adresáři projektu. Pokud zvolíme jako konfiguraci řešení Debug, testy se nám nenabídnou, balíčky jsou včetně certifikátů vygenerovány rovnou.

3.7.3. Přímá instalace pomocí Visual Studia

K tomuto způsobu nebudeme potřebovat vývojářský účet. Jediné, co je potřeba, je Visual Studio a upravit pár vlastností projektu před tím, než začneme ladit na Windows IoT. Otevřeme si vlastnosti projektu (viz. obr. 41), kde na záložce „Balení“ změníme název balíčku z předdefinovaného na vlastní, který je shodný s názvem aplikace. Následně stačí jen nastavit vzdálené zařízení, stejně jako tomu bylo při ladění aplikace.

Po spuštění ladění počkáme, až se nám aplikace spustí, a vyzkoušíme všechny její funkce. Pokud aplikace funguje, jak má, přejdeme do administrace systému pomocí webového prohlížeče (viz. obr. 18) a přejdeme na záložku Apps (viz. obr. 20), kde uvidíme ve spuštěných aplikacích aktuálně běžící aplikaci. Otevřeme si seznam všech aplikací, kde nám aplikace běží, vybereme ji a klikneme na tlačítko „Set Default“. Pokud se nás systém

dotáže na restart (může se lišit v závislosti na použitém Buildu), tak se zobrazeným hlášením ukončíme Debug ve Visual Studiu a povolíme restart pomocí webového rozhraní.

Po dokončení restartu by mělo dojít k načtení aplikace, kterou jsme si instalovali, a doporučuji opět aplikaci vyzkoušet. Když jsme si jistí, že aplikace pracuje, jak má, tak si můžeme pomocí webového rozhraní změnit výchozí aplikaci zpět na původní, pokud to tak chceme, naše nově přidaná aplikace již zůstane uložena v zařízení.

Tento postup má tu výhodu, že nepotřebujeme vlastnit vývojářský účet a vše nám proběhne na naší lokální síti. Další výhodou je, že se nezdržujeme nasazením testů aplikace a nemusíme tudíž řešit, jestli jsme v balíčku náhodou nezapomněli certifikát. Nevýhodou tohoto postupu je, že pomocí Visual Studia jej musíme opakovat při každé reinstalaci a také při nasazování na další zařízení v rámci společnosti. S již zmiňovaným souvisí další nevýhoda, která spočívá v horší přenositelnosti aplikace, protože na místo appx balíčků musíme kolegům či uživatelům podstoupit celý projekt Visual Studia, a z toho plyne nutnost, aby všichni, kdo budou chtít nasadit naši aplikaci, měli odpovídající verzi Visual Studia a podpůrných knihoven. Dále, co již považuji za jisté, znají i tento postup. Jak je vidět, tak tento postup je vhodný pro samostatné vývojáře, pracující samostatně na volné noze, nebo pro kutili či studenty.

3.7.4. Aplikace v jazyce Python

Systém Windows 10 IoT Core podporuje kromě Windows Universal Apps také možnost psát aplikace v jazyce Python, což je interpretovaný jazyk. Dle oficiálních zdrojů je Python určen k psaní Core aplikací, tedy aplikací bez grafického uživatelského rozhraní.

Pro psaní aplikací v tomto jazyce je vyžadováno Visual Studio 2015 update 2 s doplňky pro vývoj v Pythonu, Windows 10 IoT Core verze 10.0.10586.0 a novější, Python verze 3.x, Windows SDK verze 10586 a Python UWP SDK (zatím pouze Alfa verze[6]). Aplikace v Pythonu je sice možné psát i s pomocí Visual Studia 2013, ale oficiální[7] informace nehovoří o možnosti jeho využití pro Windows 10 IoT Core.

Při zakládání projektu volíme v sekci ostatní jazyky Python a typ projektu Windows IoT, další práce s Visual Studiem je obdobná jako v jiných jazycích.

Pro ladění na vzdáleném zařízení platí následující postup. Opět jako v předešlém případě zvolíme architekturu v závislosti na jednodeskovém počítači a režim vzdáleného

[6] Stav ke dni 24.7.2016 je Python UWP SDK verze 1.3 Alfa

[7] https://developer.microsoft.com/en-us/windows/iot/win10/samples/python [24.7.2016]

ladění (viz obrázek 40). Následně klikneme pravým tlačítkem v průzkumníku řešení na název projektu a z kontextového menu zvolíme vlastnosti. Ve UWP nastavíme jméno zařízení (nebo jeho IP verze 4) a číslo portu na 5678. Jako ve kterémkoli jiném případě můžeme začít ladit.

Hotovou aplikaci je možné nasadit s využitím Visual Studia velice jednoduše, stačí pouze přepnout režim z Debugg na Release a opakovat postup pro vzdálené ladění.

3.7.5. Ladění a nasazení UWP v C++

Vytvoření projektu a jeho vývoj je obdobný jako C# a VB (viz kapitola 3.5.1), ale vzdálené ladění na zařízení s Windows 10 IoT Core, se liší. V C++ je potřeba nastavit režim, architekturu systému a vzdálené zařízení (viz obrázek 38). Následně v průzkumníku řešení (Solution Explorer) klikneme na název projektu s ikonou jazyka (tučně) pravým tlačítkem myši a zvolíme vlastnosti. Ve vlastnostech si otevřeme vlastnosti konfigurace (druhé odshora) a otevřeme záložku ladění. Zkontrolujeme, případně opravíme, režim debuggeru, který by měl být vzdálené zařízení. Následně vyplníme jméno zařízení (případně IP adresu verze 4) do položky jméno zařízení a jako typ ověření zvolíme univerzální nešifrovaný protokol. Nyní je vše připraveno ke spuštění ladění a případné instalaci aplikace pomocí Visual Studia, jak je popsáno v kapitole 3.7.3, případně k následné tvorbě balíčků (viz kapitola 3.7.2).

3.8. Srovnání vývoje mezi verzemi 10531 a 14342

Za 7 měsíců vývoje, tedy od verze 10.0.10531 k verzi 10.0.14342.1000 odvedl Microsoft opravdu znatelný kus práce. Systém se dočkal mnoha změn a novinek, které opravdu výrazně zjednodušují uživatelům systému život. Výrazných změn nedosáhl jen systém jako takový, ale projevily se i v řídících aplikacích pro stolní počítač. Nyní si popíšeme postupně změny od právě řídících aplikací a proces instalace až po novinky, které poskytuje vlastní systém.

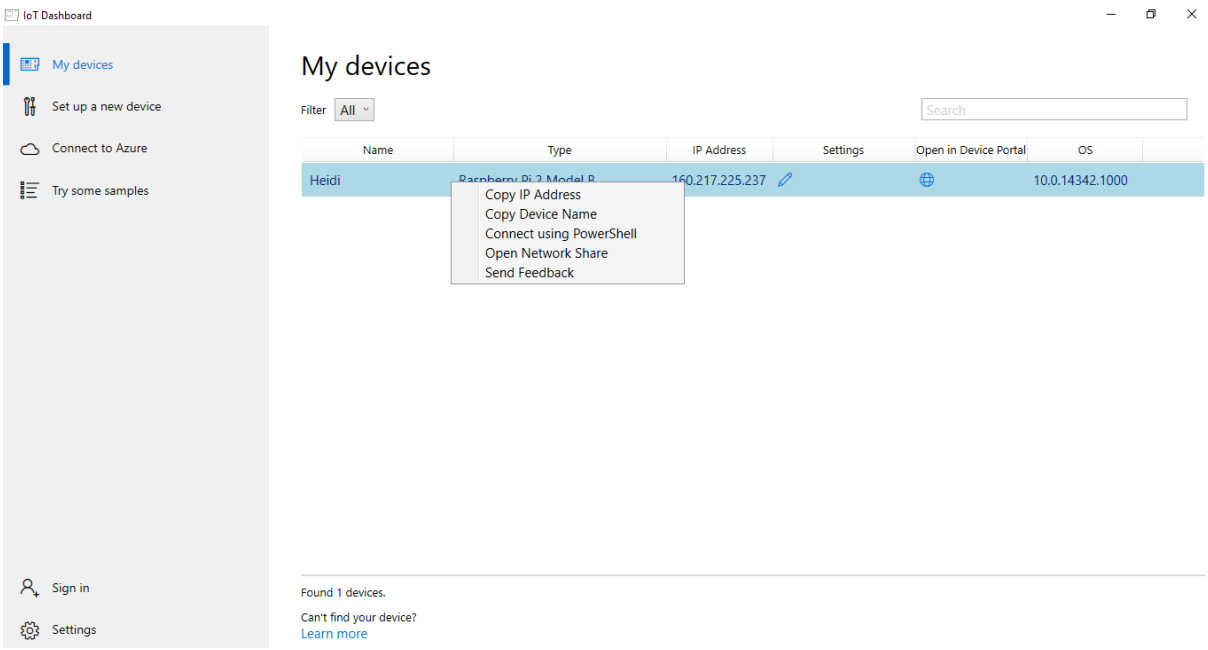
3.8.1. Novinky v řídících aplikacích pro stolní počítač

Zde se Microsoft rozhodl uživatelům výrazně zjednodušit práci vydáním aplikace Windows 10 IoT Core Dashboard, která slučuje a aktualizuje předcházející dvojici aplikací IoT Image Helper a Windows IoT Core Watcher, který od verze 10.0.14342.1000 zcela nahrazuje (pro starší verze systému lze nadále používat aplikaci Windows IoT Core Watcher). Zde je potřeba pochválit Microsoft za celkové zpracování aplikace, které převzalo

filosofii z aplikace Server Manager Dashboard. Další kladné body z pohledu uživatele je nutné Microsoftu připsat za implementaci technologie, která se osvědčila u telefonů Nokia (později Microsoft) Lumia, tedy za integrovaný proces instalace systému, který funguje v aplikaci Windows Recovery Tool. Též integrace Azure je praktickým krokem stejně jako možnost přímé instalace zkušebního příkladu aplikace do zařízení bez nutnosti spuštění Visual Studia, což se může hodit například při seznamování se s ovládáním systému, nebo při testování, zda se systém instaloval opravdu správně a je zcela funkční.

Poslední z výčtu zlepšení, které se podařilo nové aplikaci dosáhnout, je zpráva připojení, kde se uživatelé dočkali přímého vyvolání PowerShell relace, či přímého vyvolání nastavení sdílení sítě.

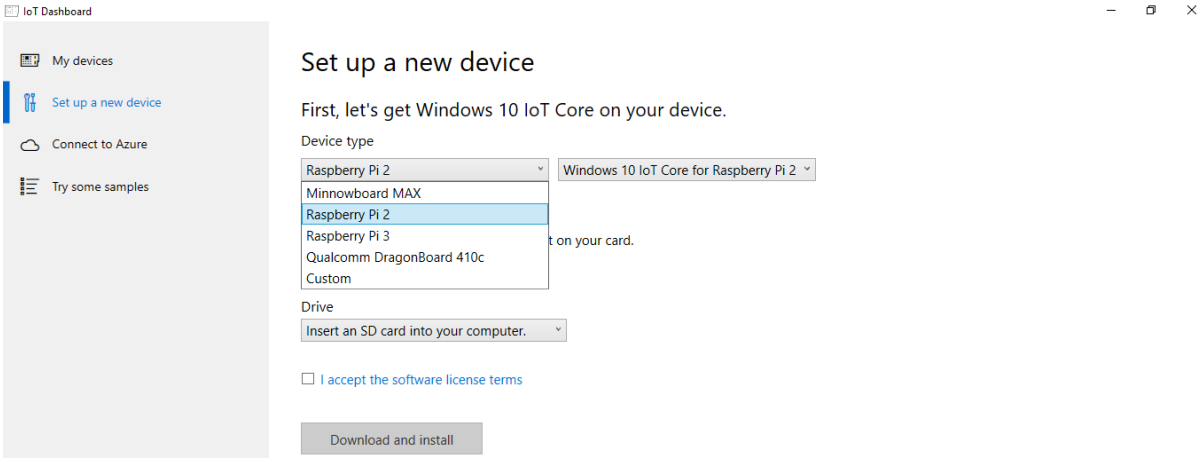
Abych jen nechválil, tak i novou aplikaci provází některé neduhy. Prvním neduhem je, že aplikace je oficiálně podporovaná jen pro Windows 8 a novější (nepodporuje serverové verze systému), dalším neduhem jsou časté požadavky na připojení k internetu, které mohou snižovat komfort při práci s aplikací v uzavřené interní síti. Posledním neduhem, který trochu s touto aplikací souvisí, je fakt, že nové verze systému již nedokáží komunikovat s aplikací Windows IoT Core Watcher, což uživatele nutí přejít k nové aplikaci, i když používají Image Helper a stahují si instalátory v podobě ISO souborů.



Obr. 47: Možnosti připojení se k Windows 10 IoT v aplikaci IoT Dashboard

3.8.2. Novinky v instalaci systému

Novinkou je možnost použít aplikaci IoT Dashboard, která umožňuje dva způsoby instalace. Jedním je volba Custom, která vede na stejný způsob práce jako s Windows IoT Image Helper a je určena pro instalaci buildů Insider, případně starších buildů systému pomocí souboru ffu. Novinkou je možnost přímé instalace aktuální verze release větve systému online obdobně, jako je tomu u telefonu s WP. Rozdílem proti telefonům je, že musíme zvolit typ desky, pro kterou chceme připravit paměťovou kartu se systémem.



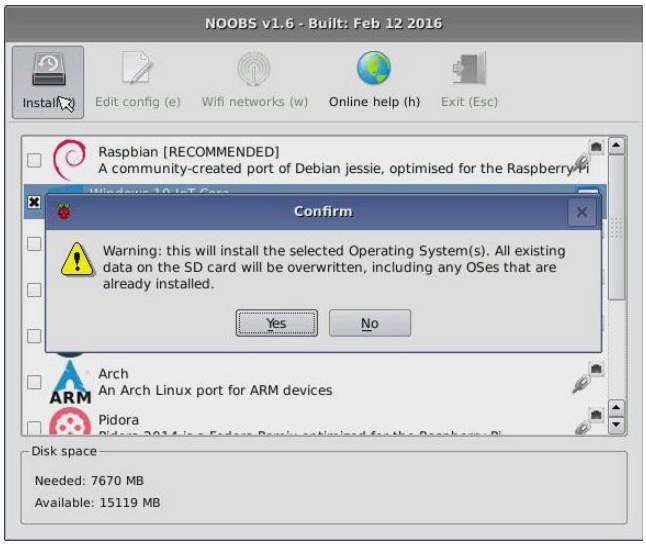
Obr. 48: Instalace systému pomocí IoT Dashboardu

3.8.2.1. Instalace pomocí NOOBS[8]

Další novinkou, která přišla krátce po vydání verze 10.0.10531, je možnost instalace systému pomocí NOOBS, která se mírně liší pro Pi 3 od ostatních zařízení.[9] Nejdříve si ukážeme využití NOOBS pro ostatní zařízení a následně si řekneme, v čem se liší jeho použití na Pi 3. Po spuštění NOOBS vybereme Windows 10 IoT Core, klikneme na tlačítko Install a potvrdíme bezpečnostní varování.

[8] BOOBS je zkratka pro New Out of Box Software

[9] Stav ke dni 19.7.2016, kdy je pro Pi3 k dispozici pouze Insider větev systému.



Obr. 49: Instalace Windows 10 Iot Core pomocí NOOBS. Zdroj: Setup with NOOBS: Choose Windows 10 IoT Core in NOOBS [online]. Redmond, 2016 [cit. 2016-07-19]. Dostupné z: https://developer.microsoft.com/cs-cz/windows/iot/win10/Noobs.htm

Instalace pokračuje výběrem větve OS, tedy jestli chceme RTM nebo Insider. Bez registrací je možné instalovat RTM. Nyní stačí jen počkat na stažení systému a odsouhlasením dokončit instalaci, pokud jsme zvolili RTM. Pokud zvolíme Insider, dostáváme se do stejného postupu, který platí i pro Pi3.

Pokud instalujeme Pi3, máme na výběr jen možnosti Insider, jsme přesměrováni na stránku pro přihlášení k našemu Microsoft účtu. Pokud ještě nejsme v Insider programu pro IoT, zobrazí se nám formulář registrace do programu. Pokud je třeba se vrátit v libovolném bodě na předchozí stránku v tomto pohledu prohlížeče, stačí kliknout pravým tlačítkem myši a vyberte položku Go Back z kontextového menu. Po vyplnění formuláře se nám zobrazí stránka stažení Windows 10 IoT Core Insider Preview a tlačítkem Confirm potvrdíme výběr větve systému. Dále z rozbalovacího menu vybereme příslušný typ jednodeskového počítače a opět potvrzujeme tlačítkem Confirm. Pokud vše proběhne úspěšně, zobrazí se nám číslo buildu a datum konce platnosti časové známky (datum, do kterého nejdéle je nutné stáhnout systém), stahování zahájíme kliknutím na tlačítko Download Now. Nyní již stačí jen potvrdit licenční ujednání.

Pokud bychom měli jakékoli problémy s instalací systému, je možné je vyřešit pomocí online průvodce instalací, který je k dispozici na adrese: https://developer.microsoft.com/cs-cz/windows/iot/GetStarted a provede nás instalací systému od výběru jednodeskového počítače, přes instalaci OS a Visual Studia až po instalaci první zkušební aplikace.

3.8.3. Novinky v systému pro build 10.0.14342.1000

Nejen na instalaci a řídící aplikaci v Redmondu zapracovali, i systém se dočkal opravdu velkého množství novinek, odbourání mnohých problémů i kosmetických změn designu řídící webové stránky. V této části si ukážeme některé novinky, kterých se systém dočkal.

3.8.3.1. Novinky ve webovém rozhraní pro správu OS

Na domovské stránce přibyla možnost změny časového pásma, nastavení rozlišení a obnovovací frekvence zobrazovací jednotky a její orientace. Na stránce Apps se krom změny podoby seznamu všech instalovaných aplikací objevila i možnost odstranění aplikačního balíčku a možnost nastavení startup chování aplikace.

Znatelných změn doznala i stránka Device Manager, která mimo změny designu do dalece přehlednější varianty, byla rozšířena o možnost aktualizace ovladačů, což je umožněno začleněním systému do služby Windows Update.

3.8.3.2. Nové funkce integrované do systému

Úplně novou funkcionalitou je průzkumník souborů, který nápadně připomíná první oficiální průzkumník souborů pro Windows Phone 8.

Novinkou v oblasti sítí je IoT Onboarding, což je integrace často vytvářené aplikace. Jde o sdílení připojení k internetu pomocí Wi-Fi, kdy IoT systém můžeme využít jako přístupový bod.

Mezi další novinky, které jen krátce zmíním, patří Windows Update, TPM 2.0, které je možné nastavit pro konkrétní sběrnici a Windows 10 IoT Remote Server, který umožní sdílení obrazového výstupu a interakcí s aplikacemi pomocí klientské aplikace, kterou je možné stáhnout pomocí aplikace Store do počítačů, tabletů a telefonů s Windows Phone, či Windows 10 Mobile. Tuto možnosti musíme povolit na stránce Remote.

Pokud jde o požadavky pro běh systému, ty zůstaly stejné, jen je potřeba počítat s tím, že nová verze systému zabere na kartě více místa nežli náš původní starý systém, s čím je potřeba počítat při provádění upgrade a zvážit, zda se nám budou všechna zálohovaná data vejít zpět na kartu, nebo je potřeba instalovat systém již na větší kartu.

4. Testování systému

4.1. Strategie testování

Celkově strategie testování vychází z postupů exploratorního testování. Tento přístup byl zvolen jako nejlepší možný z důvodu, že jediné, co je k dispozici, je již instalační médium a zprovozněný OS, proto není možné případná slabá místa odhalit analýzou zdrojového kódu. Přístup testování nazývaný „Black box“ zde není vhodný, protože systém je obecným OS, nejde o specializovaný systém, jako jsou například bankovní systémy. Při exploratorním testování dochází k seznámení testera s testovaným produktem v jeho průběhu, což je můj případ. Exploratorní testování s sebou nese riziko opomenutí otestování nějaké komponenty a nevýhodu v minimální možnosti použití automatizace. Tento druh testování je poměrně hodně závislý na zkušenostech a osobě testera a často bývá zaměňován s Adhock testováním. Exploratorní testování je dobré pro první fáze testování již ucelených systémů (především operačních) a testování uživatelských rozhraní, umožňuje poměrně jednoduše určit nejčastěji používané komponenty a odhalit, zda se otestovalo opravdu všechno
a navrhnout, co je potřeba blíže otestovat tak, aby se nic nevynechalo.

Windows 10 IoT Core je specializovaný IoT operační systém, že je potřeba se při testování zaměřit na specializované funkce, jako je práce s GPIO a I^2C sběrnicemi, nicméně to do velké míry ovlivňuje obslužná aplikace, což je potřeba vzít v potaz. Dále je pak potřeba se zaměřit na vzdálené řízení systému (nezávisle na běžící aplikaci), ale také na dostupnost aplikace spuštěné na systému v případě její chyby či chyby OS nebo HW, tedy jak rychle je systém schopen po kolapsu provést pokus o obnovu běhu aplikace a k jaké škodě při tomto obnovovacím procesu dojde.

Od IoT OS obecně očekáváme jednoduché a spolehlivé vzdálené ovládání, stabilitu běhu a nějaký automatizovaný postup reakce na chybu aplikace, vnitřní chybu systému či chybu HW, aby byla nejvyšší možná dostupnost spuštěné aplikace. Dále pak očekáváme vyřešení práce se specializovanými sběrnicemi, které nám konkrétní průmyslový počítač zpřístupňuje.

Strategie je poměrně jednoduchá, nicméně umožňuje dobře určit, co je otázkou systému a co již je otázkou aplikace, u které je snaha se důsledně držet pokynů Microsoftu, aby aplikace jako taková co nejméně ovlivňovala výkon celku.

Prvním bodem je tedy zjištění sledovaných parametrů při spuštění na prázdno

a ustálení systému (špička a ustálený stav), v systému běží jen nativní aplikace pro vzdálený přístup, základní aplikace systému a interní aplikace pro sběr dat.

Další důležitou částí testování je sledování parametrů při ladění (vzdálený debugg) různě náročných vzorových aplikací, protože jde o poměrně častou činnost jednak při vývoji samotných aplikací, při hledání příčin jejich nestandardního chování.

Neméně důležitým bodem testování je sledování parametrů při běhu různě náročných aplikací, aby bylo možné posoudit, na kolik je systém pružný a jak reaguje na různou míru zatížení.

Vzhledem ke specializaci systému je potřeba prověřit pomocí nejjednodušších možných aplikací práci se specializovanými sběrnicemi a rychlost reakcí systému na události generované sběrnicemi.

Chování systému jsem dále prověřil pomocí záměrně zacyklené aplikace, která neustále zabírala exponenciálním trendem paměť a udržovala procesor v zatížení nad 50 %. Cílem bylo zjistit, kde se nachází hranice kdy, již systém zasáhne a aplikaci ukončí.

Důležitým bodem je sledování kritických chyb, v případě tohoto typu systému především BSOD a také toho, zda se systém dokáže z BSOD probrat, či nikoliv, s čímž souvisí i monitoring stability aktualizačního procesu služby Windows update.

Poslední bod testování je výrazně nestandardní a spočívá ve sledování systému při dlouhodobém běhu aplikací, řešící některý z tradičních či méně běžných IoT scénářů. Tento test má simulovat reálné nasazení systému s konkrétním HW a není zde cílem sledovat spotřebu prostředků, ale plynulost a stabilitu celku tvořeného aplikací a systémem. Opravdovým cílem je prokázat, že systém je (či není) vhodný pro reálné nasazení a zda zvládne to, co jeho již dlouhá léta rozvíjený Linuxoví konkurenti v oblasti ryzího IoT.

4.1.1. Sledované parametry

U většiny testů jsem sledoval spotřebu systémových prostředků, tedy využití RAM a CPU. Dále se sledovala stabilita systému, evidovaly se veškerá zamrznutí a pády systému. Při specifických testech IoT scénářů jsem sledoval odezvu systému na vnější událost a také jsem sledoval fyzicky pomocí osciloskopu signály jdoucí na a z GPIO.

Dále jsem sledoval dobu potřebnou pro reload po chybě aplikace v závislosti na výchozí aplikaci, dobu potřebnou pro restart systému včetně načtení výchozí aplikace. Všechny tyto parametry jsou prvně ověřeny na systému bez doinstalovaných aplikací (systém nese jen předinstalované aplikace od Microsoftu).

Dalším netradičním sledovaným parametrem je spolehlivost vzdáleného debuggu projektu Visual Studia 2015, tedy jsou sledovány problémy s konektivitou a spolehlivost aplikace běžící v debugg modu.

4.2. Metodika testování

4.2.1 Test aplikace IoT scénářů

V tomto testu se na vzorových scénářích zkouší reálné nasazení systému v IoT scénářích. První scénář je velmi jednoduchou modelovou aplikací využívající I^2C sběrnici a senzor BMP180 k vytvoření naprosto jednoduchého teploměru, který zobrazuje teplotu a tlak v místě umístění senzoru, může to být jak kancelář (či jiný vnitřní prostor), tak venkovní prostředí. Vzorových programů, které řeší tento hojný problém, je celá řada na různé platformy, což mě vedlo k zařazení tohoto scénáře do testování. (zdrojové kódy jsou na DVD)

Druhá modelová aplikace simuluje informační stánek (bez dotykového ovládání) OC Chodov Praha. Vychází z webového prohlížeče použitého pro testování práce s internetem a je založena na použití online dostupné mapy centra, tedy pro svůj běh užívá webovou adresu obchodního centra. Rozdíl je v tom, že upravená aplikace neobsahuje adresní řádek, jen nabídku domů (tedy zobraz mapu). Aplikace webového prohlížeče je zde reálně zjednodušena na „přehrávač“ webových aplikací. (zdrojový kód je v příloze F)

V tomto testu se nehodnotí ani tak vytížení systému, které je zde hodně závislé na kvalitě aplikace, ale hodnotí se výsledné vyřešení scénáře, tedy to, jak si systém dokáže poradit i s méně kvalitní aplikací a tím pádem dokáže poskytnout požadované výstupy i méně zkušeným programátorům. Jde zde především o zkoušku toho, jak rychle dokáže systém komunikovat s aplikací a jak kvalitně aproximuje realtime zpracování problému.

4.2.2. Test práce s internetem

Vyhodnocuje se vytížení HW a rychlost prohlížení internetu, dále pak kvalita přehrávání online videí a případná omezení používání online služeb. Naměřené parametry jsou srovnány s parametry získanými na desktopovém Windows 10 (64 bitové architektury) pomocí shodného zdrojového kódu (stejné aplikace). Shodná aplikace zaručuje, že nebude měření zatíženo chybou způsobenou rozdílnou optimalizací kódu. Cílem testu je zjistit, zda je možné kvalitně, i na Windows 10 IoT, provozovat matematické a kancelářské nástroje běžící v cloudu, webové aplikace a podobně.

Čas potřebný pro načtení webové služby je poměrně dlouhý, protože aplikace nemá žádnou optimalizaci kódu (jde o nekvalitní aplikaci), proto jsem měřil čas pomocí ručních stopek. S ohledem na užitou aplikaci bych se měl pohybovat v nejméně příznivých časech a podmínkách pro práci samotnou. (zdrojový kód je v příloze G)

Testování obsahuje práce se službami Microsoft Office 365 a Office Online, kdy pro ukládání dat je využito uložiště Onedrive. Dále je otestována služba online výpočtů Wolframalpha.com. Do testování je zahrnuta i velmi populární služba youtube.com a práce s vyhledávači.

4.2.3. Test reakční doby na akci GPIO

Test má zjistit, jaká je průměrná odezva na akci provedenou zařízením připojeným přes GPIO s využitím nejjednodušší možné aplikace. Modifikoval jsem aplikaci HW Hallow Word tak, že LED svítí při stisku tlačítka, což trvá jen, dokud je tlačítko stištěno. Stisknutí tlačítka zapíše na pin logickou jedničku a úkolem aplikace je tento stav přerušením propsat na výstupní pin diody. Zpoždění dané vlastní aplikací je velmi malé, protože jde pouze o negaci hodnoty výstupního pinu při vyvolání přerušení, které je dáno změnou logického stavu na vstupním pinu. Pokud se na problém podívám jako elektrotechnik, celou aplikaci by šlo nahradit jedním kusem drátu, který by spojoval mikrospínač a diodu.

Po odzkoušení upravené aplikace pomocí vzdáleného ladícího programu jsem aplikaci nasadil pomocí Visual Studia. Po nasazení aplikace jsem do zapojení HW přidal sondy osciloskopu, jeden kanál na tlačítko a druhý kanál na diodu. Na osciloskopu jsem si zobrazil oba použité kanály a aktivoval si časové kurzory.

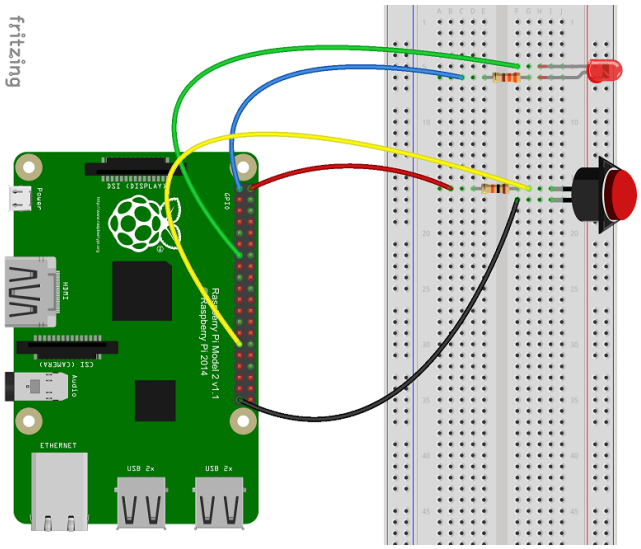
Test jsem započal spuštěním aplikace a vyčkáním na vypsání potvrzení o úspěšné inicializaci GPIO. Následně jsem si zkontroloval osciloskop a stiskl tlačítko. Po rozsvícení diody jsem okamžitě zamrazil obraz na osciloskopu. Nyní jsem na základě zobrazení obou kanálů pomocí časových kurzorů odečetl čas, který odděloval obě nástupné hrany u paty. Po odečtení jsem si čas zapsal do tabulky. Následně jsem osciloskop znovu spustil a měření zopakoval. Po 10 opakováních jsem restartoval Windows z důvodu, abych zamezil nějakému mimořádnému SW problému zkreslit výsledky. Následně jsem provedl dalších 10 měření.

Z naměřených hodnot jsem následně udělal průměr a určil chybu měření. Chybu měření jsem, po konzultaci s panem Ing. Martinem Šimůnkem z katedry měření elektrotechnické fakulty ČVUT, určil pomocí vztahu:

$$\delta = \frac{0{,}2}{sample\ rate} + stálost\_časové\_základny\ [ppm] \cdot naměřená\_hodnota \quad (1)$$

Kde 0,2 představuje jitter vzorkovaní. Všechny parametry jsou dosazeny z dokumentace použitého osciloskopu, která je na přiloženém DVD.

Zdrojové kódy aplikace rozdělené dle jednotlivých souborů jsou umístěny v příloze D. Stejnou aplikaci v jazyce Python jsem připravil i pro Raspbian a provedl test úplně stejným způsobem, což mi umožnilo získat data pro srovnání jednotlivých systémů.



Obr. 50: Rozvržení HW (zdroj.: Chris Briggs: Beginners guide to GPIO in Windows 10 IoT Core [online]. 2015-07-06 [cit. 2015-11-22]. Dostupné z: http://blog.chrisbriggsy.com/Beginners-guide-to-GPIO-Win10-IoT-Core-Insider-Preview/)

4.2.4. Test přesnosti SW časovačů

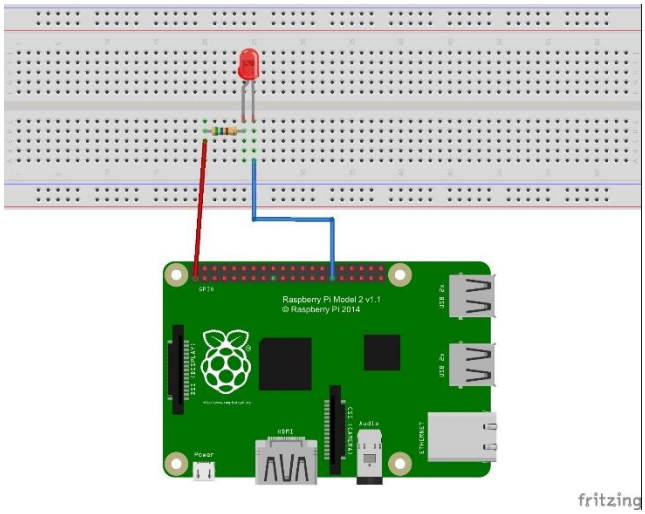
Díky nedostatku HW zdrojů hodinových signálů musí být často na většině zařízení typu Raspberry Pi2, která podporují Windows 10 IoT Core, doplněny tyto vývody o pin GPIO, který je naprogramovaný jako zdroj hodinového signálu. O řízení tohoto pinu se starají SW časovače, které jsou součástí aplikace. Nejednoduší aplikací, jejímž výkonným jádrem je časovač je, Blinky neboli HW Hallo World. Na této aplikaci, právě pro její malé nároky, jsem se rozhodl otestovat, jak přesný je hodinový signál, který řídí LED.

Celý test je opět jednoduchý, provedl jsem 4 sady měření po 5 hodnotách. K napájecímu kontaktu LED jsem připojil kanál osciloskopu. Nyní jsem (protože jde o symetrický signál) sledoval délku pulsu a následně délku rozestupu mezi pulsy v časové ose. V každé sadě jsem měřil jen délku, nebo rozestup, tedy celkově jsem změřil dva páry hodnot. Následně za každou sadu jsem spočítal průměrnou hodnotu a průměrnou odchylku od nastavené hodnoty. Ke každé hodnotě jsem stanovil příslušnou chybu měření. Všechny měřené hodnoty jsem postupně zapisoval do tabulky, kde jsem již měl předem připraveny

hodnoty, které budou nasazeny do časovače. Aby aplikace nebyla brzděna uživatelským vstupem, který by nastavoval časovač, vygeneroval jsem několik verzí aplikace, které se od sebe liší právě nastavením časovače.

Výsledkem je odchylka mezi nastavenou hodnotou a hodnotou poskytovanou pinem v jednotlivých konfiguracích a také trend této odchylky.

Chybu měření jsem, po konzultaci s panem Ing. Martinem Šimůnkem z katedry měření elektrotechnické fakulty ČVUT, určil dle vztahu (1). (zdrojový kód je v příloze E)



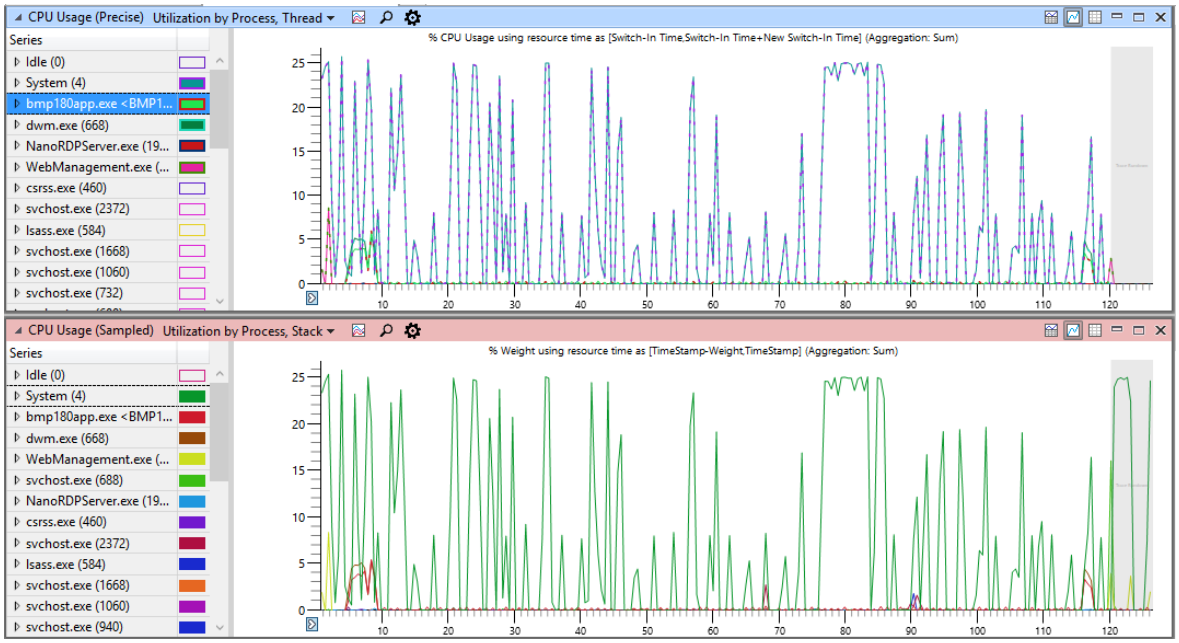
Obr. 51: Rozvržení HW (zdroj: Chris Briggs: Blinky Sample [online]. [cit. 2015-11-22]. Dostupné z: https://ms-iot.github.io/content/en-US/win10/samples/Blinky.htm)

4.3. Výsledky testů

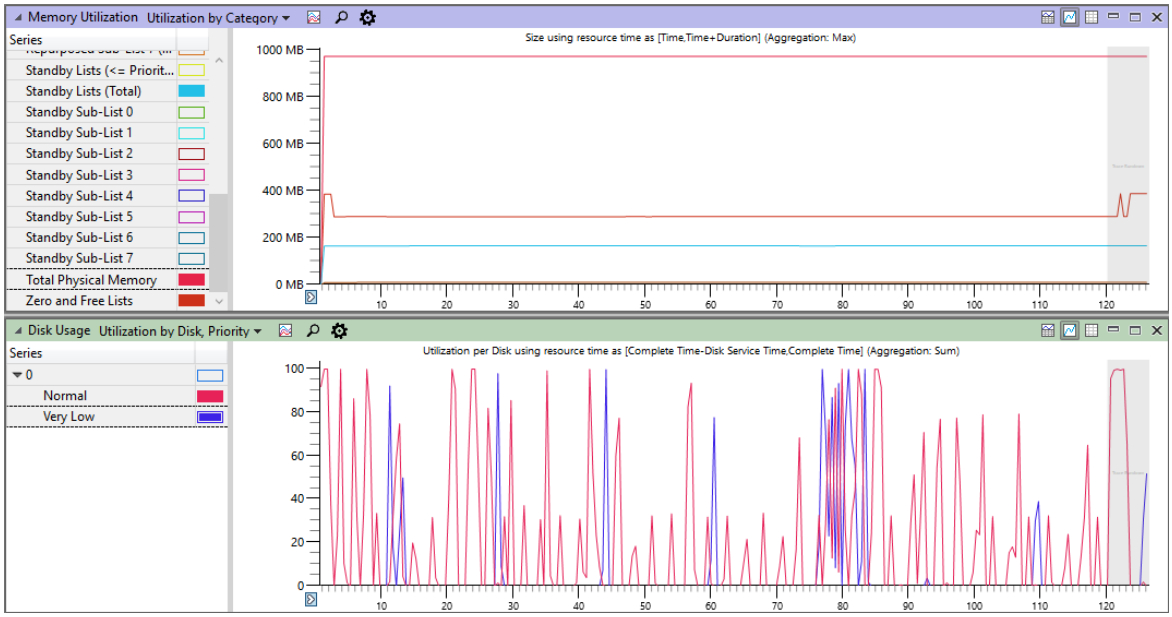
4.3.1. Test aplikace IoT scénářů

4.3.1.1. Teploměr

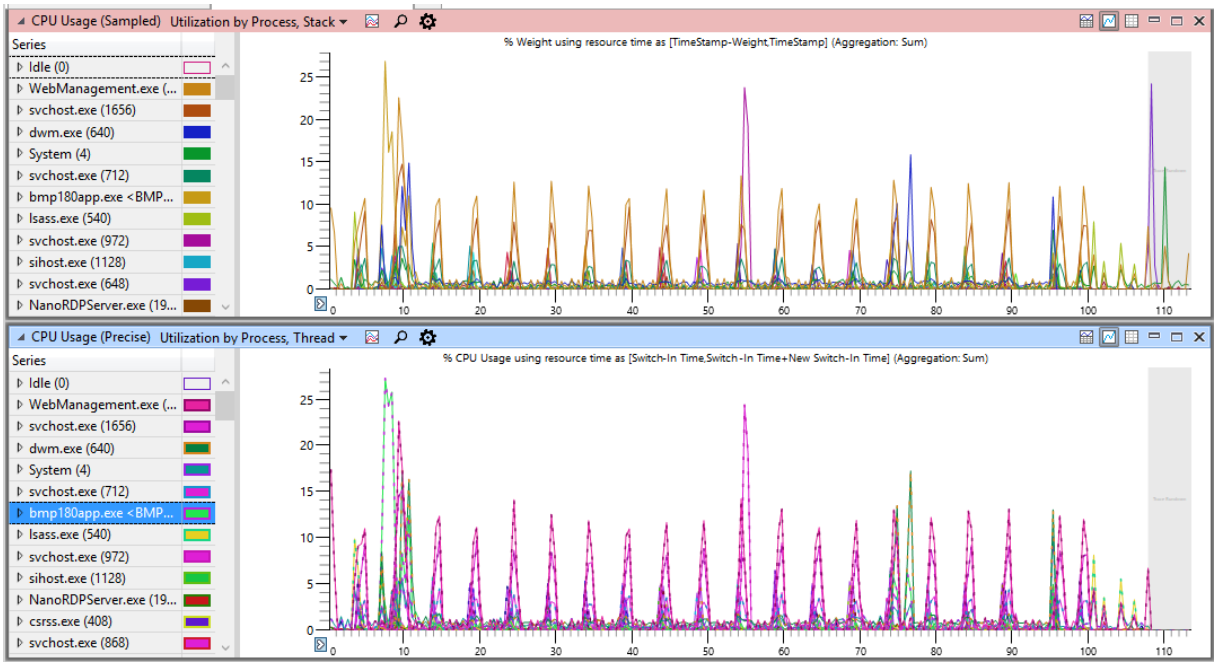
Aplikace je založena na senzoru BMP180 (datasheet je přiložen na DVD). Aplikace běží stabilně a zcela plynule. Neprojevují se žádné problémy s pomalými reakcemi, na zahřívání i ochlazování senzoru, všechny reakce na změnu teploty jsou zcela plynulé.



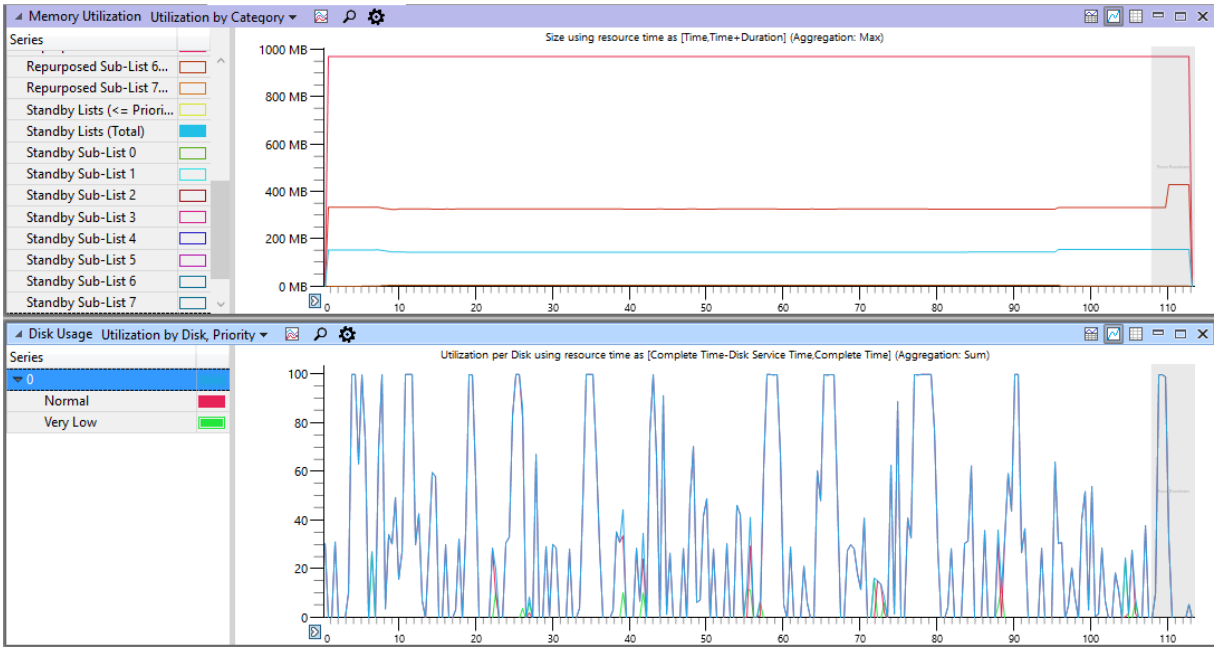
Obr. 52: CPU trace z běhu aplikace teploměr na Pi3



Obr. 53: Memory a Disk trace z běhu aplikace Teploměr na Pi3



Obr. 54: CPU trace z běhu aplikace Teploměr na Pi2



Obr. 55: Memory a Disk trace z běhu aplikace Teploměr na Pi2

Pokud se na provedení úlohy podíváme z pohledu spotřeby systémových prostředků, najdeme drobné rozdíly mezi jednotlivými deskami. Pi2 má mírně větší spotřebu fyzické paměti RAM, ale pokud jde o stránkovanou pohotovostní paměť, jsou na tom obě zařízení shodně. Dalších rozdílů si můžeme všimnout při pohledu na práci s diskem (paměťovou kartou), kde je vidět, že u Pi3 to potřebuje ještě hodně vylepšit, protože zde máme výrazně více přístupů k jednotce označených jako velmi pomalé, a to navzdory tomu, že obě desky

disponovaly pro běh aplikace kartou Kingston standartu U1. Tento rozdíl může mít vliv na plynulost běhu některých řešení, která hodně čtou či ukládají data právě na paměťovou kartu.

Pokud se podíváme na CPU stack, je vidět již znatelných rozdílů mezi deskami. Pi2 má výrazně větší spotřebu paměti zásobníku pro běh řídícího webserveru oproti Pi3, kde jednoznačně dominují základní systémové procesy. Spotřeba procesorového zásobníku přímo aplikací je naprosto minimální. Pokud se podíváme na spotřebu výpočetního času procesoru, pak na Pi2 po většinu sledované doby dominuje zpracování webserveru, zatím co na Pi3 je již jednoznačně dominantní běžící aplikace.

Na výše zmíněných rozdílech je vidět, že ačkoliv jsou systémy shodného buildu a navenek se aplikace chová shodně na obou deskách, jsou mezi systémy Windows 10 IoT Core pro Pi2 a Pi3 určité niance, o kterých si troufám říct, že nejsou dány jen rozdílnou architekturou obou desek, a tudíž jinými ovladači. Pozorování chování CPU stáčku je ve shodě s pozorováním chování při práci s webovým prohlížečem. Aplikace, které jsou náročné hodně využívají stack CPU poběží stabilněji na Pi3, protože jim zde systém nechává více prostoru, na druhou stranu pohled na využití výpočetního času ani serializace zpracování jednotlivých úloh na CPU (která je velmi zdařilá u obou desek a je shodná) nevysvětluje rozdíly v rychlosti běhu některých aplikací, které mluví ve prospěch slabší Pi2, nicméně na této konkrétní úloze pracující s I^2C sběrnicí to nebylo znatelné.

4.3.1.2. Infostánek

Aplikace informačního stánku, která je upravenou verzí webového prohlížeče, využívá tedy on-line aplikaci, která je dostupná na webu. Na Pi2 byla aplikace funkční, ale velmi pomalá a troufám si říct, že pro zákazníka neatraktivní, navzdory tomu, že byla stabilní. Nevyvolala ani zvýšené využití systémových prostředků ze strany OS, ale bylo znatelně cítit, že jde o kód, který není optimalizovaný a sám o sobě má velkou spotřebu systémových prostředků.

Naproti tomu, pokud aplikace běžela na Pi3, vyvolávala výrazně zvýšenou spotřebu prostředků systémem, pro což nemám žádné logické vysvětlení. Nešlo ani o jednorázový jev, šlo jen o opakovaný, který se projevil i po reinstalaci systému. Z pohledu uživatele byla aplikace stabilní a poměrně živá, zde bych si dokázal představit i její reálné použití.

4.3.2. Test práce s internetem

Po pravidelném používání aplikace na Pi2 i desktopu musím konstatovat, že pády aplikace, ke kterým občas na Pi2 došlo, provází i desktopovou verzi programu. Jediný

problém, na který jsem narazil ve verzi pro Pi2 bylo, že někdy dojde k pádu aplikace při práci s OneDrive z outlook.com a parametrizovaném vyhledávání na Youtube, což se na Pi3 neděje a na desktopu také ne. Pro všechny 3 verze aplikace platí, že Office online je dostupný pouze v režimu pro čtení dokumentu, nikoliv pro jeho editaci či tvorbu. Toto omezení Office online souvisí s konkrétním užitým zdrojovým kódem, nikoliv s OS, na kterém běží.

Pokud jde o procházení internetu, jsem Pi2 používal především jako hudební a video přehrávač Youtube a následně pro čtení článků. S Pi3 jsem pracoval výrazně kratší dobu, navzdory opravdu markantnímu rozdílu v rychlosti prohlížení webu je aplikace na Pi3 znatelně stabilnější, hodně je to znát například na PowerBI portálu, kde se velmi spolehlivě dají dělat editace a rychleji aplikuje dynamické filtry proti Pi2.

Srovnání průměrných časů, za které se na jednotlivých platformách načítaly vybrané webové stránky (bulvár je tam z důvodu náročnosti stránky), je vidět v následující tabulce.

Stránka / služba	Čas načtení stránky či služby [s]		
	Pi2	Pi3	X64
http://jcu.cz	2,6 ± 0,2	6,7 ± 0,2	1,8 ± 0,2
http://google.cz	1,1 ± 0,2	1,3 ± 0,2	1,0 ± 0,2
Vyhledání pojmu „IoT“ pomocí Google	1,8 ± 0,2	4,0 ± 0,2	1,6 ± 0,2
Seznam.cz	11,5 ± 0,2	31,1 ± 0,2	6,5 ± 0,2
http://youtube.com	8,1 ± 0,2	22,5 ± 0,2	8,7 ± 0,2
Spuštění přehrávání vybraného videa na youtube	14,9 ± 0,2	31,8 ± 0,2	4,5 ± 0,2
Stram.cz	10,9 ± 0,2	11,7 ± 0,2	3,2 ± 0,2
Načtení epizody Mistrů volantu na Stream.cz	14,7 ± 0,2	15,6 ± 0,2	2,0 ± 0,2
Office portál	3,2 ± 0,2	6,4 ± 0,2	2,0 ± 0,2
Word online (otevření 22 stran dokument ve formátu docx)	15,0 ± 0,2	16,6 ± 0,2	4,9 ± 0,2
PowerPoint online (otevření prezentace 18 snímků)	14,3 ± 0,2	11,5 ± 0,2	7,2 ± 0,2
SharePoint online (JCU)	17,4 ± 0,2	21,2 ± 0,2	
OneDrive	33,8 ± 0,2	43,0 ± 0,2	9,6 ± 0,2
https://www.wolframalpha.com/	(2,2 / 12,2*) ± 0,2	(6,7 / 14,2*) ± 0,2	(- / 6,4*) ± 0,2
Výpočet výrazu: sin(sqrt(X^2+Y^2))/(sqrt(X^2+Y^2)) pomocí služby Wolfram Alpha	20,8 ± 0,2	24,8 ± 0,2	8,8 ± 0,2
PowerBI portál	(6,1 / 39,6**) ± 0,2	(2,6 / 53,3**) ± 0,2	1,6 ± 0,2 / -
Živě.cz	19,3 ± 0,2	15,6 ± 0,2	7,9 ± 0,2
Blesk.cz	17,4 ± 0,2	12,9 ± 0,2	10,9 ± 0,2

Tab. 4: Časy načítání webových služeb a stránek

Časy v tabulce označené * jsou zapsány ve formátu *mobilní / plná verze webové stránky* a časy označené ** jsou zapsány ve formátu *načtení webu / přihlášení a zobrazení Dashboardu*. Časy uvedené v tabulce jsou průměrem za více pokusů o načtení webové stránky či služby. Chyba měření uvedená v tabulce je standardní chyba pro použití ručních stopek a souvisí s lidskou reakční dobou.

Jako platforma X64 byl použit Windows Server 2016 TP5 na zařízení Toshiba Satellite L750 1L8 ve výrobní konfiguraci. Důvodem bylo, že na tomto systému byl prohlížeč používán nejčastěji, protože na rozdíl od výchozího IE11 dokázal přehrát youtube a stream. Nevýhodou takto primitivní aplikace je, že dochází k automatickému přesměrování určitých odkazů do výchozího prohlížeče, a to platí především pro služby outlook.com (nelze přejít z pošty na OneDrive), PowerBI a SharePoint.

4.3.3. Test reakční doby na akci GPIO

Číslo měření	Odezva [ms]
1	8,0 ± 0,2
2	5,2 ± 0,1
3	26,0 ± 0,7
4	7,0 ± 0,2
5	20,0 ± 0,5
6	42,0 ± 1,0
7	18,0 ± 0,5
8	1,0 ± 0,02
9	4,6 ± 0,1
10	24,0 ± 0,6
11	14,8 ± 0,4
12	19,6 ± 0,5
13	38,0 ± 0,5
14	9,0 ± 0,2
15	55,6 ± 1,3
16	19,6 ± 0,5
17	12,2 ± 0,3
18	28,0 ± 0,7
19	11,4 ± 0,3
20	16,4 ± 0,4
Průměrná odezva	**19,0 ± 0, 5**

Tabulka 5: Naměřené časy odezvy pro Pi3

Číslo měření	Odezva [ms]
1	9,0 ± 0,3
2	10,4 ± 0,3
3	5,8 ± 0,2
4	13,6 ± 0,3
5	47,0 ± 1,2
6	7,4 ± 0,2
7	18,0 ± 0,5
8	22,8 ± 0,6
9	17,8 ± 0,5
10	42,0 ± 1,0
11	30,0 ± 0,8
12	31,6 ± 0,8
13	27,2 ± 0,7
14	44,0 ± 1,1
15	29,6 ± 0,7
16	22,8 ± 0,6
17	9,8 ± 0,3
18	24,4 ± 0,6
19	26,0 ± 0,7
20	38,0 ± 0,9
Průměrná odezva	**23,9** ± 0,6

Tabulka 6: Naměřené časy odezvy pro Pi2

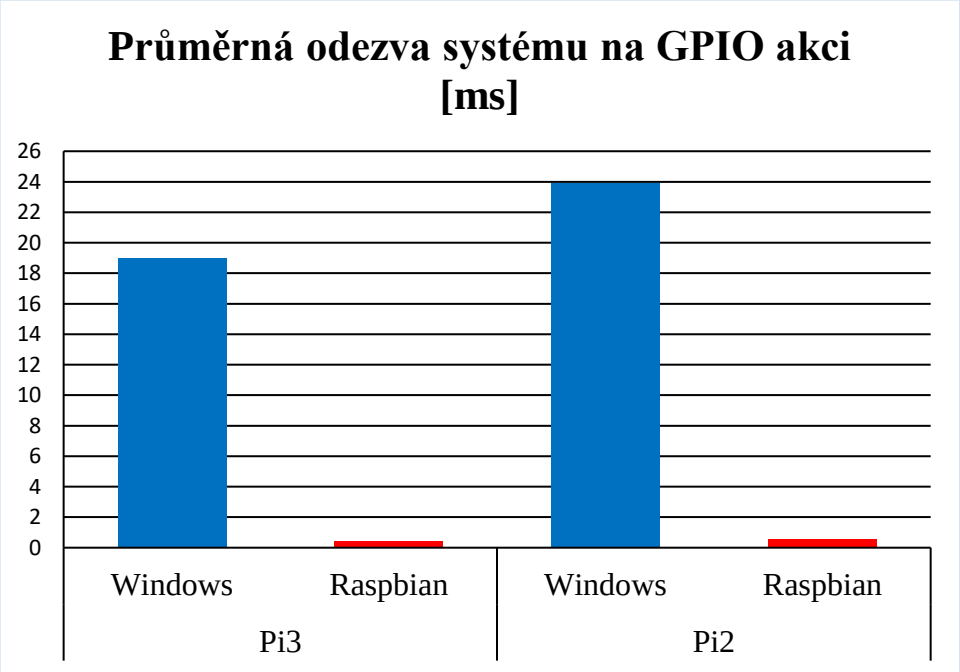
Číslo měření	Odezva [µs]
1	520 ± 13
2	420 ± 11
3	410 ± 10
4	410 ± 10
5	400 ± 10
6	380 ± 90
7	420 ± 10
8	420 ± 10
9	420 ± 10
10	410 ± 10
11	420 ± 10
12	420 ± 10
13	410 ± 10
14	410 ± 10
15	430 ± 11
16	410 ± 10
17	420 ± 11
18	410 ± 10
19	410 ± 10
20	410 ± 10
Průměrná odezva	$\mathbf{418 \pm 11}$

Tabulka 7: Naměřené časy odezvy pro Pi3 s Raspbianem

Číslo měření	Odezva [µs]
1	540 ± 13
2	560 ± 14
3	500 ± 13
4	470 ± 12
5	540 ± 14
6	550 ± 14
7	590 ± 15
8	500 ± 13
9	560 ± 14
10	620 ± 16
11	510 ± 13
12	510 ± 13
13	400 ± 10
14	510 ± 13
15	510 ± 13
16	430 ± 10
17	530 ± 13
18	660 ± 17
19	520 ± 13
20	500 ± 13
Průměrná odezva	$\mathbf{526 \pm 13}$

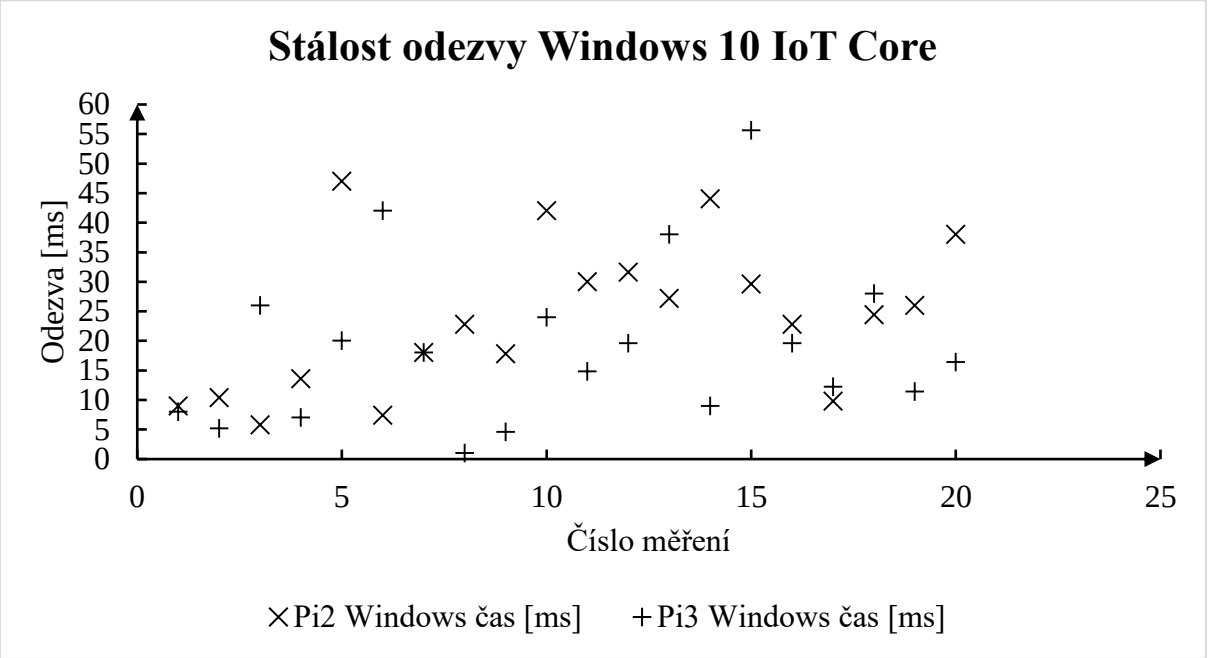
Tabulka 8: Naměřené časy odezvy pro Pi2 s Raspbianem

Z výše uvedených tabulek je jednoznačně patrné, že Raspbian má řádově lepší odezvu nežli Windows. Ale na druhou stranu musím být trochu objektivní, signál, který jsem dostal z Raspbianu, měl mnohem více překmitů a celkově nižší kvalitu ve srovnání s Windows, které vracely učebnicový signál. Bohužel na fotografiích to není příliš patrné z důvodu, že zákmity překryl časový kursor. Ve Windows aplikace běžela v grafickém režimu, jak je požadované pro Universall Apps, naproti tomu Raspbian celý běžel bez grafické nadstavby, což v kombinaci s kvalitním interpretem Pythonu přispělo k velmi rychlým reakcím systému. Vzhledem k rychlému rozvoji Windows 10 IoT Core můžeme předpokládat do budoucna zlepšení. Mnohem lepší pohled na získaná data nám dává následující graf, který shrnuje průměrné výsledky ze všech 4 měření a umožňuje tak velmi přehledné porovnání výsledků.

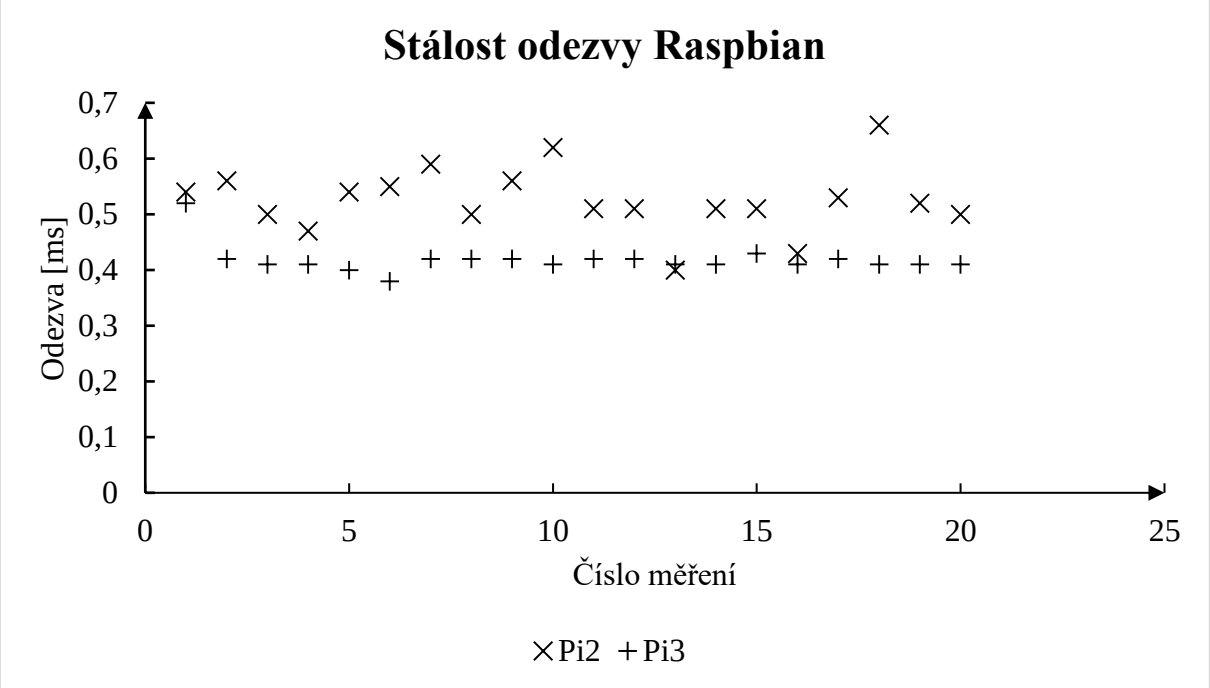


Graf č. 1: Srovnání průměrné rychlosti odezvy Windows a Raspbianu na Pi2 a Pi3

Dalším faktorem, který mluví proti nasazení Windows tam, kde je potřeba rychlá odezva na GPIO akci, je nestálost odezvy, která je výrazně větší, nežli je tomu u Raspbianu. Důvod této nestálosti odezvy neznám, protože jsem nerozebíral Windows 10 SDK, respektive Windows IoT Extensions for the UWP. Nejlépe je srovnání obou systémů patrné opět na grafech, které ovšem tentokrát nelze (především v případě Windows) proložit nějakou křivkou. Očekávanému průběhu (rovnoběžka s osou x) se nejlépe blíží Raspbian na Pi3.



Graf č. 2: Stálost odezvy Windows 10 IoT Core



Graf č. 3: Stabilita odezvy Raspbian

4.3.4. Test přesnosti SW časovačů

Číslo měření	Naměřená hodnota [ms]	Nastavená hodnota [ms]	Rozdíl [ms]
1	$29{,}4 \pm 7{,}35 \cdot 10^{-1}$	0,1	29,3
2	$13{,}4 \pm 3{,}35 \cdot 10^{-1}$	0,1	13,3
3	$18{,}2 \pm 4{,}55 \cdot 10^{-1}$	0,1	18,1
4	$16{,}0 \pm 4{,}00 \cdot 10^{-1}$	0,1	15,9
5	$14{,}6 \pm 3{,}65 \cdot 10^{-1}$	0,1	14,5
Průměr	$\mathbf{18{,}3 \pm 4{,}58 \cdot 10^{-1}}$	**0,1**	**18,2**
6	$31{,}6 \pm 7{,}90 \cdot 10^{-1}$	1	30,6
7	$37{,}6 \pm 9{,}40 \cdot 10^{-1}$	1	36,6
8	$37{,}2 \pm 9{,}30 \cdot 10^{-1}$	1	36,2
9	$31{,}2 \pm 7{,}80 \cdot 10^{-1}$	1	30,2
10	$31{,}6 \pm 7{,}90 \cdot 10^{-1}$	1	30,6
Průměr	$\mathbf{33{,}8 \pm 8{,}45 \cdot 10^{-1}}$	**1**	**32,8**
11	$33{,}4 \pm 8{,}35 \cdot 10^{-1}$	10	23,4
12	$31{,}4 \pm 7{,}85 \cdot 10^{-1}$	10	21,4
13	$31{,}8 \pm 7{,}95 \cdot 10^{-1}$	10	21,8
14	$25{,}2 \pm 6{,}30 \cdot 10^{-1}$	10	15,2
15	$31{,}4 \pm 7{,}85 \cdot 10^{-1}$	10	21,4
Průměr	$\mathbf{30{,}6 \pm 7{,}65 \cdot 10^{-1}}$	**10**	20,6
16	$536{,}0 \pm 1{,}34 \cdot 10$	500	36,0
17	$508{,}0 \pm 1{,}27 \cdot 10$	500	8,0
18	$516{,}0 \pm 1{,}29 \cdot 10$	500	16,0
19	$528{,}0 \pm 1{,}32 \cdot 10$	500	28,0
20	$532{,}0 \pm 1{,}33 \cdot 10$	500	32,0
Průměr	$\mathbf{524{,}0 \pm 1{,}31 \cdot 10}$	**500**	24,0
Číslo měření	**Naměřená hodnota [ms]**	**Nastavená hodnota [ms]**	**Rozdíl [ms]**
21	$784{,}0 \pm 1{,}96 \cdot 10$	750	34,0
22	$768{,}0 \pm 1{,}92 \cdot 10$	750	18,0
23	$788{,}0 \pm 1{,}97 \cdot 10$	750	38,0
24	$780{,}0 \pm 1{,}95 \cdot 10$	750	30,0
25	$780{,}0 \pm 1{,}95 \cdot 10$	750	30,0
Průměr	$\mathbf{780{,}0 \pm 1{,}95 \cdot 10}$	**750**	30,0

Tabulka 9: Naměřené hodnoty přesnosti časovače Pi2

Číslo měření	Naměřená hodnota [ms]	Nastavená hodnota [ms]	Rozdíl [ms]
1	$15{,}2 \pm 3{,}80 \cdot 10^{-1}$	0,1	15,1
2	$16{,}0 \pm 4{,}00 \cdot 10^{-1}$	0,1	15,9
3	$15{,}6 \pm 3{,}90 \cdot 10^{-1}$	0,1	15,5
4	$15{,}4 \pm 3{,}85 \cdot 10^{-1}$	0,1	15,3
5	$15{,}4 \pm 3{,}85 \cdot 10^{-1}$	0,1	15,3
Průměr	$\mathbf{15{,}5 \pm 3{,}88 \cdot 10^{-1}}$	**0,1**	15,4
6	$34{,}0 \pm 8{,}50 \cdot 10^{-1}$	1	33,0
7	$32{,}0 \pm 8{,}00 \cdot 10^{-1}$	1	31,0
8	$28{,}6 \pm 7{,}15 \cdot 10^{-1}$	1	27,6
9	$27{,}6 \pm 6{,}90 \cdot 10^{-1}$	1	26,6
10	$31{,}0 \pm 7{,}75 \cdot 10^{-1}$	1	30,0
Průměr	$\mathbf{30{,}6 \pm 7{,}65 \cdot 10^{-1}}$	**1**	29,6
11	$35{,}6 \pm 8{,}90 \cdot 10^{-1}$	10	25,6
12	$31{,}6 \pm 7{,}90 \cdot 10^{-1}$	10	21,6
13	$31{,}8 \pm 7{,}95 \cdot 10^{-1}$	10	21,8
14	$27{,}0 \pm 6{,}75 \cdot 10^{-1}$	10	17,0
15	$27{,}4 \pm 6{,}85 \cdot 10^{-1}$	10	17,4
Průměr	$\mathbf{30{,}7 \pm 7{,}68 \cdot 10^{-1}}$	**10**	20,7
Číslo měření	**Naměřená hodnota [ms]**	**Nastavená hodnota [ms]**	**Rozdíl [ms]**
16	$524{,}0 \pm 1{,}31 \cdot 10$	500	24,0
17	$512{,}0 \pm 1{,}28 \cdot 10$	500	12,0
18	$518{,}0 \pm 1{,}30 \cdot 10$	500	18,0
19	$518{,}0 \pm 1{,}30 \cdot 10$	500	18,0
20	$536{,}0 \pm 1{,}34 \cdot 10$	500	36,0
Průměr	$\mathbf{521{,}6 \pm 1{,}30 \cdot 10}$	**500**	21,6
21	$788{,}0 \pm 1{,}97 \cdot 10$	750	38,0
22	$780{,}0 \pm 1{,}95 \cdot 10$	750	30,0
23	$772{,}0 \pm 1{,}93 \cdot 10$	750	22,0
24	$772{,}0 \pm 1{,}93 \cdot 10$	750	22,0
25	$780{,}0 \pm 1{,}95 \cdot 10$	750	30,0
Průměr	$\mathbf{778{,}4 \pm 1{,}95 \cdot 10}$	**750**	28,4

Tabulka 10: Naměřené hodnoty přesnosti časovače Pi3

Jak je patrné z hodnot v tabulkách a také z fotografií znázorňujících průběh výstupního signálu, Pi3 poskytuje celkově kvalitnější výstup. I tak je na první pohled patrné, že časovač zdaleka nedosahuje přesnosti ani stability potřebné pro další řízení HW. Jeho použití je bez problémů možné pro vytvoření jednoduchých stopek, nebo řízení osvětlení, či LED „hada“ nebo pro tónové generátory ve slyšitelném spektru bez požadavků na přesnost tónu za účelem ladění hudebních nástrojů.

Vzhledem k nedostatku HW časovačů na deskách Pi2 a Pi3 lze časovače pro řadu praktických úloh (např.: načítání dat ze senzorů v řádově minutových intervalech) zcela bez obav použít.

5. Výsledné hodnocení systému

Z uživatelského pohledu se systém jeví jako velice zdařilý. Na rozdíl od většiny Linuxové konkurence nedisponuje plným GUI, které je zobrazeno přímo na připojený monitor, ale disponuje kvalitním a přehledným uživatelským rozraním, které je dostupné prostřednictvím webserveru. Vzdálené ovládání pomocí webserveru je velmi přehledné, spolehlivé a jednoduché, mnohým lidem může připomínat portál Reporting Services z SQL serveru 2016, nebo již o něco méně podobu přístupu pomocí webového prohlížeče k operačnímu systému Windows Server 2012R2 Essentials. I instalace, především novějších buildů, je též uživatelsky velmi přívětivá, intuitivní a maximálním možným způsobem automatizovaná. Zde je na místě pochválit Microsoft za inspiraci nástroji pro instalaci mobilní verze Windows. Vzhledem ke specializaci systému si myslím, že je uživatelsky velmi přívětivý a zdařilý, jen by se dala vytknout jedna věc a to ta, že správa uživatelských účtů, jde udělat jen pomocí PowerShellu, nikoli graficky. Dle mého názoru však IoT OS nepotřebuje více jak jeden uživatelský účet.

Systém není určen ani jako odlehčená náhrada desktopových Windows, jak je to možné vídat u jeho linuxových konkurentů, ale i s ohledem na velmi rychlý vývoj se systém jeví jako dobrá volba pro většinu projektů domácí automatizace, či jinou kutilskou činnost. Edice Core se také ukázala být dobrou alternativou pro řízení informačních stánků, či tabulí, kde dokáží provozovat v individuálně upraveném prohlížeči příslušnou webovou aplikaci, nebo využít původní zdrojový balíček aplikace určené do Windows Store, což podstatně zjednodušuje údržbu i aktualizace SW vybavení takovýchto míst. V tomto směru testy prokázaly tvrzení Microsoftu o tom, k čemu je vhodná edice Core.

Systém je poměrně hodně háklivý na rychlost microSD karty (má zásadní vliv na stabilitu systému), stálost napájení a postup vypínání systému (buď pomocí příkazu, pomocí webserveru, nebo pomocí výchozí aplikace systému). Při předčasném odpojení systému od napájení dochází především u Insider buildů k problémům s následujícím startem, který nezřídka končí „modrou smrtí“ a někdy se stane, že se problém nepodaří systému vyřešit a je nutné jej znovu instalovat. Systém, stejně jako celý HW, není vhodný pro běh s bateriovým napájením.

Z provedených testů je dobře patrné, že stabilita a rychlost celého řešení je hodně závislá na kvalitě aplikace, ale také na HW, na kterém běží. Mezi počítači Pi2 a Pi3 jsem pozoroval rozdíly v chování systému, které nebyly vždy předpokládatelné, nebo logické. Velkým zklamáním bylo chování GPIO sběrnice, jejíž reakce je výrazně pomalejší, nežli

pokud je na zařízení instalován Raspbian, což je v rozporu se snahou Microsoftu. Pomalé reakce GPIO sběrnice se mně osobně jeví, jako jeden ze silných omezujících faktorů, který limituje scénáře, kde lze systém využít. S GPIO souvisí také SW časovače, které byly též zklamáním, protože rozumně akceptovatelnou odchylku v kladném směru od požadované hodnoty začíná dosahovat až pro časy kolem ¾ s. Pro kutilské aplikace, kde stačí data z nějakého snímače načíst jednou za sekundu, nebo i po delších časových intervalech nejde o problém, překážkou se to stává, pokud pomocí SW časovače mají synchronizovat nějaká zařízení. Zde se dle mého názoru Microsoftu nepodařilo vyřešit programovou cestou problém s nedostatkem HW časovačů na většině zařízení, pro která je systém určen.

Co se Microsoftu povedlo opravdu splnit je, že aplikace jsou bez nutnosti dopisu jediné čárky kódu přenositelné napříč všemi zařízeními, která používají Windows 10, nebo Windows Server 2016[10]. Jediné, na co je potřeba si dát pozor je, pokud aplikace využívá specializované sběrnice (GPIO, I^2C…), aby zařízení, na kterém je aplikace spouštěna tyto sběrnice opravdu fyzicky mělo, což limituje přenositelnost kódů určených IoT na zbytek platformy (musí se na to myslet především při prvních pokusech ladění kódu), nicméně opačně to neplatí, takže pokud si napíši například multimediální centrum, bude mi fungovat jak na IoT zařízeních (pokud bude dostačovat paměť), tak na telefonu, tabletu i počítači. Jako velké plus bych to viděl zejména pro vývojáře, kteří mohou na tomto systému testovat aplikace bez toho, aby nějak poškodili svůj vlastní OS skrytými vadami v aplikaci.

Závěrem mohu konstatovat, že Windows 10 IoT Core je ve většině případů kvalitní bezplatnou alternativou pro scénáře, které využívají Windows Embedded a navzdory relativně rané fázi vývoje jsou kvalitním konkurentem IoT distribucím Linuxu. Systém obsahuje spoustu míst, kde je potřeba odvést ještě opravdu hodně práce, ale jak jsem se mohl v průběhu vypracování své bakalářské práce přesvědčit, v Microsoftu si to uvědomují a intenzivně pracují na rozvoji tohoto systému.

[10] Za předpokladu, že systém je vybaven Desktop Experience, uživatel není built in admin a nemá zakázané UWP bezpečnostní politikou.

6. Sezam použitých obrázků

7. Seznam použité literatury

7.1. Publikace

1. **Petr Roudenský, Anna Havlíčková.** *Řízení kvality softwaru.* Brno : Computer Press, 2013. IBSN 978-80-251-3816-8.
2. VÍT, Tomáš. Raspberry Pi 2: Ještě rychlejší minipočítač. *Computer*. Brno: Mladá fronta, 2015, **2015**(4), 3.
3. *The MagPi: The official Raspberry Pi magazine*. 2016, **2016**(8).

7.2. Internetové zdroje

4. PowerShell to connect to device running Windows 10 IoT Core Core. *Hackster.io.* [Online] Microsoft, 2015. [Citace: 13. 11 2015.] https://www.hackster.io/windowsiot/powershell-to-connect-to-a-machine-running-windows-10-16a479.
5. Deploying an App with Visual Studio. *Windows Dev Center.* [Online] Microsoft. [Citace: 24. 7 2016.] https://developer.microsoft.com/en-us/windows/iot/win10/AppDeployment.htm#python.
6. Windows 10 for the Internet of Your Things. *Microsoft.* [Online] Microsoft, 2015. [Citace: 13. 11 2015.] https://www.microsoft.com/en-us/WindowsForBusiness/windows-iot
7. Porovnání nabídek nástroje Visual Studio 2015. *Visual Studio.* [Online] Micorosoft. [Citace: 29. 5 2016.] https://www.visualstudio.com/cs-cz/products/compare-visual-studio-2015-products-vs.aspx
8. Downloads and Tools. *Windows Dev Center.* [Online] Micorosoft. [Citace: 5. 11 2015.] https://developer.microsoft.com/en-us/windows/iot/Downloads.htm.
9. Soubory ke stažení pro Visual Studio. *Visual Studio.* [Online] Microsoft. [Citace: 20. 11 2015.] https://www.visualstudio.com/cs-cz/downloads/download-visual-studio-vs.aspx.
10. Windows 10 specifikace. *Microsoft.* [Online] Microsoft. [Citace: 5. 11 2015.]
11. Setup with NOOBS. *Windows Dev Center.* [Online] Microsoft, 2016. [Citace: 19. 7 2016.] https://developer.microsoft.com/cs-cz/windows/iot/win10/Noobs.htm.

8. Seznam použitých zkratek

Zkratka	Význam zkratky
UWP	Univerzální aplikace Windows 10
Obr.	Obrázek
Tab.	Tabulka
UAI	Ústav aplikované informatiky Přírodovědecké fakulty Jihočeské univerzity v Českých Budějovicích
UFY	Ústav fyziky a biofyziky Přírodovědecké fakulty Jihočeské univerzity v Českých Budějovicích
Mgr.	Magistr, akademický titul
Ph.D.	Vysokoškolský titul
Sb.	Sbírka, ve významu sbírky zákonů
odst.	Odstavec
JCU	Jihočeská univerzita v Českých Budějovicích
č.	Číslo
IoT	Internet of Things (internet věcí)
SW	Software
HW	Hardware
OS	Operační systém
ATM	Bankomat, prodejní terminál
Tzv.	Takzvaný
Např.	Například
RTM	Release To Manufacturing (produkční verze)
U1	Rychlostní standard paměťových karet
Wi-Fi nebo WiFi	Wireless Fidelity (standard bezdrátových sítí)
USB	Universal Serial Bus
GPIO	General-purpose input/output
x86	32 bitová architektura operačního systému či procesoru
x64	64 bitová architektura operačního systému či procesoru
ARM	Advanced RISC Machine
TP3	Technical Preview 3 (samotné TP je pak Technical Preview)
HDD	Pevný disk
RAM	Paměť s libovolným přístupem (vyrovnávací paměť počítače)
PC	Osobní počítač, v širším významu jen počítač
IDE	Integrované vývojové prostředí
MSDN	Microsoft Developers Network
MSDN AA	MicroSoft Developers Network for Academics alliance
DVD	Digital Video Disk
HDMI	High-Definition Multi-media Interface
SDHC	Secure digital High capacity
Apps	Aplikace moderního uživatelského rozhraní Windows 8, 8.1, Server 2012 a Server 2012R2, pokud předchází slovo Universal, jedná se o aplikace pro rodinu systémů Windows 10 a Windows Server 2016
IP	Internet Protocol
MAC	Media Access Control
SSH	Secure Shell

Zkratka	Význam zkratky
NT	Typ jádra OS Windows (New Technology)
NOOBS	New Out Of the Box Software
CPU	Central Procesor Unit
LED	Dioda emitující světlo na principu spontánní emise elektronů

9. Seznam příloh

A. Příkazy vzdálené relace Windows 10 IoT Core začínající klíčovým slovem get
B. DVD s elektronickou podobou práce, obrazovým materiálem a elektronickými přílohami
C. Popis pinů Raspberry Pi 2
D. Zdrojový kód testování reakční doby
E. Zdrojový kód testu přesnosti SW časovačů
F. Zdrojový kód informačního stánku
G. Zdrojový kód webového prohlížeče
H. Ukázkový zdrojový kód v Pythonu
I. Zdrojový kód měření reakční doby GPIO sběrnice pro Raspbian
J. Obrazová dokumentace měření na GPIO sběrnici
K. Obrázky novinek ve webovém rozhraní pro správu OS
L. Obrázky testu práce s internetem
M. Postup stažení systému pro Pi3
N. Grafy vytížení systému při simulaci informačního stánku
O. Práce s FTP

Příloha A: Příkazy vzdálené relace Windows 10 IoT začínající klíčovým slovem get

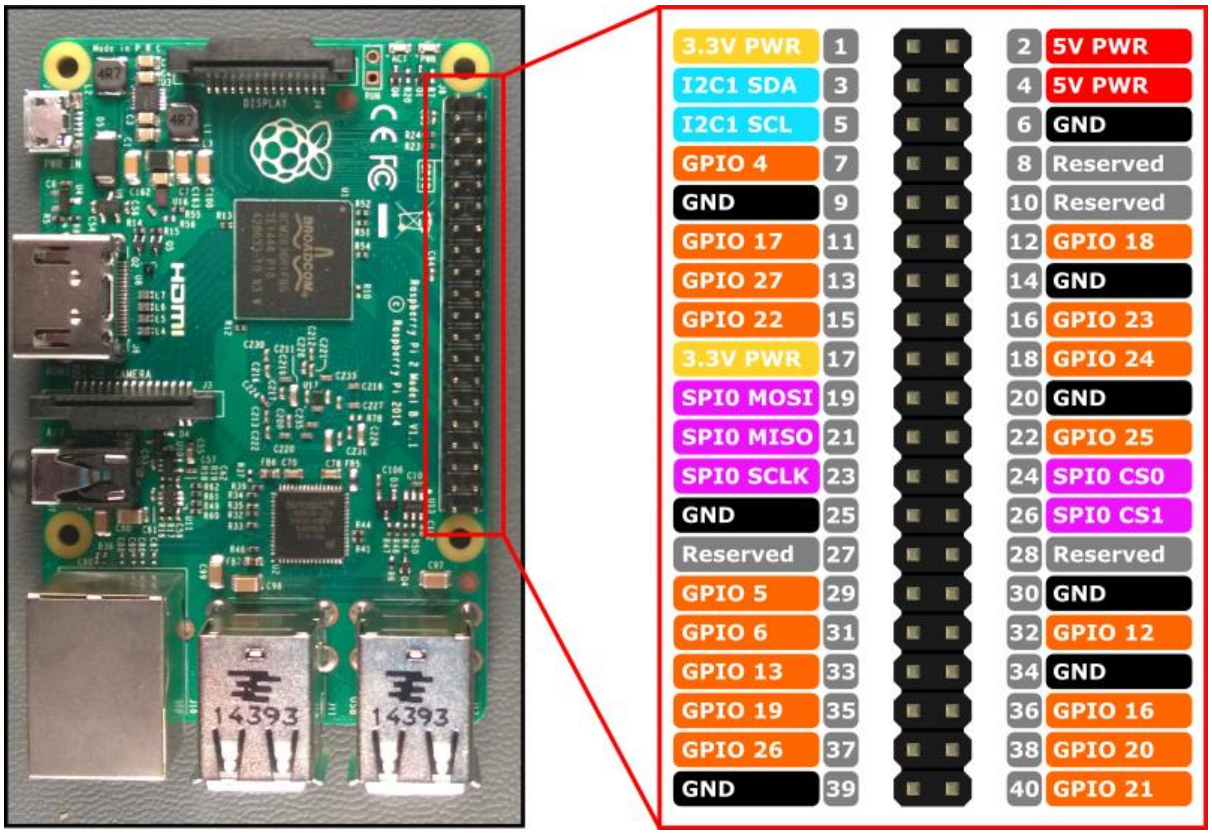
Name	*Category*	*Module*	*Synopsis*
Get-Verb	Function		
Get-FileHash	Function	Microsoft.PowerShell.U...	...
Get-Command	Cmdlet	Microsoft.PowerShell.Core	...
Get-Help	Cmdlet	Microsoft.PowerShell.Core	...
Get-History	Cmdlet	Microsoft.PowerShell.Core	...
Get-Job	Cmdlet	Microsoft.PowerShell.Core	...
Get-Module	Cmdlet	Microsoft.PowerShell.Core	...
Get-PSHostProcessInfo	Cmdlet	Microsoft.PowerShell.Core	...
Get-PSSession	Cmdlet	Microsoft.PowerShell.Core	...
Get-PSSessionCapability	Cmdlet	Microsoft.PowerShell.Core	...
Get-PSSessionConfiguration	Cmdlet	Microsoft.PowerShell.Core	...
Get-Alias	Cmdlet	Microsoft.PowerShell.U...	...
Get-Culture	Cmdlet	Microsoft.PowerShell.U...	...
Get-Date	Cmdlet	Microsoft.PowerShell.U...	...
Get-Event	Cmdlet	Microsoft.PowerShell.U...	...
Get-EventSubscriber	Cmdlet	Microsoft.PowerShell.U...	...
Get-FormatData	Cmdlet	Microsoft.PowerShell.U...	...
Get-Host	Cmdlet	Microsoft.PowerShell.U...	...
Get-Member	Cmdlet	Microsoft.PowerShell.U...	...
Get-PSBreakpoint	Cmdlet	Microsoft.PowerShell.U...	...
Get-PSCallStack	Cmdlet	Microsoft.PowerShell.U...	...
Get-Random	Cmdlet	Microsoft.PowerShell.U...	...
Get-Runspace	Cmdlet	Microsoft.PowerShell.U...	...
Get-RunspaceDebug	Cmdlet	Microsoft.PowerShell.U...	...
Get-TypeData	Cmdlet	Microsoft.PowerShell.U...	...
Get-UICulture	Cmdlet	Microsoft.PowerShell.U...	...
Get-Unique	Cmdlet	Microsoft.PowerShell.U...	...
Get-Variable	Cmdlet	Microsoft.PowerShell.U...	...
Get-ControlPanelItem	Cmdlet	Microsoft.PowerShell.M...	Get-ControlPanelItem...
Get-WmiObject	Cmdlet	Microsoft.PowerShell.M...	Get-WmiObject...
Get-Location	Cmdlet	Microsoft.PowerShell.M...	Get-Location...
Get-ComputerInfo	Cmdlet	Microsoft.PowerShell.M...	Get-ComputerInfo...
Get-PSProvider	Cmdlet	Microsoft.PowerShell.M...	Get-PSProvider...
Get-PSDrive	Cmdlet	Microsoft.PowerShell.M...	Get-PSDrive...
Get-ChildItem	Cmdlet	Microsoft.PowerShell.M...	Get-ChildItem...
Get-Content	Cmdlet	Microsoft.PowerShell.M...	Get-Content...
Get-Process	Cmdlet	Microsoft.PowerShell.M...	Get-Process...
Get-ItemPropertyValue	Cmdlet	Microsoft.PowerShell.M...	Get-ItemPropertyValue...
Get-Service	Cmdlet	Microsoft.PowerShell.M...	Get-Service...
Get-HotFix	Cmdlet	Microsoft.PowerShell.M...	Get-HotFix...

Name	*Category*	*Module*	*Synopsis*
Get-TimeZone	Cmdlet	Microsoft.PowerShell.M...	Get-TimeZone...
Get-Clipboard	Cmdlet	Microsoft.PowerShell.M...	Get-Clipboard...
Get-EventLog	Cmdlet	Microsoft.PowerShell.M...	Get-EventLog...
Get-Transaction	Cmdlet	Microsoft.PowerShell.M...	Get-Transaction...
Get-ComputerRestorePoint	Cmdlet	Microsoft.PowerShell.M...	Get-ComputerRestorePoint...
Get-Item	Cmdlet	Microsoft.PowerShell.M...	Get-Item...
Get-ItemProperty	Cmdlet	Microsoft.PowerShell.M...	Get-ItemProperty...
Get-AppxVolume	Cmdlet	Appx	Get-AppxVolume...
Get-AppxPackageManifest	Cmdlet	Appx	Get-AppxPackageManifest...
Get-AppxLog	Function	Appx	...
Get-AppxDefaultVolume	Cmdlet	Appx	Get-AppxDefaultVolume...
Get-AppxLastError	Function	Appx	...
Get-AppxPackage	Cmdlet	Appx	Get-AppxPackage...
Get-CimInstance	Cmdlet	CimCmdlets	Get-CimInstance...
Get-CimClass	Cmdlet	CimCmdlets	Get-CimClass...
Get-CimSession	Cmdlet	CimCmdlets	Get-CimSession...
Get-CimAssociatedInstance	Cmdlet	CimCmdlets	Get-CimAssociatedInstance...
Get-DnsClientCache	Function	DnsClient	...
Get-DnsClientServerAddress	Function	DnsClient	...
Get-DnsClient	Function	DnsClient	...
Get-DnsClientGlobalSetting	Function	DnsClient	...
Get-WinEvent	Cmdlet	Microsoft.PowerShell.D...	Get-WinEvent...
Get-LocalUser	Cmdlet	Microsoft.PowerShell.L...	Get-LocalUser...
Get-LocalGroup	Cmdlet	Microsoft.PowerShell.L...	Get-LocalGroup...
Get-LocalGroupMember	Cmdlet	Microsoft.PowerShell.L...	Get-LocalGroupMember...
Get-AuthenticodeSignature	Cmdlet	Microsoft.PowerShell.S...	Get-AuthenticodeSignature...
Get-ExecutionPolicy	Cmdlet	Microsoft.PowerShell.S...	Get-ExecutionPolicy...
Get-Credential	Cmdlet	Microsoft.PowerShell.S...	Get-Credential...
Get-Acl	Cmdlet	Microsoft.PowerShell.S...	Get-Acl...
Get-WSManInstance	Cmdlet	Microsoft.WSMan.Manage...	Get-WSManInstance...
Get-WSManCredSSP	Cmdlet	Microsoft.WSMan.Manage...	Get-WSManCredSSP...
Get-NetAdapterPowerManagement	Function	NetAdapter	...
Get-NetAdapterIPsecOffload	Function	NetAdapter	...
Get-NetAdapterQos	Function	NetAdapter	...
Get-NetAdapterRdma	Function	NetAdapter	...
Get-NetAdapterPacketDirect	Function	NetAdapter	...
Get-NetAdapterChecksumOffload	Function	NetAdapter	...
Get-NetAdapterVMQQueue	Function	NetAdapter	...
Get-NetAdapterRss	Function	NetAdapter	...
Get-NetAdapterEncapsulatedPack...	Function	NetAdapter	...
Get-NetAdapterSriov	Function	NetAdapter	...
Get-NetAdapterVmq	Function	NetAdapter	...
Get-NetAdapterHardwareInfo	Function	NetAdapter	...
Get-NetAdapterVPort	Function	NetAdapter	...
Get-NetAdapterSriovVf	Function	NetAdapter	...

Name	*Category*	*Module*	*Synopsis*
Get-NetAdapterLso	Function	NetAdapter	...
Get-NetAdapterAdvancedProperty	Function	NetAdapter	...
Get-NetAdapterStatistics	Function	NetAdapter	...
Get-NetAdapterRsc	Function	NetAdapter	...
Get-NetAdapter	Function	NetAdapter	...
Get-NetAdapterBinding	Function	NetAdapter	...
Get-NetEventProvider	Function	NetEventPacketCapture	...
Get-NetEventVmSwitchProvider	Function	NetEventPacketCapture	...
Get-NetEventPacketCaptureProvider	Function	NetEventPacketCapture	...
Get-NetEventNetworkAdapter	Function	NetEventPacketCapture	...
Get-NetEventVmNetworkAdapter	Function	NetEventPacketCapture	...
Get-NetEventVmSwitch	Function	NetEventPacketCapture	...
Get-NetEventVFPProvider	Function	NetEventPacketCapture	...
Get-NetEventSession	Function	NetEventPacketCapture	...
Get-NetEventWFPCaptureProvider	Function	NetEventPacketCapture	...
Get-NetIPsecPhase2AuthSet	Function	NetSecurity	...
Get-NetFirewallPortFilter	Function	NetSecurity	...
Get-NetFirewallProfile	Function	NetSecurity	...
Get-NetIPsecMainModeSA	Function	NetSecurity	...
Get-NetFirewallInterfaceFilter	Function	NetSecurity	...
Get-NetIPsecRule	Function	NetSecurity	...
Get-NetIPsecMainModeRule	Function	NetSecurity	...
Get-NetFirewallServiceFilter	Function	NetSecurity	...
Get-DAPolicyChange	Cmdlet	NetSecurity	Get-DAPolicyChange...
Get-NetFirewallSecurityFilter	Function	NetSecurity	...
Get-NetFirewallInterfaceTypeFi...	Function	NetSecurity	...
Get-NetFirewallSetting	Function	NetSecurity	...
Get-NetFirewallRule	Function	NetSecurity	...
Get-NetFirewallApplicationFilter	Function	NetSecurity	...
Get-NetIPsecQuickModeCryptoSet	Function	NetSecurity	...
Get-NetFirewallAddressFilter	Function	NetSecurity	...
Get-NetIPsecQuickModeSA	Function	NetSecurity	...
Get-NetIPsecDospSetting	Function	NetSecurity	...
Get-NetIPsecMainModeCryptoSet	Function	NetSecurity	...
Get-NetIPsecPhase1AuthSet	Function	NetSecurity	...
Get-NetCompartment	Function	NetTCPIP	...
Get-NetOffloadGlobalSetting	Function	NetTCPIP	...
Get-NetUDPSetting	Function	NetTCPIP	...
Get-NetUDPEndpoint	Function	NetTCPIP	...
Get-NetTCPSetting	Function	NetTCPIP	...
Get-NetTransportFilter	Function	NetTCPIP	...
Get-NetIPAddress	Function	NetTCPIP	...
Get-NetIPv6Protocol	Function	NetTCPIP	...
Get-NetIPInterface	Function	NetTCPIP	...
Get-NetRoute	Function	NetTCPIP	...

Name	*Category*	*Module*	*Synopsis*
Get-NetIPConfiguration	Function	NetTCPIP	...
Get-NetIPv4Protocol	Function	NetTCPIP	...
Get-NetTCPConnection	Function	NetTCPIP	...
Get-NetNeighbor	Function	NetTCPIP	...
Get-NetPrefixPolicy	Function	NetTCPIP	...
Get-PnpDeviceProperty	Function	PnpDevice	...
Get-PnpDevice	Function	PnpDevice	...
Get-TpmSupportedFeature	Cmdlet	TrustedPlatformModule	Get-TpmSupportedFeature...
Get-TpmEndorsementKeyInfo	Cmdlet	TrustedPlatformModule	Get-TpmEndorsementKeyInfo.
Get-Tpm	Cmdlet	TrustedPlatformModule	Get-Tpm...

Příloha C: Popis pinů Raspberry Pi 2



Popis pinů Raspberry Pi 2 (Zdroj: Windows Dev Center: Blinky Sample *[online]. 2015 [cit. 2015-11-21]. Dostupné z: http://ms-iot.github.io/content/en-US/win10/samples/Blinky.htm)*

Příloha D: Zdrojový kód testování reakční doby

Soubor MainPage.xaml

```
<Page
     x:Class="GPIOIntroPi.MainPage"
     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     xmlns:local="using:GPIOIntroPi"
     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
     mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
            <TextBlock x:Name="GpioStatus" Text="Waiting to initialize GPIO..."
Margin="10,50,10,10" TextAlignment="Center" FontSize="26.667" />
        </StackPanel>
    </Grid>
</Page>
```

Zdroj: GitHubab: MainPage.xaml *[online]. 2015, 6.7.2015 [cit. 2015-11-21]. Dostupné z: https://gist.githubusercontent.com/ChrisBriggsy/82c7a1a3cd24e553ca29/raw/59ee5c27276e17e96b8e623c9afe013003ffc534/MainPage.xaml*

Soubor MainPage.xaml.cs

```
using Windows.UI.Xaml.Controls;
using Windows.Devices.Gpio;
using Windows.UI.Xaml;
using System;

namespace GPIOIntroPi
{
     public sealed partial class MainPage : Page
     {
        private const int LED_PIN = 6;
        private const int PB_PIN = 5;
        private GpioPin pin;
        private GpioPin pushButton;
        private DispatcherTimer timer;
        private GpioPinValue pushButtonValue;
        public MainPage()
         {
            InitializeComponent();
            timer = new DispatcherTimer();
            timer.Interval = TimeSpan.FromMilliseconds(1);
            timer.Tick += Timer_Tick;
            timer.Start();
            Unloaded += MainPage_Unloaded;
            InitGPIO();
         }
         private void InitGPIO()
         {
             var gpio = GpioController.GetDefault();
             if (gpio == null)
             {
```

```
                GpioStatus.Text = "Toto zařízení nemá GPIO.";
                return;
            }
            pushButton = gpio.OpenPin(PB_PIN);
            pin = gpio.OpenPin(LED_PIN);
            pushButton.SetDriveMode(GpioPinDriveMode.Input);
            pin.Write(GpioPinValue.High);
            pin.SetDriveMode(GpioPinDriveMode.Output);
            GpioStatus.Text = "GPIO je připravena.";
        }
        private void MainPage_Unloaded(object sender, object args)
        {
            pin.Dispose();
            pushButton.Dispose();
        }
        private void FlipLED()
        {
            pushButtonValue = pushButton.Read();
            if (pushButtonValue == GpioPinValue.High)
            {
                pin.Write(GpioPinValue.High);
            }
            else if (pushButtonValue == GpioPinValue.Low)
            {
                pin.Write(GpioPinValue.Low);
            }
        }
        private void Timer_Tick(object sender, object e)
        {
            FlipLED();
        }
    }
}
```

Upraveno: GitHubab: MainPage.xaml [online]. 2015, 6.7.2015 [cit. 2015-11-21]. Dostupné z: https://gist.githubusercontent.com/ChrisBriggsy/d7ca4826e0919135cd16/raw/5a71e4363aecbca430343d9edea3f6c13d464306/MainPage.xaml.cs

Soubor App.xaml.cs

```
using GPIOIntroPi;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

namespace spinani
{
    /// <summary>
```

```
    /// Poskytuje chování specifické pro aplikaci, které doplňuje výchozí třídu
Application.
    /// </summary>
    sealed partial class App : Application
    {
        /// <summary>
        /// Inicializuje objekt aplikace typu singleton. Jedná se o první řádek
spuštěného vytvořeného kódu,
        /// který je proto logickým ekvivalentem metod main() nebo WinMain().
        /// </summary>
        public App()
        {
            Microsoft.ApplicationInsights.WindowsAppInitializer.InitializeAsync(
                Microsoft.ApplicationInsights.WindowsCollectors.Metadata |
                Microsoft.ApplicationInsights.WindowsCollectors.Session);
            this.InitializeComponent();
            this.Suspending += OnSuspending;
        }

        /// <summary>
        /// Vyvolá se při normálním spuštění aplikace koncovým uživatelem. Ostatní
vstupní body
        /// se použijí například při spuštění aplikace za účelem otevření
konkrétního souboru.
        /// </summary>
        /// <param name="e">Podrobnosti o žádosti o spuštění a procesu</param>
        protected override void OnLaunched(LaunchActivatedEventArgs e)
        {

#if DEBUG
            if (System.Diagnostics.Debugger.IsAttached)
            {
                this.DebugSettings.EnableFrameRateCounter = true;
            }
#endif

            Frame rootFrame = Window.Current.Content as Frame;

            // Neopakovat inicializaci aplikace, pokud už má okno obsah,
            // jenom ověřit, jestli je toto okno aktivní
            if (rootFrame == null)
            {
                // Vytvořit objekt Frame, který bude fungovat jako kontext navigace,
a spustit procházení první stránky
                rootFrame = new Frame();

                rootFrame.NavigationFailed += OnNavigationFailed;

                if (e.PreviousExecutionState ==
ApplicationExecutionState.Terminated)
                {
                    //TODO: Načíst stav z dříve pozastavené aplikace
                }

                // Umístit rámec do aktuálního objektu Window
                Window.Current.Content = rootFrame;
            }

            if (rootFrame.Content == null)
            {
                // Pokud není navigační zásobník obnovený, přejít na první stránku
                // a nakonfigurovat novou stránku předáním požadovaných informací ve
formě
```

```
                // parametru navigace
                rootFrame.Navigate(typeof(MainPage), e.Arguments);
            }
            // Zkontrolovat, jestli je aktuální okno aktivní
            Window.Current.Activate();
        }

        /// <summary>
        /// Vyvolá se, když selže přechod na určitou stránku.
        /// </summary>
        /// <param name="sender">Objekt Frame, u kterého selhala navigace</param>
        /// <param name="e">Podrobnosti o chybě navigace</param>
        void OnNavigationFailed(object sender, NavigationFailedEventArgs e)
        {
            throw new Exception("Failed to load Page " + e.SourcePageType.FullName);
        }

        /// <summary>
        /// Vyvolá se při pozastavení běhu aplikace. Stav aplikace se uloží
        /// bez informace, jestli se aplikace ukončí nebo obnoví s obsahem
        /// stále neporušené paměti.
        /// </summary>
        /// <param name="sender">Zdroj žádosti o pozastavení</param>
        /// <param name="e">Podrobnosti žádosti o pozastavení</param>
        private void OnSuspending(object sender, SuspendingEventArgs e)
        {
            var deferral = e.SuspendingOperation.GetDeferral();
            //TODO: Uložit stav aplikace a zastavit jakoukoliv aktivitu na pozadí
            deferral.Complete();
        }
    }
}
```

Zdroj: Automaticky generováno programem Visual Studio

Příloha E: Zdrojový kód testu přesnosti SW časovačů

Soubor MainPage.xaml

```
<!--
    Copyright(c) Microsoft Open Technologies, Inc. All rights reserved.
    The MIT License(MIT)
    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files(the "Software"), to deal
    in the Software without restriction, including without limitation the rights
    to use, copy, modify, merge, publish, distribute, sublicense, and / or sell
    copies of the Software, and to permit persons to whom the Software is
    furnished to do so, subject to the following conditions :
    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.
    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    THE SOFTWARE.
-->


<Page
    x:Class="Blinky.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:Blinky"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
            <Ellipse x:Name="LED" Fill="LightGray" Stroke="White" Width="100"
Height="100" Margin="10"/>
            <TextBlock x:Name="DelayText" Text="500ms" Margin="10"
TextAlignment="Center" FontSize="26.667" />
            <TextBlock x:Name="GpioStatus" Text="Počkejte na inicializaci GPIO..."
Margin="10,50,10,10" TextAlignment="Center" FontSize="26.667" />
        </StackPanel>
    </Grid>
</Page>
```

Zdroj: Windows Dev Center: Blinky Sample [online]. 2015 [cit. 2015-11-21]. Dostupné z: http://ms-iot.github.io/content/en-US/win10/samples/Blinky.htm

Soubor MainPage.xaml.cs

```
// Copyright (c) Microsoft. All rights reserved.

using System;
using Windows.Devices.Gpio;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Media;

namespace Blinky
{
    public sealed partial class MainPage : Page
    {
        private const int LED_PIN = 5;
        private GpioPin pin;
        private GpioPinValue pinValue;
        private DispatcherTimer timer;
        private SolidColorBrush redBrush = new
SolidColorBrush(Windows.UI.Colors.Red);
        private SolidColorBrush grayBrush = new
SolidColorBrush(Windows.UI.Colors.LightGray);

        public MainPage()
        {
            InitializeComponent();

            timer = new DispatcherTimer();
            timer.Interval = TimeSpan.FromMilliseconds(500);
            timer.Tick += Timer_Tick;
            InitGPIO();
            if (pin != null)
            {
                timer.Start();
            }
        }

        private void InitGPIO()
        {
            var gpio = GpioController.GetDefault();

            // Show an error if there is no GPIO controller
            if (gpio == null)
            {
                pin = null;
                GpioStatus.Text = "Toto zařízení neobsahuje GPIO sběrnici.";
                return;
            }

            pin = gpio.OpenPin(LED_PIN);
            pinValue = GpioPinValue.High;
            pin.Write(pinValue);
            pin.SetDriveMode(GpioPinDriveMode.Output);

            GpioStatus.Text = "GPIO piny jsou nastaveny správně.";

        }
        private void Timer_Tick(object sender, object e)
        {
            if (pinValue == GpioPinValue.High)
```

```
            {
                pinValue = GpioPinValue.Low;
                pin.Write(pinValue);
                LED.Fill = grayBrush;
            }
            else
            {
                pinValue = GpioPinValue.High;
                pin.Write(pinValue);
                LED.Fill = redBrush;
            }
        }
    }
}
```

Upraveno: Windows Dev Center: Blinky Sample [online]. 2015 [cit. 2015-11-21]. Dostupné z: http://ms-iot.github.io/content/en-US/win10/samples/Blinky.htm

Soubor App.xaml.cs

```
// Copyright (c) Microsoft. All rights reserved.

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

// The Blank Application template is documented at
http://go.microsoft.com/fwlink/?LinkId=402347&clcid=0x409

namespace Blinky
{
    /// <summary>
    /// Provides application-specific behavior to supplement the default Application
class.
    /// </summary>
    sealed partial class App : Application
    {
        /// <summary>
        /// Initializes the singleton application object.  This is the first line of
authored code
        /// executed, and as such is the logical equivalent of main() or WinMain().
        /// </summary>
        public App()
        {
            InitializeComponent();
            Suspending += OnSuspending;
        }
```

```
        /// <summary>
        /// Invoked when the application is launched normally by the end user.
Other entry points
        /// will be used such as when the application is launched to open a specific
file.
        /// </summary>
        /// <param name="e">Details about the launch request and process.</param>
        protected override void OnLaunched(LaunchActivatedEventArgs e)
        {

#if DEBUG
            if (System.Diagnostics.Debugger.IsAttached)
            {
                DebugSettings.EnableFrameRateCounter = true;
            }
#endif

            Frame rootFrame = Window.Current.Content as Frame;

            // Do not repeat app initialization when the Window already has content,
            // just ensure that the window is active
            if (rootFrame == null)
            {
                // Create a Frame to act as the navigation context and navigate to
the first page
                rootFrame = new Frame();
                // Set the default language
                rootFrame.Language =
Windows.Globalization.ApplicationLanguages.Languages[0];

                rootFrame.NavigationFailed += OnNavigationFailed;

                if (e.PreviousExecutionState ==
ApplicationExecutionState.Terminated)
                {
                    //TODO: Load state from previously suspended application
                }

                // Place the frame in the current Window
                Window.Current.Content = rootFrame;
            }

            if (rootFrame.Content == null)
            {
                // When the navigation stack isn't restored navigate to the first
page,
                // configuring the new page by passing required information as a
navigation
                // parameter
                rootFrame.Navigate(typeof(MainPage), e.Arguments);
            }
            // Ensure the current window is active
            Window.Current.Activate();
        }

        /// <summary>
        /// Invoked when Navigation to a certain page fails
        /// </summary>
        /// <param name="sender">The Frame which failed navigation</param>
        /// <param name="e">Details about the navigation failure</param>
        void OnNavigationFailed(object sender, NavigationFailedEventArgs e)
        {
```

```
            throw new Exception("Failed to load Page " + e.SourcePageType.FullName);
        }

        /// <summary>
        /// Invoked when application execution is being suspended.  Application
state is saved
        /// without knowing whether the application will be terminated or resumed
with the contents
        /// of memory still intact.
        /// </summary>
        /// <param name="sender">The source of the suspend request.</param>
        /// <param name="e">Details about the suspend request.</param>
        private void OnSuspending(object sender, SuspendingEventArgs e)
        {
            var deferral = e.SuspendingOperation.GetDeferral();
            //TODO: Save application state and stop any background activity
            deferral.Complete();
        }
    }
}
```

Zdroj: Automaticky generováno programem Visual Studio

Příloha F: Zdrojový kód informačního stánku

Soubor MainPage.xaml

```
<Page
    x:Class="IoTweb.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:IoTweb"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="65"></RowDefinition>
            <RowDefinition Height="*"></RowDefinition>
        </Grid.RowDefinitions>
        <!--Address bar-->
        <StackPanel Grid.Row="0" Orientation="Horizontal">
            <Button x:Name="button" Content="Domu" Height="61" Margin="0,2,0,0"
VerticalAlignment="Top" Width="95" RenderTransformOrigin="0.711,0.511"
FontFamily="Vladimir Script" FontSize="37.333" Background="#33020525"
BorderBrush="#FF080E44" Foreground="#FFF31414" Click="button_Click"/>
        </StackPanel>

        <!--Web view control-->
        <WebView x:Name="webView" Grid.Row="1" />
    </Grid>

</Page>
```

Zdroj: upraveno: *Novaspirit: Coding Web Browser For Windows 10 IOT on Raspberry PI 2* [online]. 2015 [cit. 2015-12-08]. Dostupné z: http://www.novaspirit.com/2015/09/22/coding-web-browser-for-windows-10-iot-on-raspberry-pi-2/

Soubor MainPage.xml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Popups;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

// Šablona položky prázdné stránky je popsaná na adrese
http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409.

namespace IoTweb
{
    /// <summary>
    /// Prázdná stránka, která se dá použít samostatně nebo v rámci objektu Frame
    /// </summary>
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
            webView.Navigate(new Uri("http://www.centrumchodov.cz/W/do/centre/mapa-
centra"));
        }
        private async void button_Click(object sender, RoutedEventArgs e)
        {
            try
            {
               webView.Navigate(new
Uri("http://www.centrumchodov.cz/W/do/centre/mapa-centra"));
            }
            catch (Exception f)
            {
                MessageDialog dlg = new MessageDialog("Error: " + f.Message);
                await dlg.ShowAsync();

            }
        }
    }
}
```

Zdroj: upraveno: *Novaspirit: Coding Web Browser For Windows 10 IOT on Raspberry PI 2* [online]. 2015 [cit. 2015-12-08]. Dostupné z: http://www.novaspirit.com/2015/09/22/coding-web-browser-for-windows-10-iot-on-raspberry-pi-2/

Soubor App.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

namespace IoTweb
{
    /// <summary>
    /// Poskytuje chování specifické pro aplikaci, které doplňuje výchozí třídu
Application.
    /// </summary>
    sealed partial class App : Application
    {
        /// <summary>
        /// Inicializuje objekt aplikace typu singleton. Jedná se o první řádek
spuštěného vytvořeného kódu,
        /// který je proto logickým ekvivalentem metod main() nebo WinMain().
        /// </summary>
        public App()
        {
            Microsoft.ApplicationInsights.WindowsAppInitializer.InitializeAsync(
                Microsoft.ApplicationInsights.WindowsCollectors.Metadata |
                Microsoft.ApplicationInsights.WindowsCollectors.Session);
            this.InitializeComponent();
            this.Suspending += OnSuspending;
        }

        /// <summary>
        /// Vyvolá se při normálním spuštění aplikace koncovým uživatelem. Ostatní
vstupní body
        /// se použijí například při spuštění aplikace za účelem otevření
konkrétního souboru.
        /// </summary>
        /// <param name="e">Podrobnosti o žádosti o spuštění a procesu</param>
        protected override void OnLaunched(LaunchActivatedEventArgs e)
        {

#if DEBUG
            if (System.Diagnostics.Debugger.IsAttached)
            {
                this.DebugSettings.EnableFrameRateCounter = true;
            }
#endif

            Frame rootFrame = Window.Current.Content as Frame;

            // Neopakovat inicializaci aplikace, pokud už má okno obsah,
```

```
            // jenom ověřit, jestli je toto okno aktivní
            if (rootFrame == null)
            {
                // Vytvořit objekt Frame, který bude fungovat jako kontext navigace,
a spustit procházení první stránky
                rootFrame = new Frame();

                rootFrame.NavigationFailed += OnNavigationFailed;

                if (e.PreviousExecutionState ==
ApplicationExecutionState.Terminated)
                {
                    //TODO: Načíst stav z dříve pozastavené aplikace
                }

                // Umístit rámec do aktuálního objektu Window
                Window.Current.Content = rootFrame;
            }

            if (rootFrame.Content == null)
            {
                // Pokud není navigační zásobník obnovený, přejít na první stránku
                // a nakonfigurovat novou stránku předáním požadovaných informací ve
formě
                // parametru navigace
                rootFrame.Navigate(typeof(MainPage), e.Arguments);
            }
            // Zkontrolovat, jestli je aktuální okno aktivní
            Window.Current.Activate();
        }

        /// <summary>
        /// Vyvolá se, když selže přechod na určitou stránku.
        /// </summary>
        /// <param name="sender">Objekt Frame, u kterého selhala navigace</param>
        /// <param name="e">Podrobnosti o chybě navigace</param>
        void OnNavigationFailed(object sender, NavigationFailedEventArgs e)
        {
            throw new Exception("Failed to load Page " + e.SourcePageType.FullName);
        }

        /// <summary>
        /// Vyvolá se při pozastavení běhu aplikace. Stav aplikace se uloží
        /// bez informace, jestli se aplikace ukončí nebo obnoví s obsahem
        /// stále neporušené paměti.
        /// </summary>
        /// <param name="sender">Zdroj žádosti o pozastavení</param>
        /// <param name="e">Podrobnosti žádosti o pozastavení</param>
        private void OnSuspending(object sender, SuspendingEventArgs e)
        {
            var deferral = e.SuspendingOperation.GetDeferral();
            //TODO: Uložit stav aplikace a zastavit jakoukoliv aktivitu na pozadí
            deferral.Complete();
        }
    }
}
```

Zdroj: Automaticky generováno programem Visual Studio

Příloha G: Zdrojový kód webového prohlížeče

Soubor MainPage.xaml

```
<Page
    x:Class="IoTweb.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:IoTweb"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="65"></RowDefinition>
            <RowDefinition Height="*"></RowDefinition>
        </Grid.RowDefinitions>
        <!--Address bar-->
        <StackPanel Grid.Row="0" Orientation="Horizontal">
            <TextBox x:Name="Web_Address" FontSize="34.667" TextWrapping="Wrap"
Text="http://www.jcu.cz" VerticalAlignment="Center"
VerticalContentAlignment="Center" Height="54" Width="958" KeyUp="Web_Address_KeyUp"
FontFamily="Vladimir Script" FontWeight="Bold"/>
            <Button x:Name="Go_Web" Content="Jdi" HorizontalAlignment="Right"
VerticalAlignment="Center" Height="60" Width="107" Click="Go_Web_Click"
FontFamily="Vladimir Script" FontSize="34.667" Foreground="#FF102774"/>
            <Button x:Name="button" Content="Domu" Height="61" Margin="0,2,0,0"
VerticalAlignment="Top" Width="95" RenderTransformOrigin="0.711,0.511"
FontFamily="Vladimir Script" FontSize="37.333" Background="#33020525"
BorderBrush="#FF080E44" Foreground="#FFF31414" Click="button_Click"/>
        </StackPanel>

        <!--Web view control-->
        <WebView x:Name="webView" Grid.Row="1" />
    </Grid>

</Page>
```

Zdroj: upraveno: *Novaspirit: Coding Web Browser For Windows 10 IOT on Raspberry PI 2* [online]. 2015 [cit. 2015-12-08]. Dostupné z: http://www.novaspirit.com/2015/09/22/coding-web-browser-for-windows-10-iot-on-raspberry-pi-2/

Soubor MainPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Popups;
using Windows.UI.Xaml;
```

```
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

// Šablona položky prázdné stránky je popsaná na adrese
http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409.

namespace IoTweb
{
    /// <summary>
    /// Prázdná stránka, která se dá použít samostatně nebo v rámci objektu Frame
    /// </summary>
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
            webView.Navigate(new Uri("http://jcu.cz"));
        }
        private void Go_Web_Click(object sender, RoutedEventArgs e)
        {
            DoWebNavigate();
        }

        private void Web_Address_KeyUp(object sender, KeyRoutedEventArgs e)
        {
            if (e.Key == Windows.System.VirtualKey.Enter)
            {
                DoWebNavigate();
            }
        }
        private async void DoWebNavigate()
        {
            try
            {
                if (Web_Address.Text.Length > 0)
                {
                    webView.Navigate(new Uri(Web_Address.Text));

                }
                else
                {
                    MessageDialog dlg = new MessageDialog("you need to enter a web
address.");
                    await dlg.ShowAsync();
                }
            }
            catch (Exception e)
            {
                MessageDialog dlg = new MessageDialog("Error: " + e.Message);
                await dlg.ShowAsync();

            }
        }

        private async void button_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                if (Web_Address.Text.Length > 0)
```

```
                {
                    webView.Navigate(new Uri("http://jcu.cz"));

                }
                else
                {
                    MessageDialog dlg = new MessageDialog("Zadejte platnou webovou
adresu.");
                    await dlg.ShowAsync();
                }
            }
            catch (Exception f)
            {
                MessageDialog dlg = new MessageDialog("Error: " + f.Message);
                await dlg.ShowAsync();

            }
        }
    }
}
```

Zdroj: upraveno: *Novaspirit: Coding Web Browser For Windows 10 IOT on Raspberry PI 2* [online]. 2015 [cit. 2015-12-08]. Dostupné z: http://www.novaspirit.com/2015/09/22/coding-web-browser-for-windows-10-iot-on-raspberry-pi-2/

Soubor App.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

namespace IoTweb
{
    /// <summary>
    /// Poskytuje chování specifické pro aplikaci, které doplňuje výchozí třídu
Application.
    /// </summary>
    sealed partial class App : Application
    {
        /// <summary>
        /// Inicializuje objekt aplikace typu singleton. Jedná se o první řádek
spuštěného vytvořeného kódu,
        /// který je proto logickým ekvivalentem metod main() nebo WinMain().
```

```
        /// </summary>
        public App()
        {
            Microsoft.ApplicationInsights.WindowsAppInitializer.InitializeAsync(
                Microsoft.ApplicationInsights.WindowsCollectors.Metadata |
                Microsoft.ApplicationInsights.WindowsCollectors.Session);
            this.InitializeComponent();
            this.Suspending += OnSuspending;
        }

        /// <summary>
        /// Vyvolá se při normálním spuštění aplikace koncovým uživatelem. Ostatní
vstupní body
        /// se použijí například při spuštění aplikace za účelem otevření
konkrétního souboru.
        /// </summary>
        /// <param name="e">Podrobnosti o žádosti o spuštění a procesu</param>
        protected override void OnLaunched(LaunchActivatedEventArgs e)
        {

#if DEBUG
            if (System.Diagnostics.Debugger.IsAttached)
            {
                this.DebugSettings.EnableFrameRateCounter = true;
            }
#endif

            Frame rootFrame = Window.Current.Content as Frame;

            // Neopakovat inicializaci aplikace, pokud už má okno obsah,
            // jenom ověřit, jestli je toto okno aktivní
            if (rootFrame == null)
            {
                // Vytvořit objekt Frame, který bude fungovat jako kontext navigace,
a spustit procházení první stránky
                rootFrame = new Frame();

                rootFrame.NavigationFailed += OnNavigationFailed;

                if (e.PreviousExecutionState ==
ApplicationExecutionState.Terminated)
                {
                    //TODO: Načíst stav z dříve pozastavené aplikace
                }

                // Umístit rámec do aktuálního objektu Window
                Window.Current.Content = rootFrame;
            }

            if (rootFrame.Content == null)
            {
                // Pokud není navigační zásobník obnovený, přejít na první stránku
                // a nakonfigurovat novou stránku předáním požadovaných informací ve
formě
                // parametru navigace
                rootFrame.Navigate(typeof(MainPage), e.Arguments);
            }
            // Zkontrolovat, jestli je aktuální okno aktivní
            Window.Current.Activate();
        }

        /// <summary>
        /// Vyvolá se, když selže přechod na určitou stránku.
```

```
        /// </summary>
        /// <param name="sender">Objekt Frame, u kterého selhala navigace</param>
        /// <param name="e">Podrobnosti o chybě navigace</param>
        void OnNavigationFailed(object sender, NavigationFailedEventArgs e)
        {
            throw new Exception("Failed to load Page " + e.SourcePageType.FullName);
        }

        /// <summary>
        /// Vyvolá se při pozastavení běhu aplikace. Stav aplikace se uloží
        /// bez informace, jestli se aplikace ukončí nebo obnoví s obsahem
        /// stále neporušené paměti.
        /// </summary>
        /// <param name="sender">Zdroj žádosti o pozastavení</param>
        /// <param name="e">Podrobnosti žádosti o pozastavení</param>
        private void OnSuspending(object sender, SuspendingEventArgs e)
        {
            var deferral = e.SuspendingOperation.GetDeferral();
            //TODO: Uložit stav aplikace a zastavit jakoukoliv aktivitu na pozadí
            deferral.Complete();
        }
    }
}
```

Zdroj: Automaticky generováno programem Visual Studio

Příloha H: Ukázkový zdrojový kód v Pythonu

StartupTask.py

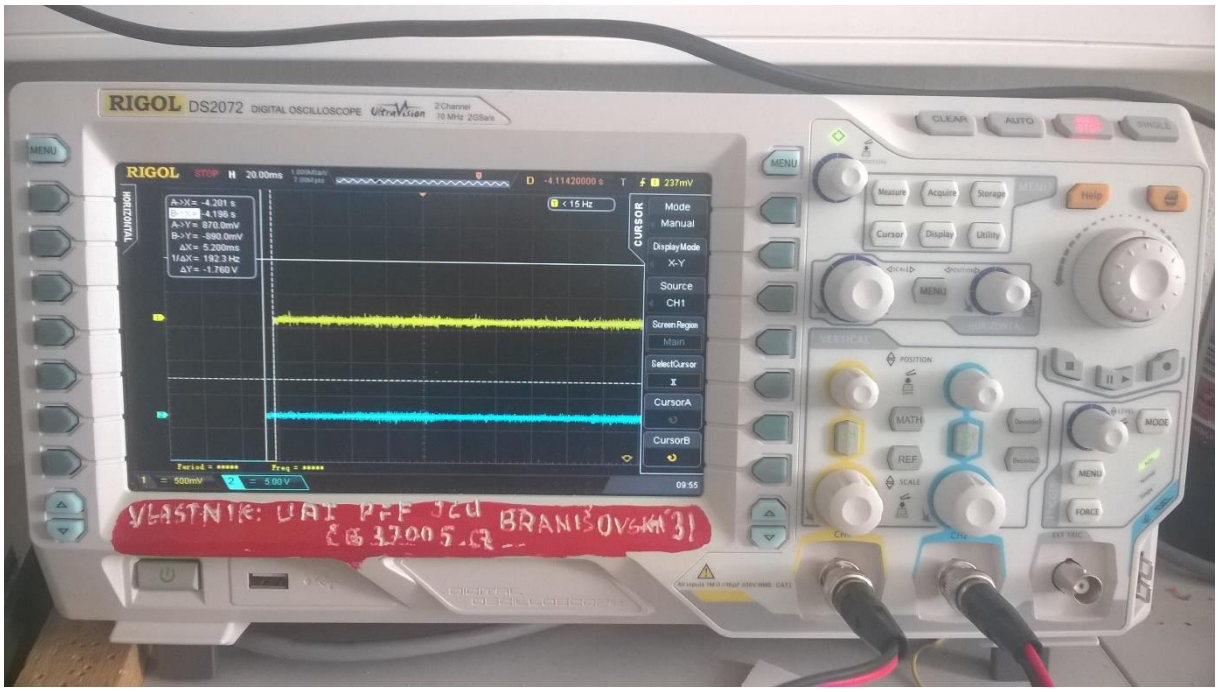
```
# Add your code here to run in your startup task
# Příklad blinky (HW Hallo World) v jazyce Python
# Účelem je pouze demonstrovat možnost psaná v Pythonu a ukázat, jak lze pomocí
Pythonu přistoupit ke sběrnici GPIO
import _wingpio as gpio #Přidání modulu pro komunikaci s GPIO
import time

led_pin = 4 #Nastavení výstupu na GPIO pin 4
pinValue = gpio.HIGH #Nastavení logické úrovně na pinu

gpio.setup(led_pin, gpio.OUT, gpio.PUD_OFF, gpio.HIGH) #Inicializace GIPO sběrnice

while True:
    if pinValue == gpio.HIGH:
        pinValue = gpio.LOW
        gpio.output(led_pin, pinValue) #Zápis hodnoty do pinu
    else:
        pinValue = gpio.HIGH
        gpio.output(led_pin, pinValue)

    time.sleep(0.5) #doba, po kterou na pinu zůstane nastavený stav zapsaný v
aktuální iteraci

       gpio.cleanup() #Vymazání konfigurace GPIO při ukončení programu
```

Upraveno: Python Blinky Sample. *Windows Dev Center* [online]. Redmond: Microsoft, 2016 [cit. 2016-07-31]. Dostupné z: https://developer.microsoft.com/cs-cz/windows/iot/win10/samples/pythonblinky

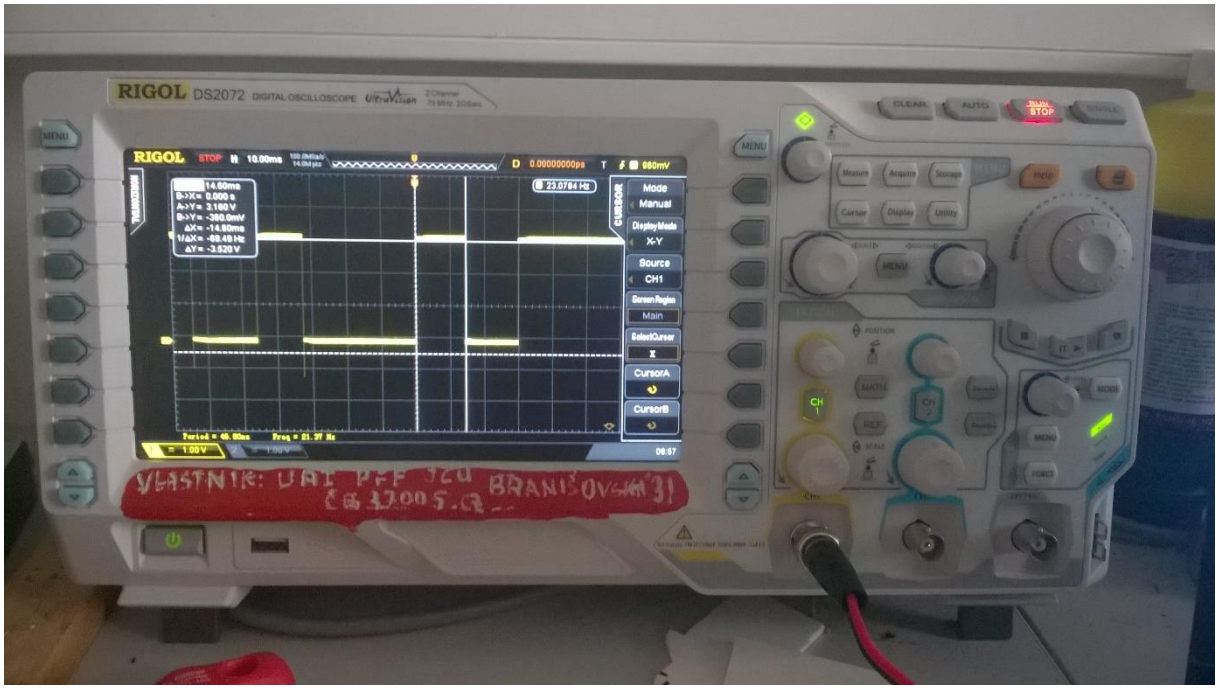
Příloha I: Zdrojový kód měření reakční doby GPIO sběrnice pro Raspbian

```
from gpiozero import LED, Button
from signal import pause
led = LED(23)
button = Button(24)
led.on()
button.when_released = led.on
button.when_pressed = led.off
pause()
```

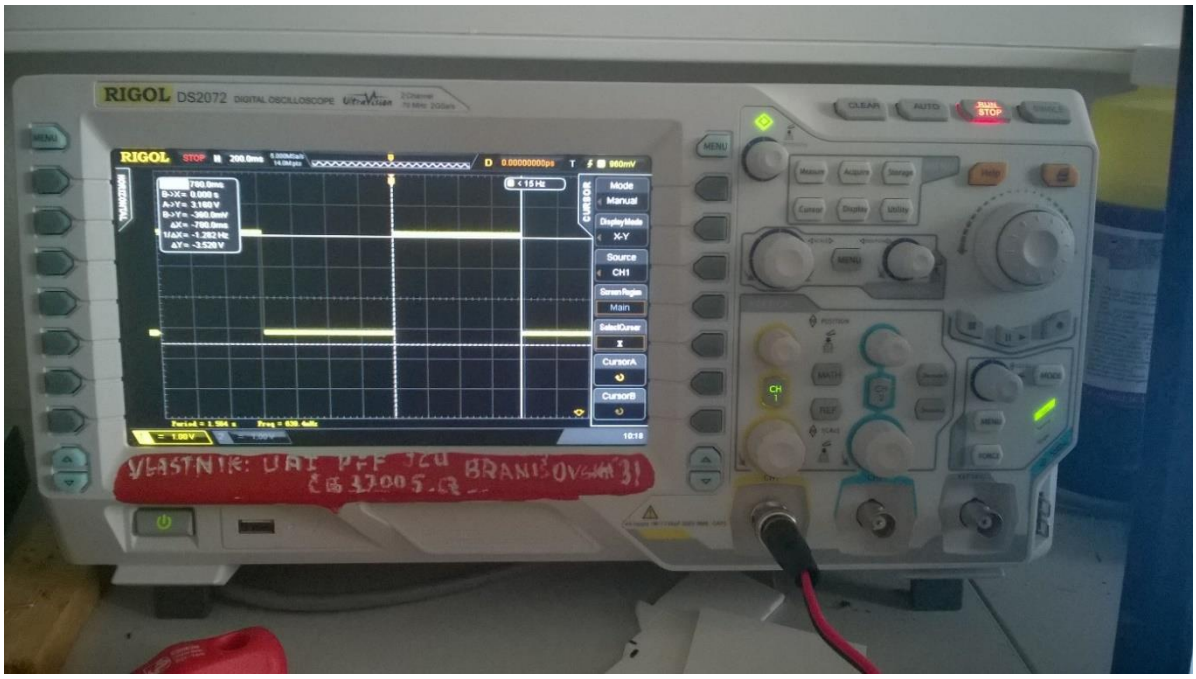
Příloha J: Obrazová dokumentace měření na GPIO sběrnici



Měření reakční doby události GPIO

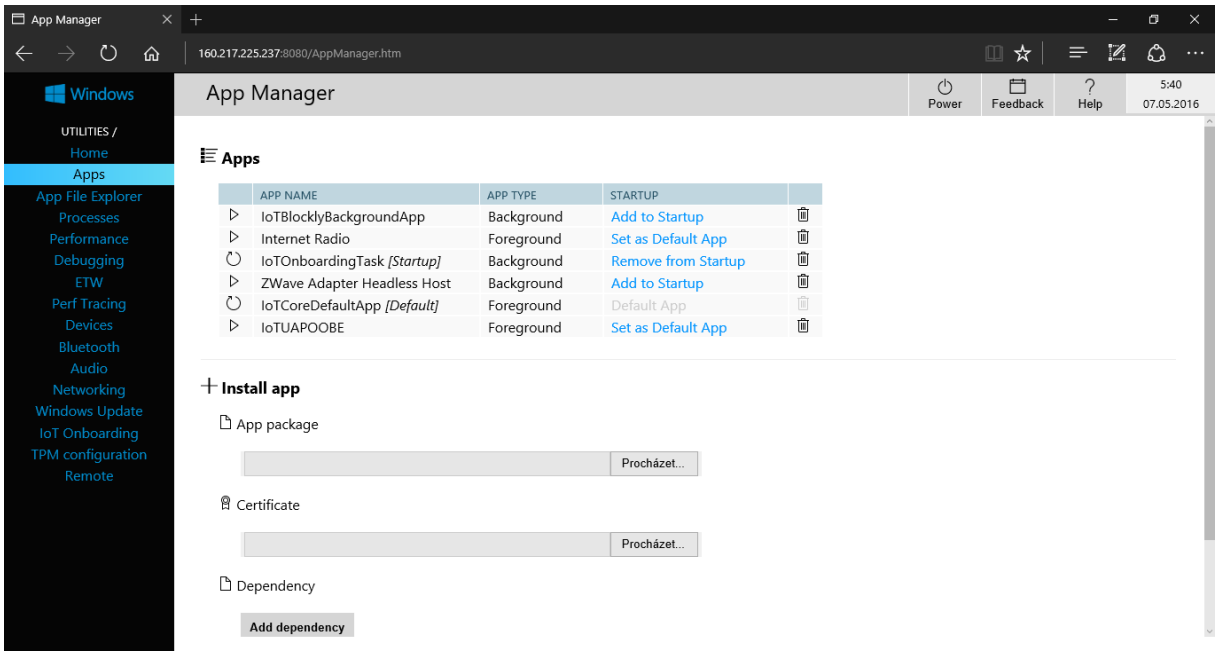


Výstupní signál z SW časovače Pi2

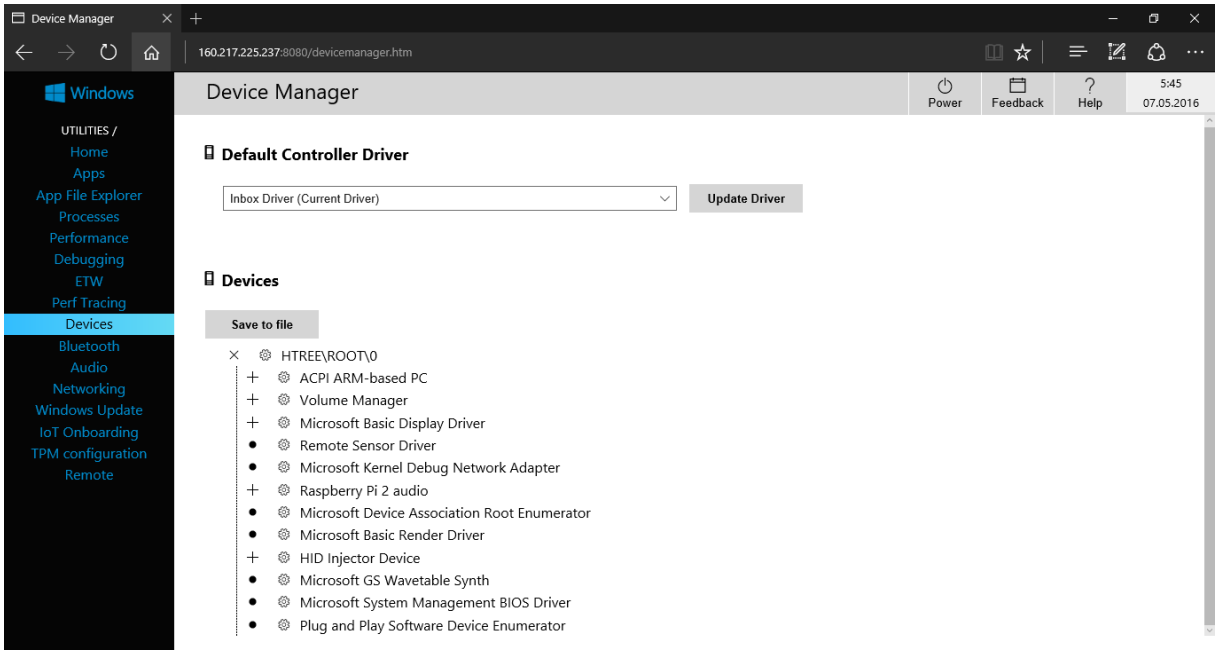


Výstupní signál z SW časovače Pi2

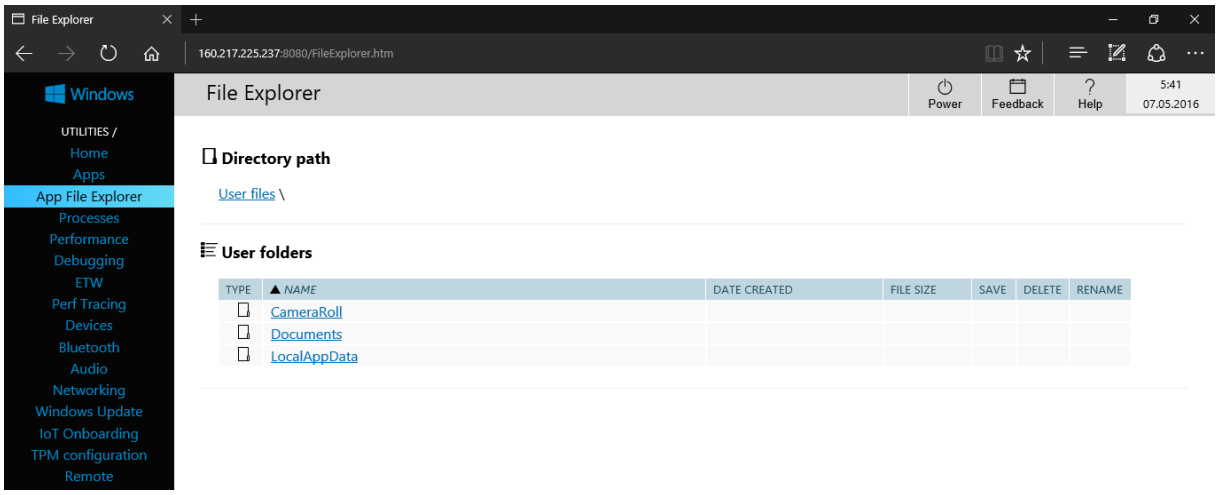
Příloha K: Obrázky novinek ve webovém rozhraní pro správu OS



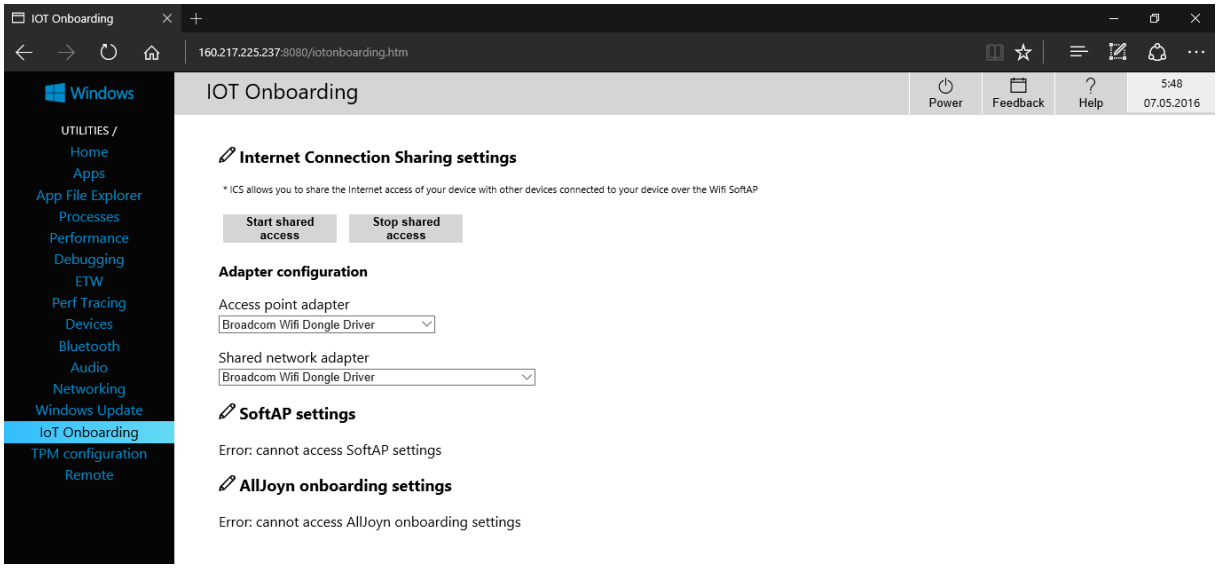
Nová zpráva programů pomocí webového rozhraní



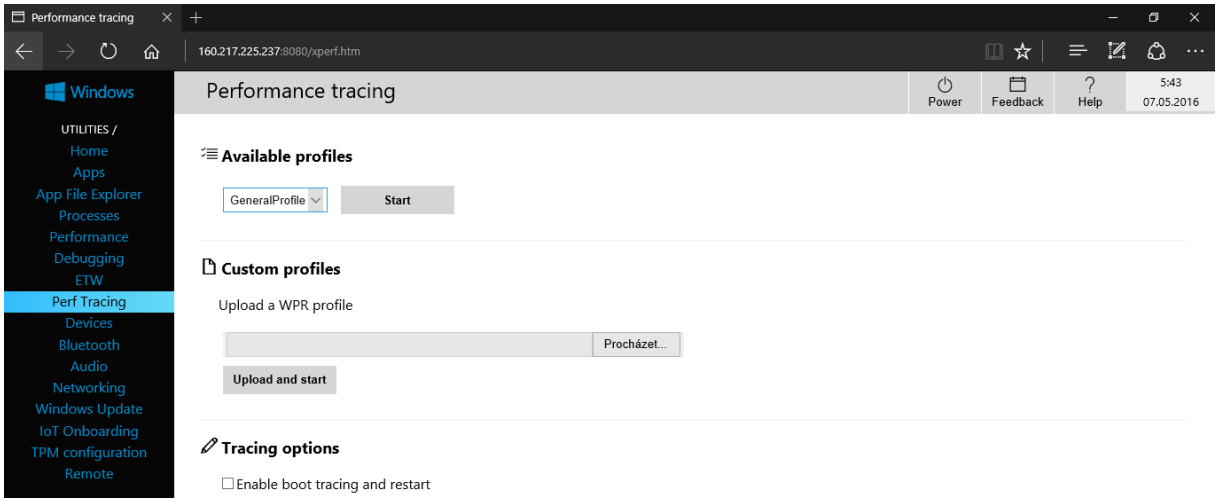
Nový Device Manager



Průzkumník souborů



Sdílení síťového připojení



Perfonmance Tracing

Příloha L: Obrázky testu práce s internetem

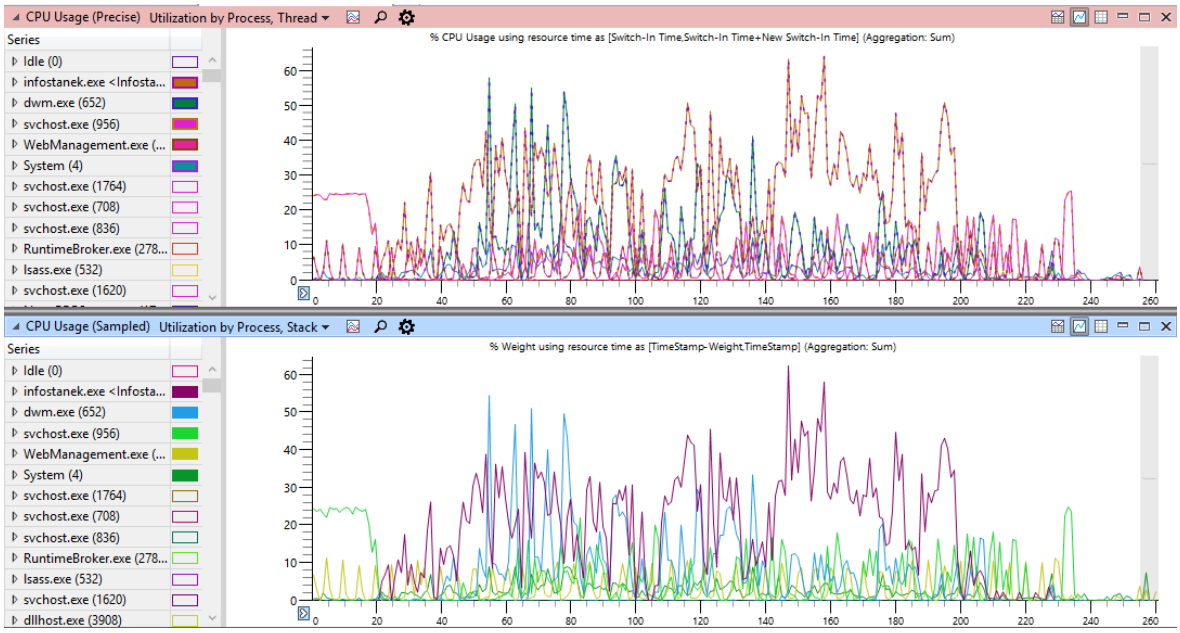
Přehrávání hudby z Youtube pomocí Windows 10 IoT Core a Pi2

Webový prohlížeč pro Windows 10 IoT Core spuštěný na Windows Server 2016 TP5

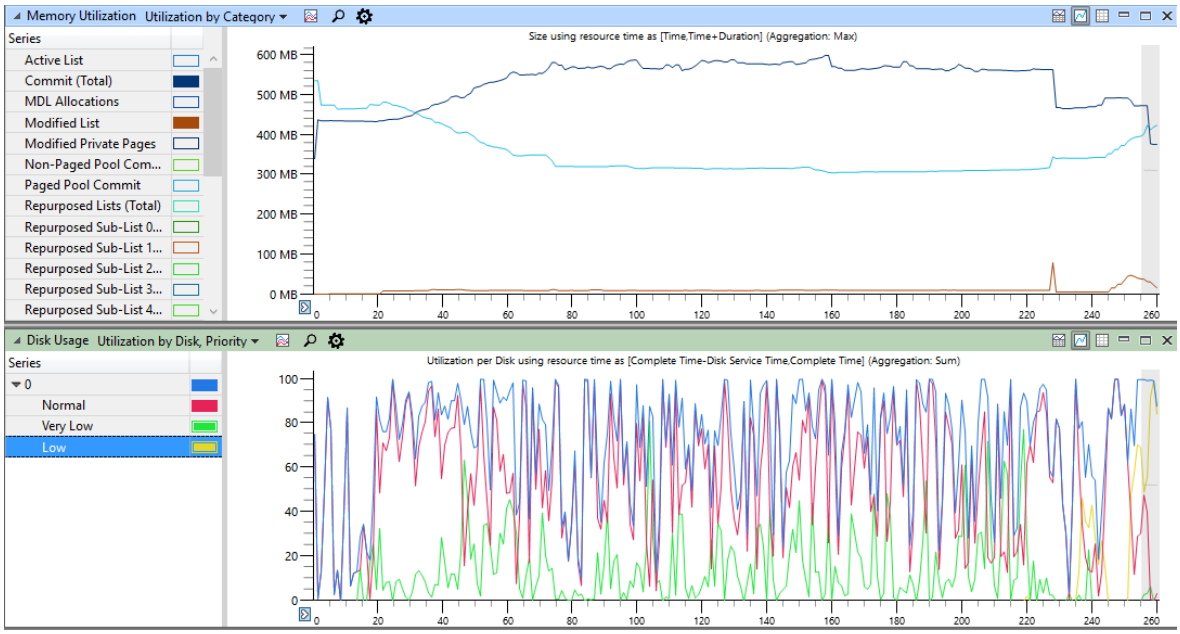
Příloha M: Postup stažení systému pro Pi3

Část postupu registrace a stažení Insider Buildu Windows 10 IoT. Zdroj: Setup with NOOBS: Using a Raspberry Pi 3: [online]. Redmnod, 2016 [cit. 2016-07-19]. Dostupné z: https://developer.microsoft.com/cs-cz/windows/iot/win10/Noobs.htm

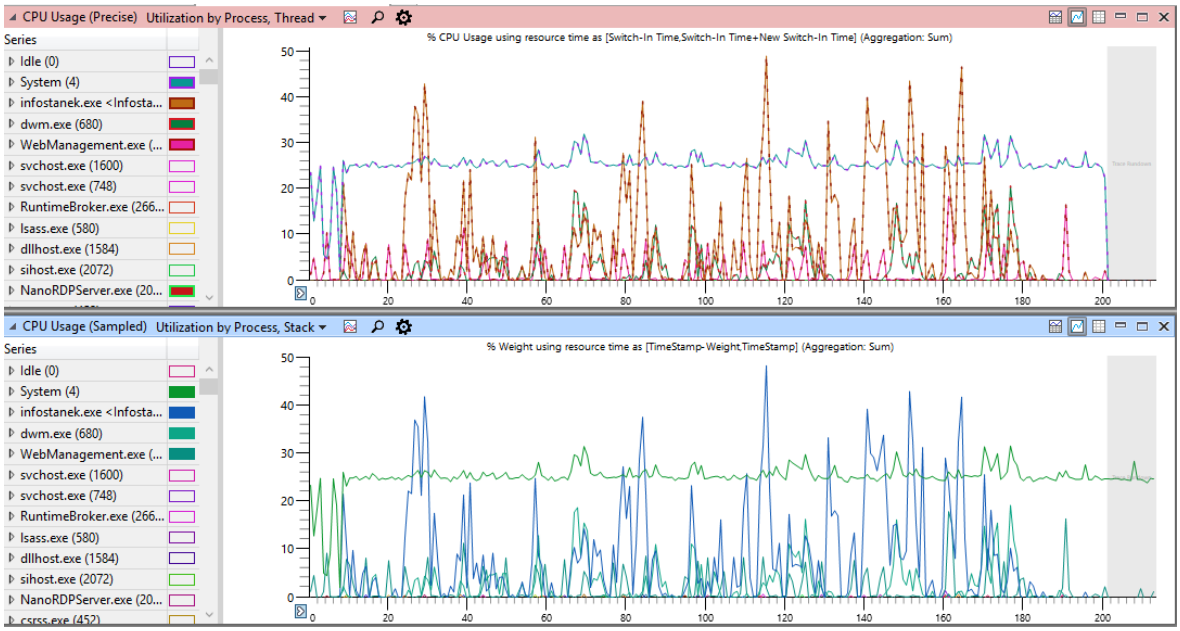
Příloha N: Grafy vytížení systému při simulaci informačního stánku



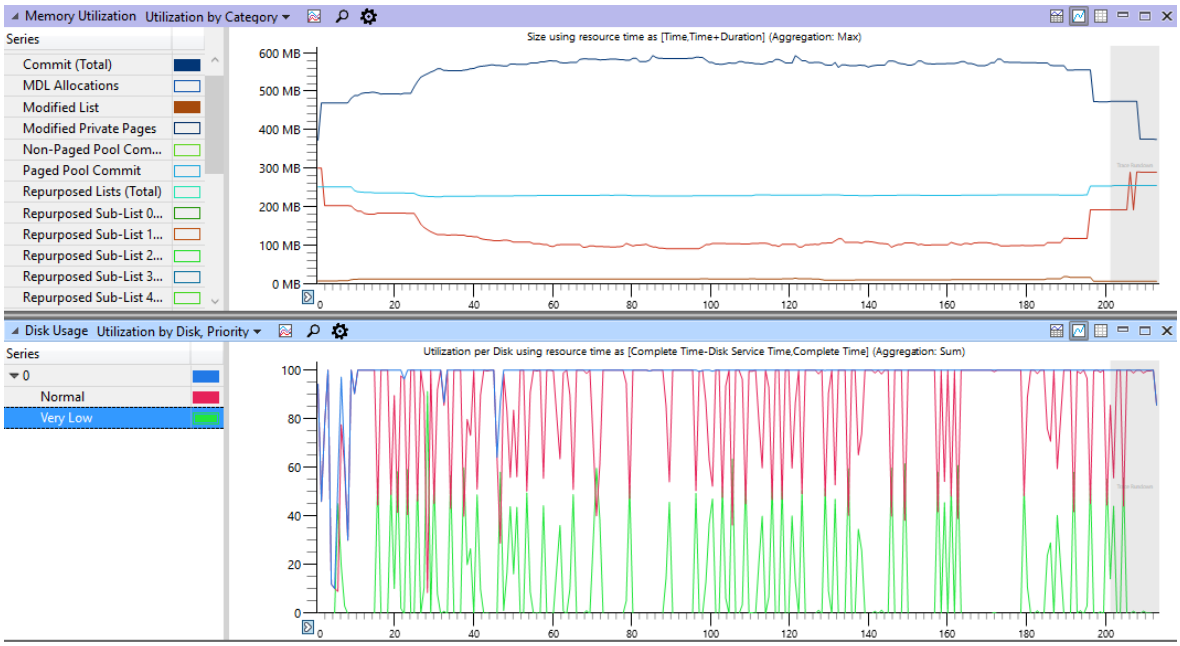
CPU trace z běhu aplikace teploměr na Pi2



Memory a Disk trace z běhu aplikace Teploměr na Pi2



CPU trace z běhu aplikace teploměr na Pi3

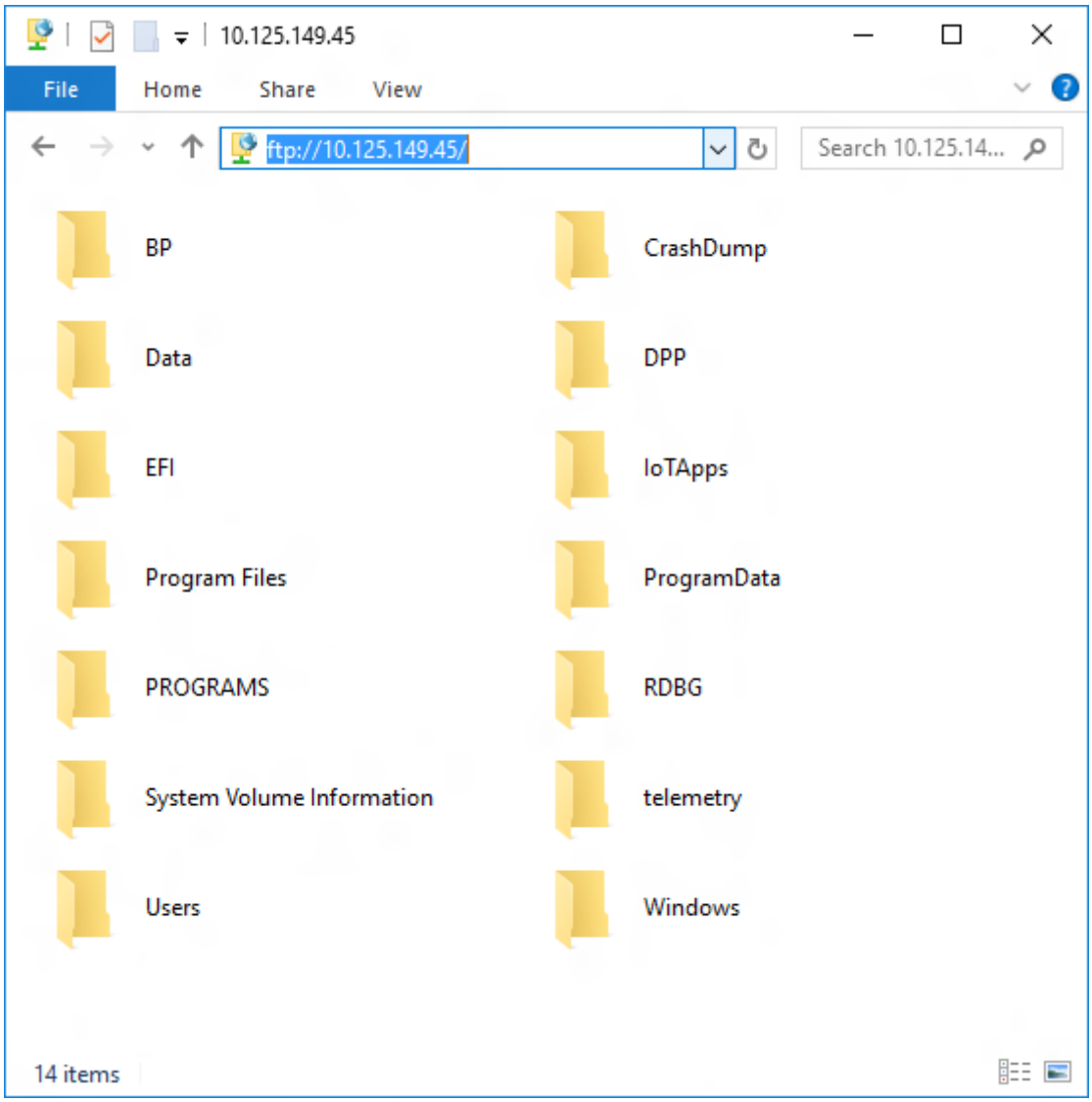


Memory a Disk trace z běhu aplikace Teploměr na Pi3

Příloha O: Práce s FTP

Pro jednorázové spuštění FTP potřebujeme použít PowerShell, kde zadáme příkaz: `start C:\Windows\System32\ftpd.exe` Pokud si nejsme zcela jisti, zda se nám podařilo FTP úspěšně spustit, tak si to ověříme pomocí příkazu: `tlist`

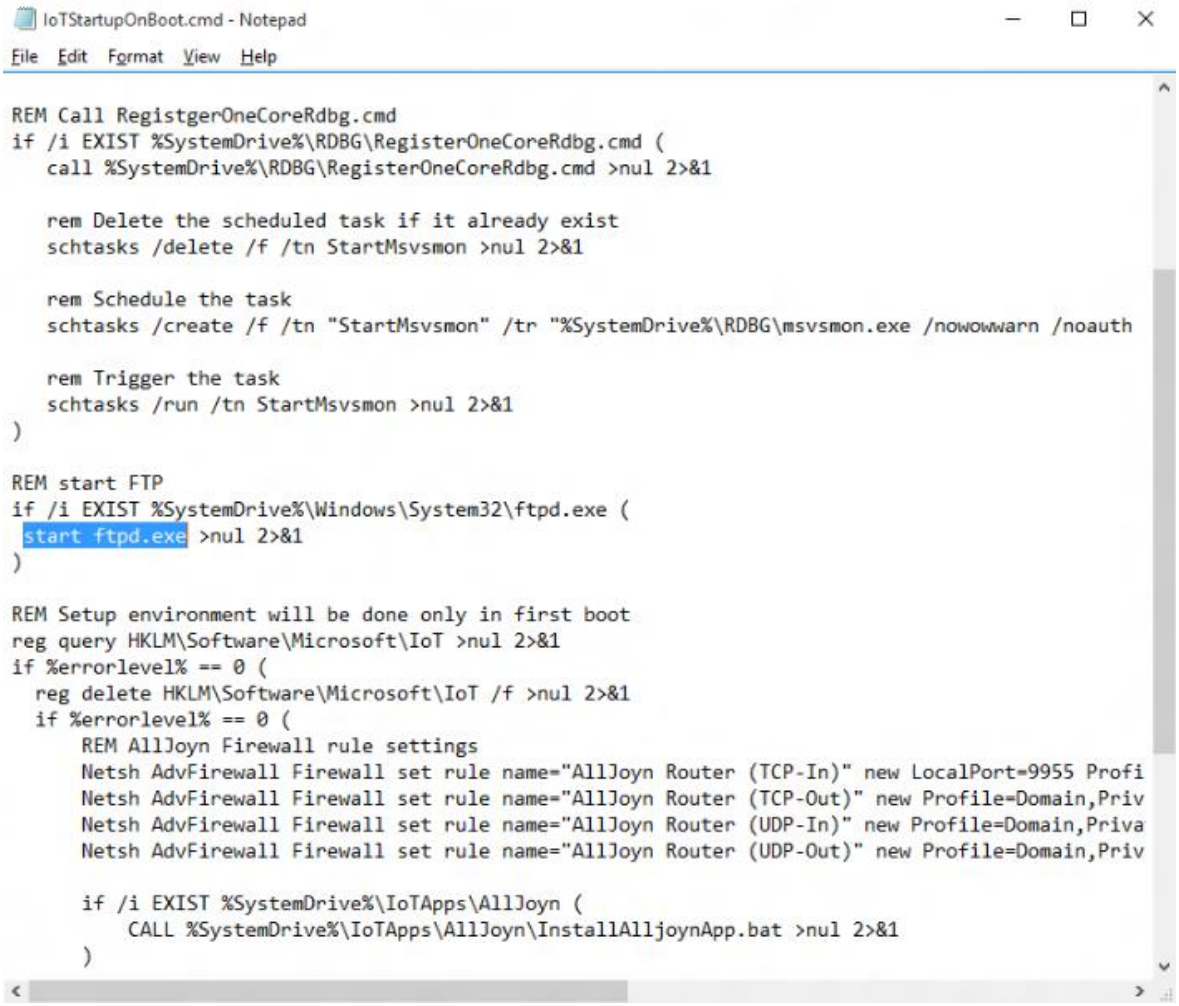
Nyní se již můžeme pomocí FTP klienta, nebo Průzkumníku Windows připojit k FTP našeho Windows IoT Core a pracovat se soubory.



Ukázkový obsah disku zpřístupněný pomocí FTP. Zdroj: File Transfer Protocol. *Windows Dev Center* [online]. Redmond: Micorosoft, 2016 [cit. 2016-11-05]. Dostupné z: https://developer.microsoft.com/en-us/windows/iot/docs/ftp

Pro opětovné vypnutí FTP slouží v PowerShellu příkaz: `kill -processname ftpd*`

Pokud chceme FTP spouštět se startem systému, musíme upravit soubor IoTStartupOnBoot.cmd dle následujícího obrázku:



Úprava souboru IoTStartupOnBoot.cmd pro automatické spouštění FTP. *Zdroj:* File Transfer Protocol. *Windows Dev Center* [online]. Redmond: Micorosoft, 2016 [cit. 2016-11-05]. Dostupné z: https://developer.microsoft.com/en-us/windows/iot/docs/ftp